Blockchain Trilemma Solver Algorand has Dilemma over Undecidable Messages

Mauro Conti
conti@math.unipd.it

Ankit Gangwal*
ankit.gangwal@phd.unipd.it

Michele Todero†
michele.todero@studenti.unipd.it

ABSTRACT

A variety of solutions, e.g., Proof-of-Work (PoW), Proof-of-Stake (PoS), Proof-of-Burn (PoB), and Proof-of-Elapsed-Time (PoET), have been proposed to make consensus mechanism used by the blockchain technology more democratic, efficient, and scalable. However, these solutions have a number of limitations, e.g., PoW approach requires a huge amount of computational power, scales poorly, and wastes a lot of electrical energy. Recently, an innovative protocol called Algorand has been proposed to overcome these limitations. Algorand not only guarantees an overwhelming probability of linearity of the blockchain, but it also aims to solve the "blockchain trilemma" of decentralization, scalability, and security.

In this paper, we present a security analysis of Algorand. To the best of our knowledge, it is the first security analysis as well as the first formal study on Algorand. We designed an attack scenario in which a group of malicious users tries to break the protocol, or at least limit it to a reduced partition of network users, by exploiting a security flaw in the messages validation process of the Byzantine Agreement (BA). Since the source code or an official simulator for Algorand was not available at the time of our study, we created a simulator (which is available on request) to implement the protocol and assess the feasibility of our attack scenario. Our attack requires the attacker to merely have the trivial capability of establishing multiple connections with targeted nodes, and it costs practically nothing to the attacker. Our results show that it is possible to slow down the message validation process on honest nodes - which eventually forces them to select default values on the consensus - leaving the targeted nodes behind in the chain as compared to the non-attacked nodes. Even though our results are subject to the real implementation of the protocol, the core concept of our attack remains valid.

CCS CONCEPTS

• **Security and privacy → Distributed systems security**; **Network security**; • **Networks → Network algorithms**;

KEYWORDS

Altcoin, Bitcoin, Blockchain, Consensus.

1 INTRODUCTION

With the introduction of Bitcoin [23] in 2008, blockchain technology has been adopted by almost every scenario in today's digital world with an aim to keep temper-proof records [10]. Originally, the concept of distributed consensus - upon which the blockchain technology is founded - dates back to the '90s [12, 14, 16, 34]. But, after the success of Bitcoin, a multitude of other areas have been transformed by the idea of a transparent and distributed public ledger, such as elections, contract management, insurance, charity, and ownership transfer through inheritance and will [4].

The Bitcoin system combines together different well-known concepts including digital signatures, hashing functions, PoW, and Merkle trees. However, with an exponential growth in the number of users and Bitcoin-miners, the Bitcoin network observed collateral effects of its consensus protocol. In particular, it suffers from scalability [11], hashing power centralization, mining strategies exposures [13, 21, 30], and energy- and computational-deficiency [24]. To this end, a variety of solutions have been proposed over time. Some of these solutions aim to improve the existing version of PoW (e.g., Litecoin[1], Primecoin[2] [18]) while other proposals used completely different concepts, such as PoS, PoB [25, 32], and PoET [3]. As a representative example, PoS systems (both stake modification-based [1, 19, 27, 29] and actual balance-based [2, 35]) have critical concerns from the security point of view [20]; such issues have been partially solved by providing formally-proved security guarantees [6, 17] or by discouraging users to act maliciously through punitive procedures [7, 8]. Other works [28, 31] tried to achieve better scalability and throughput by implementing a block graph structure based on Directed Acyclic Graph (DAG), albeit, these solutions have their own demerits.

On the other side, Algorand [9, 15] - an innovative protocol for the blockchain technology - has been proposed to overcome these limitations. Algorand uses a process, called "cryptographic sortition," to securely and unpredictably elect a set of voters from the network periodically. These voters are responsible for reaching consensus through a Byzantine Agreement (BA) protocol, one block at a time. It guarantees an overwhelming probability of the blockchain's linearity and a block generation time of nearly a minute. It is comparable to the approaches presented in [6, 17] and has the potential to shape the future of the blockchain technology.

