Blockchain Security Evaluation with Twins

Bastien Faivre
EPFL
Lausanne, Switzerland
bastien.faivre@epfl.ch

***Abstract*—Blockchain technology, with its promise of decentralized and secure operations, has gained significant attention across various sectors. However, while extensive research and discussions have been directed towards securing *inner-blockchain* communications with solid consensus protocols, the client-node communications aspect has been relatively overlooked.**
This paper addresses this gap by presenting a novel adversarial simulation that exposes potential vulnerabilities in the client-node communication of prominent blockchains. Despite the reputed security of blockchains, we demonstrate that ignoring this aspect can have serious security implications. We showcase this through a successful attack scenario carried out on two popular blockchains - Quorum (IBFT) and Algorand. The objective of this attack is not to compromise the blockchain networks per se, but to underline the critical need for strengthened security in client-node communications.
As a solution to this vulnerability, we introduce the Byzantine Shield. This mechanism, acting as a client proxy, enhances communication security by aggregating responses from multiple nodes, thus negating the reliance on a potentially compromised single node. Our results demonstrate the effectiveness of this shield in mitigating risks, offering an additional layer of protection to the blockchain network.
However, our research also highlights the necessity for further exploration into the scalability of such defenses, particularly in high-workload scenarios due to the increased number of requests generated. With this study, we aim to contribute towards bolstering the robustness of blockchain technologies and inspire future research into securing blockchain infrastructures.

I. Introduction

Blockchain technologies have garnered substantial attention in recent years, primarily due to their potential to transform various industries. A crucial element that elevates the appeal of blockchains is their promise of enhanced security. However, it's important to note that while *inside-security* within the blockchain network is robust, no *by default* security for client-node communication is guaranteed.
Indeed, most clients and wallets interact with the blockchains through a single node, which inherently increases vulnerability to potential attacks if the node would appear to be malicious. Furthermore, many DApps (Decentralized Applications) rely on blockchain service providers such as *Alchemy*[1] or *Infura*[2]. These services act as bridges between DApps and the blockchain. Consequently, it is important to consider the potential security risks associated with this reliance on such third-party services. If these services were to be compromised, it could potentially disrupt the operation of the DApps. Hence it is also crucial for these DApps to maintain certain practices in their interactions with these service providers. This context underscores the urgency of addressing security in client-node communication within the blockchain environment, setting the stage for the focus of this study.
The rapidly evolving landscape of blockchain technologies has introduced a multitude of blockchains, each with unique characteristics and potential vulnerabilities. Critical challenge developers and users face is determining the security of these diverse blockchain solutions. In particular, the question of client-node communication security remains under-addressed, creating a considerable risk.
In response to this problem, our paper makes the following contributions:

- We introduce a **blockchain adversarial simulation** designed to scrutinize client-node communication by simulating a simple yet impactful attack. This attack simulation aims to highlight potential vulnerabilities and motivate the need for better security measures in this area.
- We propose a novel solution - the **Byzantine Shield**. This mechanism is designed to fortify client-node communication security by mitigating potential attacks, with the ultimate aim of being integrated into client and wallet software as a *by-default* feature. Additionally, we address the limitations of our proposed shield, particularly regarding scalability and transaction handling, within the scope of this contribution.

This paper is organized as follows: section 2 provides necessary background information about the idea used in the simulation. Section 3 details the design of our proposed adversarial simulation and Byzantine Shield. Section 4 elaborates on the implementation of these contributions. Section 5 presents the evaluation, including the results from testing various blockchains, the complexity of the attack, and the challenges faced by the mitigation shield. Finally, section 7 concludes the paper by summarizing our findings.

II. Background

We introduce the concept of **Twins**, as defined in the paper *Twins: BFT Systems Made Robust*[1]. This paper presents an automated unit test generator for byzantine attacks. Byzantine Fault Tolerant (BFT) protocols are designed to tolerate attacks or malfunctions within internal nodes. But creating byzantine attacks for validating these systems poses challenges. Hence, this paper introduced an approach for performing byzantine attacks and examining their behavior.

[1]https://www.alchemy.com/
[2]https://www.infura.io/

Twins produces three types of byzantine behavior: leader equivocation, double voting, and losing internal state such as forgetting *lock* guarding voted values. Rather than coding incorrect behavior, Twins uses a novel approach to create faulty behavior by duplicating correct and unmodified node behavior. It duplicates nodes to emulate these behaviors, making both duplicates appear as a single node exhibiting equivocal behavior. For instance, both duplicates can send messages in the same protocol round, but these messages will carry conflicting proposals or votes. The paper also discusses cases where one twin sends a vote in one round and the other twin *forgets* it has voted in the next round. While Twins cannot cover all possible byzantine attacks, it provides extensive coverage, capturing a broad spectrum of vulnerabilities including safety, liveness, timing, and responsiveness.