Motivation: Given the promising properties of the Algorand protocol in terms of decentralization and scalability, the security aspects of its consensus algorithm are crucial. To the best of our knowledge, our work is the first analysis of Algorand's security. We aim to at least start a discussion about a possible security flaw on the message validation process, which could possibly expose honest nodes' bandwidth and memory resources to Distributed Denial-of-Service (DDoS) attacks.

* *Corresponding author.*
† *This work is an outcome of Michele Todero's M.Sc. thesis.*

Mauro Conti and Ankit Gangwal are with the Department of Mathematics, University of Padua, Italy. Michele Todero is with PwC Advisory SpA Italy - Cyber Security.

[1] Litecoin adopted its *scrypt* PoW function's design to increase the decentralization of the system and further optimize the mining process [26].
[2] Primecoin exploits the computational power used for the mining activity to contribute to mathematical research works.

Contribution: The major contributions of our work are as follows:

(1) We demonstrate a practically feasible attack on the Algorand protocol. The attack has the potential to target an arbitrarily sized group of honest users and slow down the consensus process; leaving the targeted nodes behind in the chain as compared to the non-attacked nodes.
(2) We implemented the protocol in our simulator created in Java and evaluated the feasibility of our attack. The simulator is available on request.

Organization: The rest of the paper is organized as follows: Section 2 gives a thorough description of the Algorand protocol. We elucidate our attack scenario in Section 3. Section 4 presents the implementation details of our simulator, evaluation settings, and our results. Section 5 discusses the feasibility of our attack and the possible countermeasures. Finally, Section 6 concludes the paper by listing potential future works.

2 ALGORAND

Since the main focus of our work is on Algorand, we first introduce the Algorand protocol. We elucidate the main limitations of current blockchain technologies in Section 2.1 and present the distinct features of Algorand in Section 2.2. Section 2.3 discusses the assumptions specified by the Algorand protocol with respect to the adversary model. Section 2.4 gives an overview of the network topology, communication protocol, and the description of messages used by Algorand. Then, consensus mechanism of Algorand is described in Section 2.5. Finally, the technical details of cryptographic sortition, a critical component of the consensus process, are discussed in Section 2.6.

2.1 The limitations of current Blockchain

The main motivation for the development of Algorand lies in the underlying assumption and technical problems that are essentially shared by most of the PoW-based blockchains [9]. Those assumption and limitations include:

(1) [Assumption] Majority of nodes contributing to the computational power of the network will be honest.
(2) [Technical Problem] Wastage of computational power as well as electrical energy.
(3) [Technical Problem] Concentration of power, i.e., the total computing power for block generation lies within just few mining pools.
(4) [Technical Problem] Ambiguity in the consensus reached by different network nodes on the confirmed transactions, which leads to fork of blockchain.

2.2 Salient features of Algorand

The protocol has distinct and promising features:

(1) It is designed to be fully decentralized and democratic. Moreover, there is no distinction between the role of different groups of users, e.g., miners vs. "normal" users.
(2) Each user runs the same functions with negligible hardware requirements (i.e., with negligible computational effort) as the concept of miner/mining does not exist in Algorand.
(3) It is scalable with the number of nodes as well as in terms of confirmed transactions throughput. The authors [15] have shown that the throughput of Algorand is fifty times that of the Bitcoin system.
(4) The probability of blockchain forking is practically zero ($P_{fork} = 10^{-12}$).

2.3 Protocol assumptions

To guarantee security of the blockchain, Algorand relies on the conventional assumption of PoS systems, i.e., Honest Majority of Money (HMM). In particular, Algorand assumes a continuum in the ownership of currency belonging to honest users, i.e., at any round r at least a fraction h (greater than two-third) of the money held by honest accounts on round $r - k$ must still belong to the honest users, for some integers h and k that are defined by the protocol. Together with HMM, Algorand requires a partial synchronization assumption on the honest nodes' clocks. It states that the honest nodes' clocks are not required to be strictly synchronized, but they are assumed to have the same speed with a bounded tolerance.