While the concept of Twins has been leveraged for *inner-blockchain* security purposes, primarily aimed at testing consensus protocols against byzantine nodes, our paper employs this concept in a different context - the realm of client-node communication. We carry out an attack in which a client communicates with the blockchain via a byzantine node. This node, aligning with the Twins principle, is, in fact, a set of two nodes with identical credentials and normal behavior, as per their implementation. However, these nodes do not belong to the same blockchain.

The bifurcation lies in the fact that one node is linked to the *real* blockchain instance, while the second node is connected to a clone *evil* blockchain. Consequently, the client unwittingly interacts with two blockchains, not realizing the split in communication. This scenario presents an opportunity to analyze the potential vulnerabilities and challenges inherent in client-node communication and the security measures needed to address them. Refer to sections III-A and III-B for more information and visual support about the attack architecture and scenario.

III. Design

In this section, we outline the design of our adversarial simulation and the Byzantine Shield, our mitigation strategy. We detail our system architecture which features twin nodes and a generic proxy to simulate non-byzantine behavior. Then, we depict a hypothetical attack scenario highlighting the potential vulnerabilities in client-node communication. Finally, we present the Byzantine Shield, designed to reduce the risks associated with reliance on a single node and enhance the security of client-node communication.

A. Architecture

Our design (shown in Figure 1) comprises two separate yet identical blockchain instances operating independently. These instances, despite running the same blockchain, can exhibit differences in the number of running nodes and the associated private/public key pairs. Notably, both instances feature a node - termed a *twin* - that mirrors the other, signifying that *Twin 1* and *Twin 2* possess identical credentials, including private/public key pairs. The distinguishing factor lies in their respective blockchain instance affiliations.

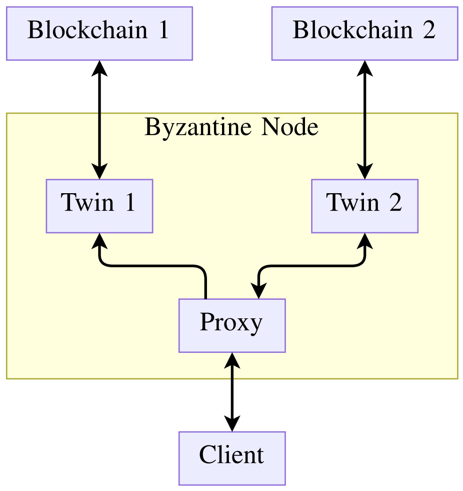

Fig. 1. Architecture of the adversarial simulation

These twins are both connected to a proxy that serves as the intermediary for client communication. The role of the proxy is to strategically divert client traffic to either *Twin 1*, *Twin 2*, or both. Conversely, when relaying messages in the opposite direction, the proxy only forwards messages from one twin at a time. This selective forwarding is a deliberate design choice, intended to maintain the facade of a normal, legitimate node in the eyes of the client. This approach allows the system to mimic the behavior of a standard, non-byzantine node.

The client interacts with the proxy under the impression that they are communicating with a legitimate node. However, the client is interacting with a byzantine node comprised of the twins and the proxy. This layered setup allows us to explore potential vulnerabilities in client-node communication within the blockchain environment.

B. Attack scenario

To underscore the potential vulnerabilities in client-node communication, we present an illustrative attack scenario (shown in Figure 2). In this scenario, Alice and Bob engage in a transaction, where Alice, the adversary, has malicious intentions.

Imagine Alice intends to buy a valuable item from Bob and plans to pay in cryptocurrency. However, Alice has an ulterior motive; she wishes to trick Bob into believing he has received payment when in fact, she has not transferred any tokens.

Now, a crucial point to consider in this attack design is the notion of genericity. In a straightforward attack scenario, Alice would simply need to modify the implementation or code of the node to carry out her deceptive operation. However, such an approach loses the genericity of the blockchain since the modification in the node implementation would be dependent on the specific blockchain.