Furthermore, Algorand was built to face a very strong adversary. The adversary is capable to perfectly synchronize all malicious nodes that he[3] controls and can corrupt honest users indiscriminately and instantly. However, the attacker's capabilities are bounded by the following three main facts: (1) he can corrupt honest users until the HMM property is maintained; (2) he cannot forge secret keys of the honest users that he have not corrupted; and (3) he cannot prevent the honest recipient to receive messages.

2.4 Network communication

The Algorand network is a Bitcoin-like peer-to-peer network. Each user is identified by a public/private key pair, and each key pair corresponds to a node in the topology. For simplicity, we will use the terms "user" and "node" interchangeably. Each node establishes and maintains a different TCP connection with each of his peers. The current specifications of Algorand do not specify if a node has to choose his peers based on their stake, which indeed exposes the network to a Sybil attack[4].

Nodes broadcast messages through the network using a standard gossip protocol - each node gossips his message to his peers, who in turn relays it to their neighbors. A message has to be validated before it can be relayed, and it is never sent twice to the same user. The protocol defines four types of messages: (1) transactions; (2) voting messages; (3) block proposals; and (4) credential messages. A transaction (t) from user i to user j is defined as:

$$t = sig_i(pk_i, pk_j, a, I, H(SI)), \tag{1}$$

where pk_u is the public key of the user u, a stands for the amount of Algorand units to transfer from pk_i to pk_j, I is an optional string to describe the transfer, $H(SI)$ is a 256-bit hash produced by hashing a secret string SI (not specified in the body of the message), and $sig_u(x)$ denotes digitally signed data x using the private key of user u.

[3] In this paper, we have mentioned different entities as a masculine entity without any gender-bias.
[4] In a sybil attack, the attacker creates a large number of nodes at zero cost (i.e., with no stake) to increase the probability of connecting with his targets.

A voting message (vm) from a user i in round r and step s is defined as:

$$vm_i^{r,s} = (sig(v_i^{r,s}), \sigma_i^{r,s}), \tag{2}$$

where $\sigma_i^{r,s}$ represents the sortition proof of the user i in s^{th} step of r^{th} round (further details in Section 2.6), $v_i^{r,s}$ defines the vote of user i in round r and step s.

A block proposal (bp) for round r from user i is defined as:

$$bp_i^r = (B_i^r, sig_i(H(B_i^r)), \sigma_i^{r,1}), \tag{3}$$

where B_i^r is defined as

$$B_i^r = \begin{cases} (r, PAY^r, sig_i(Q^{r-1}), H(B^{r-1})), & \text{if } B_i^r \neq E^r; \\ (r, Q^{r-1}, H(B^{r-1})), & \text{otherwise;} \end{cases} \tag{4}$$

where PAY^r is the set of transactions t, while Q^r is called *seed* and defined as

$$Q^r = \begin{cases} H(sig_{l^r}(Q^{r-1}, r)), & \text{if } B^r \neq E^r; \\ H(Q^{r-1}, r), & \text{otherwise;} \end{cases} \tag{5}$$

where l^r is the user who created the consensus block B^r for round r. The seed is a parameter needed in the sortition process (more details in Section 2.6). E^r is the default block for round r, and it is defined as

$$E^r = (r, Q^{r-1}, H(B^{r-1})). \tag{6}$$

Finally, the credential message (cm) from a user i for a round r is defined as:

$$cm_i^r = (\sigma_i^{r,1}). \tag{7}$$

It consists of the sortition proof of the block proposer (in this case, user i). Each sortition proof is associated with a priority value in a deterministic way (further details in Section 2.6). Credential messages are far smaller in size than the block proposals. Thus, they propagate faster through the network. Credential messages are gossiped by the block proposers at the beginning of a round along with their block proposals. Since the block proposals and the credential messages from the same block proposer contain the same sortition proof, the peer nodes can leverage the priority values from the credential messages to not relay block proposals with low priorities - which helps in preventing congestion in the network.

2.5 Consensus algorithm

Algorand's consensus algorithm consists of a synchronous protocol that combines the concepts of PoS systems with a Byzantine fault tolerance agreement. The protocol virtually schedules the time into rounds. At each round, all the network nodes attempt to reach consensus on a new block of transactions. Each round is composed of the following actions:

- Each node in the network must first check its role to determine whether he has to propose a block for a given round. To do so, the nodes use cryptographic sortition, which requires them to run a trivial (a single hash) computation challenge. In case a node has to propose a block, he collects all the pending transactions inside a block proposal and gossips it together with the sortition proof. The block's size is limited by a fixed protocol parameter.
- Each node waits for incoming block proposals from the previous step for a predefined duration of time. Among all the valid blocks he receives, he selects the one with the highest priority sortition proof. Then, he computes an hash using this block, sets this hash as the input for the BA of the next steps.
- All nodes in the network try to reach consensus on one block through a BA protocol. The BA has two key phases. Both the phases are composed by subsequent steps. At each step, a distinct group of users (called committee members) is elected in an unpredictable way through cryptographic sortition. These committee members spread their voting messages based on votes received from the previous step. The committee members attach a sortition proof that proves their legitimacy while spreading their voting messages. The two phases of BA are as follows:
  (1) The first phase is called the *reduction* phase that comprises two steps. In the first step, each committee member votes for the hash of the blocks proposed for consideration. In the second step, committee members vote for the hash that received votes over a certain threshold (defined by protocol). In case none of the hash/block receives enough votes, committee members vote for the hash of the default empty block.
  (2) The next phase consists of another binary BA to ensure that each node agrees on the same consensus; the consensus can conclude on the output of the previous phase or on the default value in the absence of a majority.

2.6 Cryptographic sortition

Cryptographic sortition is used by each network node at the beginning of every step of each phase to determine whether he has a role in that particular step. It is a simple and lightweight process that involves computing a single hash and a digital signature. The sortition algorithm is implemented using a Verifiable Random Function (VRF[5]) [22]. In particular, the hash for a certain step s and round r is computed by the user i as

$$y = H(sig_i(r, s, Q^{r-1})), \tag{8}$$

where H is a hashing function, y a 256-bit long hash, and Q^{r-1} is the seed quantity defined by Eq. 5. $sig_i(r, s, Q^{r-1})$ is called the sortition proof and is attached to voting messages or block proposals to prove the legitimacy of the corresponding voters or block proposers. The hash is used by the nodes to verify a certain sortition condition. The possibility that the condition is verified is directly proportional to the stake of the node and depends on a constant parameter π_{role} fixed in the protocol, which guarantees that on average a fixed number of nodes are elected at each step for a certain role. For this reason, the protocol allows users to be elected more than once in the same step. A user i can possibly be elected at each step w_i times, where w_i is the number of user's units of currency. Formally, the interval $[0, 1)$ is partitioned into w_i intervals I, where the j-th interval I_j is defined as:

$$I_j = \Big[\sum_{x=0}^{j} B(x; w_i; p), \sum_{x=0}^{j+1} B(x; w_i; p) \Big), j \in (0, 1, \ldots w_i) \tag{9}$$

[5]VRF functions allow users to produce verifiable computations. The users produce a proof using their private keys, which can be verified by other users via the producer's public key.

where $B(x; w_i; p)$ is the binomial probability that user i is elected exactly x times on w_i chances, each with probability $p = \pi_{role}/W$. Here, π_{role} estimates the expected number of sorted users for that role and W is the total amount of currency in the network. The number of votes are found by applying the binary point operator to the hash (i.e., $y/2^{y.length}$) and searching the interval I_j with which the value is associated. If the value lies on the j-th interval, then i can vote j times. There are at least three interesting observations about this procedure:

(1) Since on average a fixed number of nodes are elected at each step, the traffic of voting messages in the network does not increase with the number of participants.
(2) The procedure to decide whether a user i is sorted for a certain step in a round can be executed only by the user i as it involves a signature process.
(3) Since Q^{r-1} can be computed by a node only when that node reaches consensus on round $r-1$, the result of the sortition of user i in a certain step of round r is unpredictable and unknown even to the user i. This avoids the possibility of collusion between voters of one or more steps to manipulate the consensus process on a certain round.