Hence, to maintain the genericity, Alice uses the architecture defined earlier (refer to section III-A). She initiates this deceptive operation by controlling a second, entirely private blockchain. She simulates a transaction (Action 1 in Figure

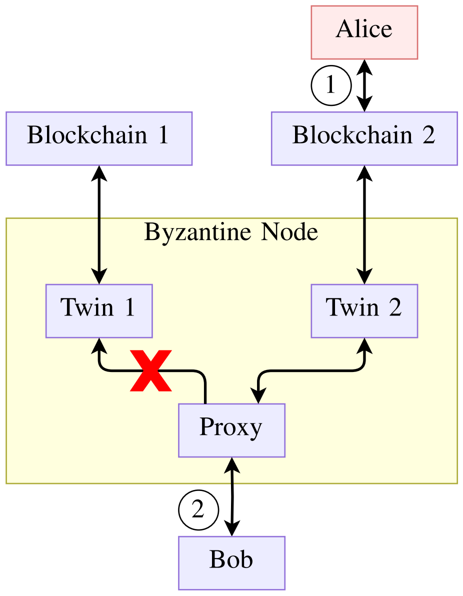

Fig. 2. Attack scenario

2) on her private blockchain to "purchase" Bob's item. When Bob checks his balance or inquires about the details of the transaction (Action 2 in Figure 2), Alice, in control of the byzantine node and hence the proxy, diverts Bob's query to her private blockchain. Note that the proxy is the key point of the attack genericity since its communication with the twins is completely independent of the blockchain.

This private blockchain carries a record of Alice's fraudulent transaction. Consequently, Bob receives a seemingly legitimate confirmation of the transaction and, believing he has received payment, sends Alice the valuable item. However, this is a cunning ruse - the transaction is non-existent on the public, trusted blockchain.

Having successfully deceived Bob, Alice redirects the communication flow back to the real public blockchain. It is at this point when Bob later verifies his balance on the public blockchain, he finds no tokens transferred from Alice. By then, it's too late - Alice has acquired the valuable item without making any payment.

Through this attack scenario, Alice has successfully exploited potential weaknesses in client-node communication, highlighting the urgent need for enhanced security measures to mitigate such risks.

C. Mitigation

The described attack and its implications illuminate critical vulnerabilities inherent in client-node communications. An intuitive solution to mitigate these potential threats is configuring the client to automatically communicate with multiple nodes, reducing the risk associated with reliance on a single, potentially byzantine, node.

Our proposed **Byzantine Shield** (shown in Figure 3), acting as a client proxy, aims to implement this solution. The design of the Byzantine Shield is aimed at seamless integration into client and wallet software as a *by-default* feature. As a result, users do not need to be aware of or concerned about this added layer of security. The client communicates exclusively with the shield, which then propagates the client's request to multiple nodes. The selection of these nodes would be handled by a configuration process within the client software. An increase in the number of contacted nodes amplifies the security of the communication, as it theoretically reduces the proportion of byzantine nodes within the set. Upon receipt of the responses, the Byzantine Shield performs an analysis and aggregates the feedback. This approach enables the shield to deliver a coherent, safe, and reliable response to the client. In doing so, the shield serves as a robust protective measure against the potential deception of byzantine nodes, enhancing the security and trustworthiness of client-node communications within the blockchain environment.

The determination of the number of nodes to be contacted by the Byzantine shield is contingent upon its aggregation method. For a more detailed discussion on this relationship and related factors, please refer to section V-D.

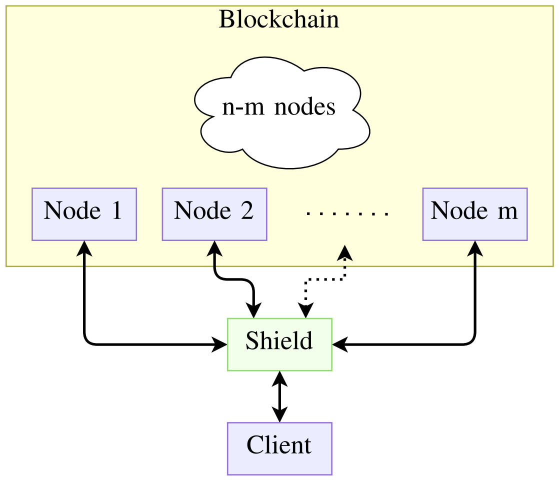

Fig. 3. Byzantine Shield between a client and a blockchain with n nodes

IV. Implementation

In this section, we delve into the specifics of our adversarial simulation and the Byzantine Shield's implementation. We begin by detailing the setup of our adversarial twin node system, outlining how we generate unique configurations for different blockchains and establish communication channels through the proxy. We also explain the client-proxy interactions and the potential for future work using the dynamic nature of our system. We then move on to the Byzantine Shield implementation, describing how it handles client communications, performs validity checks, relays messages to all nodes, and aggregates responses. We discuss the challenges faced when handling transaction-related messages and our chosen solution, highlighting the complexity and flexibility of our Byzantine Shield design.

A. Adversarial simulation

Our implementation[3] starts with the configuration generation tailored to the specific blockchain being tested; this configuration can vary between different blockchains. Initially, we establish two wallets (for Alice and Bob) with a specific fund amount, ensuring it is more significant than the blockchain's gas price, thereby enabling token transmission.

For a streamlined setup, once we have generated the blockchain's configuration, we duplicate it for the second blockchain instance and modify the associated P2P and RPC ports for each node within the configuration. In this way, we have successfully created two blockchain instances, and by implication, our twin nodes, as each node now has its counterpart by construction. We can designate one pair of nodes (twins by their credentials) as *Twin 1* and *Twin 2*, respectively. Note that some blockchains might require changes beyond port adjustments, but in our tested blockchains, modifying the ports proved sufficient.

We then initiate the proxy, which features two listening ports: one for the client and the other for configuration changes. The configuration port is utilized to receive the *flow configuration* the proxy should follow. For example, a configuration may instruct *forwards only to Twin 1 and return the response of Twin 1*, encoded in JSON. New configurations apply to future (but not currently open) connections. While this dynamic ability to alter the proxy configuration isn't used in our scenario, it may be beneficial for future related work,

When a client connects to the proxy, the necessary tunnels as specified in the configuration are set up. These tunnels merely forward incoming messages to the designated output. Each client connection is treated as a separate process, allowing the proxy to handle multiple client connections concurrently. However, this feature is not exploited in our scenario.

Lastly, Alice and Bob are not actual programs but abstract terms. Alice sends the transaction to her blockchain using the appropriate tool depending on the blockchain in question. For instance, *Geth*[4] can be used to transmit the transaction on an EVM-based blockchain, or a Python Algorand SDK[5] for the Algorand[6] blockchain. Similarly, Bob's action will depend on the blockchain used; for example, we might use *Geth* for EVM-based blockchains or JSON-RPC[7] requests for Algorand.

B. Byzantine Shield

Our Byzantine Shield[8], serving as a client proxy, establishes a listening port for client communications. Each client message is subjected to a validity check[9]; invalid messages are discarded.

[3]https://github.com/BastienFaivre/twins-attack

[4]https://geth.ethereum.org/

[5]https://github.com/algorand/py-algorand-sdk

[6]https://algorand.com/

[7]https://www.jsonrpc.org/

[8]https://github.com/BastienFaivre/byzantine-shield

[9]Our implementation currently supports blockchains compatible with the JSON-RPC format only

If the message is deemed valid, it is relayed to all nodes defined in the shield's configuration (please refer to section V-D for more details about the number of nodes to contact), preserving the original HTTP request format sent by the client. It's crucial to impose a timeout for each request to prevent the shield from waiting indefinitely for responses that might never arrive.

Upon receipt of all responses, the shield aggregates them into a single unified response, which is then relayed back to the client. The aggregation model operates on a basic principle: selecting the most frequently occurring response as the final response. This includes treating timeouts as valid responses, a key facet to ensuring system robustness. Absent this feature, an attacker could potentially disrupt communication between the shield and the honest nodes while maintaining a connection with its byzantine node, leading to misinterpretation of response validity.

However, this majority voting method does present certain challenges, particularly in handling transaction-related messages. For instance, when transmitting a transaction, if at least one node accepts it (adding it to its memory pool), the client should be informed that the transaction is being inserted into the blockchain. However, if all other nodes reject the transaction, the existing model would convey to the client that the transaction was declined, which isn't accurate since one node accepted it. We found it challenging to develop a generic solution to this problem.

Our resolution for this issue is to allow the configuration of the shield with a list of methods. When a client sends a message containing one of these methods, the shield refrains from aggregating the response based on the most frequently occurring response. Instead, it returns a single response, adhering to the JSON-RPC format, that summarizes all received responses. This ensures the client is aware of the acceptance of the transaction by at least one node. It also prevents the client from re-submitting the same transaction, mistakenly believing the initial transaction failed. This can avoid unintended double transactions.