3 OUR ATTACK

We explain our attack in this section, starting from attack preliminaries in Section 3.1 followed by a brief introduction to flooding attacks in Section 3.2, and the description of our attack in Section 3.3.

3.1 Attack preliminaries

In our adversary model, we assume that all the malicious nodes are coordinated by a single/unique entity. This translates into the following two properties of the malicious nodes:

(1) Each malicious node can sign messages using any other malicious node's private key.
(2) The malicious nodes participate in the protocol either passively or actively. These malicious nodes coordinate and use the messages they receive from their honest peers to become aware of the protocol's execution status.

We assume that all the malicious nodes and their public/private key pairs were created by the adversary sufficiently in advance, i.e., before the attack takes place. This enables all of them to become the voters or block proposers in the protocol[6]. As mentioned in Section 2.4, we can also safely assume that the network is vulnerable to the Sybil attack, meaning that the honest nodes choose their peers randomly without relying on their stake. In this way, the malicious nodes have the same probability as of the other nodes to connect to honest peers. On the other side, honest peers can control/limit the number of incoming connections, denoted by *max-connections*.

3.2 A typical flooding attack

In a typical network scenario, a flooding attack targets honest nodes' bandwidth and memory resources by sending a huge number of message to him. Typically, a flooding attack sends a huge number of fake or invalid messages towards the target nodes to slow down their message reception and message processing. Crafting such attacks from a single nodes is not very effective because nodes in Bitcoin-like decentralized networks - where no level of trust is assumed between participants - often rely on "misbehaviour" scores to label and identify possible malicious users. As soon as an honest node finds that one of his peers is malicious, he can stop processing messages from that peer, drop the connection, and blacklist him to prevent future connection requests from that peer for a certain time. Hence, the attacker needs a large number of malicious nodes connected to the target. So, he can send some messages from each connection in a coordinated manner and create the same impact without being detected (or at least fully detected).

3.3 Magnifying attack's impact using undecidable messages

Our attack aims to increase the impact of the flooding attacks by exploiting the message validation process of Algorand. Due its design, our attack also enables an adversary to have a longer connection time before getting blacklisted by the target node. In Algorand, each message that is a part of the consensus algorithm (i.e., voting messages (Eq. 2) and block proposals (Eq. 3)) is subject to two distinct checks during the validation process:

(1) *Stateless checks*: A set of checks on a message/packet that a validator node can perform to know its current status (e.g., packet's structure, packet's authentication, check for duplicated messages).
(2) *Sortition proof check*: The sortition proof must be checked to verify that the sender/author of the message had the right to gossip it.

As described Section 2.6, the sortition proof is a byte-string that represents the signature of a user i in the form $sig_i(r, s, Q^{r-1})$, the validator node should hash this string to verify whether the given output satisfies the sortition property. However, before blindly hashing the signature, the protocol requires him to verify it first. Thus, he needs all the parameters - that were originally used to create it - including the seed quantity Q^{r-1}. This is a stateful check because the validator should have already reached consensus on the round $r-1$ to independently compute Q^{r-1}. If that is not the case, then the message cannot be fully validated at the current status of the validator node. In such a situation, the only option that the validator node has is to store the message and wait until it can be fully validated because messages that are not yet fully validated, cannot be discarded. We refer to these messages as *undecidable messages*.

At this point, the aim of the attacker is to flood honest targets with as many undecidable messages as possible to saturate their bandwidth and possibly their memory. These two objectives decide the type and number of messages used to construct our attack vector. With respect to the type, we chose the largest possible message, i.e., block proposals containing max-sized blocks. Since messages need to pass the stateless checks, each public key can gossip only a single block proposal per round. However, keys are created at zero cost and each malicious node can sign messages with any of the

[6]This assumption is necessary because Algorand increases the unpredictability of the outcome of sortition process by limiting the set of keys that can participate in the process. The eligible keys must be created at least k rounds before.

adversary keys. Hence, each node can include an arbitrary number of block proposals in his payload - one for each key[7].

Now, we describe the attack's execution in practice. The attacker connects to his target as one of his peers. Since the target node is assumed to honestly execute the Algorand protocol, as soon as he receives and validates a message for a certain round r, he has to relay it to all of his peers including the malicious node. Hence, the attacker just needs to (1) prepare its payload in advance[8] for the attack in round r; (2) wait to receive a block proposal or a credential message (Eq. 7) from his target[9] in round r to be aware of the fact that the target has started the execution of round r; and (3) send the payload to the target from each connected malicious node.

4 EVALUATION

In this section we present the evaluation of our proposed attack. Section 4.1 provides the technical details of our simulator while the evaluation settings and the results are presented in Section 4.2 and Section 4.3, respectively.

4.1 Simulator

Since, the source code or an official simulator for Algorand was not available at the time of our experiments, we built our own simulator that is completely written in Java. Both the works [9, 15] on Algorand propose slightly different, but overlapping versions of the protocol. The differences are in the terminating conditions to reach the final majority consensus within BA. However, our threat model is independent of the particulars of the chosen version of BA. We referred to the protocol described in the technical paper [15] for the implementation in our simulator.

We model an honest node as an entity composed of three main components: (1) a network interface to send/receive messages from peers; (2) a validator process to verify the received messages; and (3) the process that runs the protocol. Since our attack focuses on flooding block proposals or voting messages, transactions are never sent/received directly in our simulation. Instead, the throughput (transactions confirmed per second) is simulated by creating block proposals containing a fixed number of arbitrary transactions[10].

To reduce the impact of simultaneous/parallel signature verification during simulation, we simulate block validation by using a sleep procedure. Furthermore, nodes are provided with a cache memory not to verify the same transaction twice[11]. While block signatures are simulated other checks, e.g., packet authentication and sortition proof validation, are actually computed legitimately. We do not assume any latency in the network communication for the sake of presenting the minimum impact of our proposed attack, which would further enhance in the presence of latency.

4.2 Evaluation settings

All simulations were done on a virtual machine on OpenStack [5] running Ubuntu 16.04 with 16 Intel Core Processors (Hashwell, no TSX, IBRS) and 64 GB of RAM. In our experiments, we simulated a network of 500 nodes, where each node had a 30 Mbps upload and 30 Mbps download bandwidth. The target nodes were connected to all the malicious nodes involved in the attack. However, the number of malicious nodes does not represent the *max-connections* parameter controlled by the honest nodes as honest nodes were connected to malicious as well as other honest peers. For the sake of simplicity, we assumed that once an undecidable message fails in the verification process, then all messages from the sender of that message are discarded without being processed. Each simulation was run for ~45 minutes, and the attack payload was prepared using different blocks sizes during different experiments.

To evaluate the effectiveness of our attack, we focus on: (1) the number of "legitimate" messages received and validated in due time by the honest nodes and (2) the average time taken to complete a round. These two metrics are co-related in a way that if too many messages are not received/processed in due time for a given step, the execution time of the corresponding step increases until a timeout is reached. We thoroughly evaluated our attack scenario by varying: (1) the number of malicious nodes involved in the attack; (2) the number of block proposals sent from each malicious node, i.e., keys per malicious node; and (3) the size (in MB) of the attack payload.