V. Evaluation

In this section, we evaluate the effectiveness of our proposed attack simulation and the performance of our proposed Byzantine Shield. We begin by examining how our attack scenario performed against two popular blockchains: Quorum (IBFT) and Algorand, and the results confirm the attack's success in both instances. We then delve into the complexity of our test setup and assess the practicality of reproducing the attack. Our analysis of the Byzantine Shield covers its performance, compatibility, as well as its unique challenges. We also examine the implications of our shield on system scalability, underscoring the significance of our research and the potential areas for further exploration.

A. *Results*

In our evaluation process, we chose to test our attack scenario against two different blockchains: Quorum (IBFT)[10] and Algorand[11]. Both these blockchains were selected based on their distinctive characteristics and wide usage.
Quorum, based on the Ethereum platform, uses Istanbul Byzantine Fault Tolerance (IBFT) consensus algorithm. IBFT, a robust consensus mechanism, provides transaction finality, preventing forks in the blockchain. Despite this, our adversarial simulation was successful in exploiting client-node communication vulnerabilities.
On the other hand, Algorand, a high-performance blockchain that uses a Pure Proof-of-Stake (PPoS) consensus protocol, was no exception. Even with its innovative design, aimed to address blockchain trilemma - scalability, security, and decentralization - our attack scenario managed to deceive the client by providing misleading information.
In both cases, the client could not realize that the response they received was not based on the actual state of the blockchain (refer to section III-B for context). This highlights the susceptibility of current blockchains to deceptive manipulation in client-node communication and underscores the necessity for enhanced security measures in this aspect of blockchain interaction.

B. *Complexity*

Our evaluation test can be divided into two main sections: the setup of the test architecture, specifically the setting up of the blockchain instances, and the execution of the attack itself.
The setup phase is blockchain-dependent, and its duration heavily relies on the complexity of the blockchain setup process. For Quorum, it took us approximately 15 hours to establish a robust setup that allowed us to:

- Install the blockchain on any new machine.
- Generate the blockchain configuration, including the wallet initialization.
- Start and stop the blockchain.

For Algorand, as we already had the base scripts in place, we only needed to adapt them to fit the blockchain's specifics, taking us about 8 hours. It's worth noting that the duration of this process largely depends on the quality of the documentation provided by the blockchain. However, once the setup is done, it only takes a few minutes to reproduce the entire setup on new machines, as we simply need to execute the scripts we've implemented.
The second part of our test - the attack execution - only involves two requests: a transaction and a query. These requests are made manually, taking roughly 5 minutes to write initially. Once these requests are established, it takes a matter of seconds to execute them.
Overall, we found that the initial setup process can be time-consuming, taking several hours. However, once everything is set up, reproducing the attack from scratch on new machines is a matter of 5 minutes. Looking forward, we estimate that testing a new blockchain would require approximately 8 hours for both the setup and testing phases. This equates to one working day to read the blockchain documentation, adapt the scripts, and perform the test.

[10]https://consensys.net/quorum/
[11]https://algorand.com/

C. *Byzantine Shield mitigates the attack scenario*

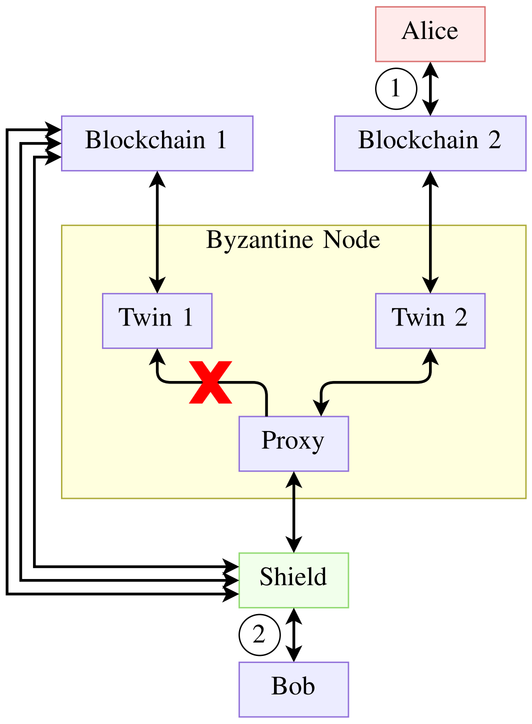

Fig. 4. Attack scenario with Byzantine Shield

To illustrate the effectiveness of the Byzantine Shield, we revisit the earlier illustrative attack scenario (section III-B), this time with Bob wisely employing the Shield for his transactions.
Reconsider Alice and Bob's transaction situation, where Alice plans to trick Bob into believing he has received payment for a valuable item when she has not transferred any tokens. Now, with Bob using the Byzantine Shield (as shown in Figure 4), the sequence of events alters significantly.
As previously, Alice initiates her deceptive operation by controlling a secondary, entirely private blockchain. She simulates a transaction on her private blockchain (Action 1 in Figure 4) to *purchase* Bob's item. Now, Bob, using the Byzantine Shield, queries his balance or transaction details (Action 2 in Figure 4). The Shield, instead of sending the query to a single node, disperses the request to multiple nodes, including both Alice's Byzantine node and honest nodes in the public blockchain.
Since Alice's Byzantine node still provides fraudulent transaction details, the shield receives one seemingly legitimate confirmation of the transaction from Alice's Byzantine node. However, the honest nodes on the public blockchain provide no record of this transaction. The Byzantine Shield, designed to select the most frequently occurring response, disregards

the response from Alice's Byzantine node since it is the minority. The Shield then returns the majority response to Bob, accurately showing no record of Alice's transaction.
Bob, therefore, realizes he has not received any payment from Alice, and wisely refrains from sending the valuable item. The Byzantine Shield has successfully mitigated Alice's attempted attack.

D. Byzantine Shield analysis

As detailed in the implementation of the shield in Section IV-B, we initially emphasized that the shield only supports blockchains compatible with the JSON-RPC format. Some methods, especially those related to transactions, need to be handled differently due to the intricacies and variability of transaction requests and responses. We found it challenging to devise a generic solution to this problem and our current approach involves manually marking such methods for different handling.
Turning to scalability, the request execution time through the shield is constrained by the slowest responding node, with an upper bound set by the timeout value. Thus, the shield's performance heavily depends on the responsiveness of the contacted nodes and the shield's settings. This aspect could impact the client's performance at a large scale during intensive usage.
Reflecting on our aggregation model, it becomes evident that the ideal number of nodes the shield should contact is intrinsically linked to the number of byzantine nodes the blockchain can tolerate. If a blockchain tolerates f byzantine nodes, the shield should ideally contact at least $2f+1$ nodes to minimize the byzantine nodes' impact on client-node communication. This serves as a preliminary mitigation strategy, but it does not fully negate the potential for byzantine attacks.
Indeed, even with this choice of contacting $2f + 1$ nodes, it's important to note that the expected outcomes may not always align with reality. This assumption is premised on the belief that the other $f + 1$ nodes would all respond identically. If all byzantine nodes coordinate to provide the same response, and the non-byzantine nodes do not provide identical responses, the shield, with its current aggregation model, will inadvertently accept the malicious response. Thus, the apparent simplicity of the $2f + 1$ choice hides a more complex dynamic that calls for additional exploration.
Finally, the assumption that each client's request generates at least $2f + 1$ *real* requests on the blockchains has significant implications. If every client uses the shield, this scenario could lead to a substantial increase in requests, which poses potential scalability challenges for the blockchains. As highlighted in a recent study [2], many blockchains are not yet capable of handling the workloads experienced by centralized applications. Consequently, the introduction of a mitigation tool that increases the number of requests adds a considerable load to the system, necessitating further research to address this issue without compromising the system's overall security and functionality.

VI. Conclusion

This paper unveils a unique adversarial simulation that successfully exposes vulnerabilities in the client-node communication of prominent blockchains, Quorum (IBFT), and Algorand. To mitigate these risks, we introduce the Byzantine Shield, a client proxy that enhances security by aggregating responses from multiple nodes, thus negating a reliance on a single potentially byzantine node.
While the setup for our attack test can be complex, the significant security enhancements provided by the Byzantine Shield underscore its importance. Our research also highlights the need for further investigation into the scalability of such defenses, considering the increased number of requests generated in high-workload scenarios.
Through our contributions, we hope to enhance the robustness of blockchain technologies and pave the way for future research into securing blockchain infrastructures.

References

[1] Shehar Bano, Alberto Sonnino, Andrey Chursin, Dmitri Perelman, Zekun Li, Avery Ching, and Dahlia Malkhi. Twins: Bft systems made robust. *arXiv preprint arXiv:2004.10617*, 2020.
[2] Vincent Gramoli, Rachid Guerraoui, Andrei Lebedev, Chris Natoli, and Gauthier Voron. Diablo: A benchmark suite for blockchains. pages 540–556, 2023.