To show the impact of our attack, we considered realistic settings where only a very low number of nodes (1% to 3%) are malicious in the network. The maximum number of block proposals sent from one malicious node at the time of attack was 70, following the analysis of the cryptographic sortition for block proposers by the authors of the protocol [15]. We set the block proposal receiving time and the step timeout (explained in the next section) to 150 seconds and 60 seconds, respectively.

4.3 Results

The execution time for a round consists of two parts: (1) block proposal receiving time and (2) step timeout. The block proposal receiving time is constant and is defined as the maximum window of time for the nodes to receive the block proposals for a given round. The step timeout is the maximum amount of time by which each step must complete. Here, each step finishes as soon as enough voting messages are processed or the step timeout occurs. In "no attack" scenario, the largest contribution to round time should come from the block proposal receiving time and the steps should execute before the step timeout occurs. The same is reflected in our

[7] As long as the messages are authenticated and sent only once, a single public key is able to create a different undecidable message for every step in each round. It means that it can create a block proposal on the first step and a voting message for every step of BA (until the maximum m-th step allowed by the protocol). However, sending one message for every step implies that the given public key is sorted at every step of that round, which is very unlikely. A similar argument can be used by the target node to statistically label malicious users without waiting for their messages to be fully verifiable. Moreover, given a reasonable max-block size (order of MB) and m set to 150 steps [15], a single block proposal containing a max-size block would make up the large majority of the total payload weight. Hence, choosing a single block proposal for each key would imply that each user is elected only once in a given round, and cheaters are more difficult to identify a-priori.

[8] The attacker does not need any information to pre-compute a payload for a certain round r. He just needs to sign some random messages declaring them as if they belong to round $r + 1$. All additional information will be checked only once the message becomes fully verifiable in round $r + 1$, by that time the attacker would have already achieved his goal.

[9] Block proposals together with the credential messages are gossiped first through the network at each round.

[10] We assume that in a real implementation of the protocol, messages and transactions are processed by independent processes.

[11] Without a dedicated cache, transactions are validated at least twice. The time when the transaction is gossiped by its original creator and the time when it is received in a block.

results shown in Figure 1. Here, in the absence of malicious nodes, nearly 83% of the round time was taken by the fixed block proposal receiving window. On the other side, the round time remains nearly the same as "no attack" scenario with a lower number of malicious nodes/keys per malicious node. However, the protocol's performance starts to deteriorate with increasing number of malicious nodes/keys per malicious node, starting from the attack payload of just 500 blocks (10 malicious nodes with 50 malicious keys). In facts, the average round time with all the remaining higher attack configurations was over 390 seconds; which means that these configurations caused at least four step timeouts in addition to the block proposal receiving time. As mentioned earlier, our proposed attack does not prevent nodes from reaching consensus. Instead, it aims to significantly increase the execution time of each round, which eventually leads the attacked nodes to reach their consensus on the default value, leaving the targeted nodes behind the non-attacked nodes in the chain.

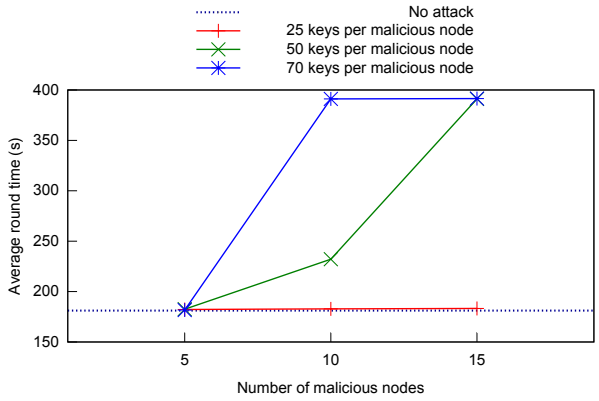


Figure 1: Effect of the number of malicious nodes and keys per malicious nodes upon average round time

The next important criteria to evaluate the impact of an attack on Algorand is the percentage of "legitimate" messages received and "legitimate" messages validated in due time. Figure 2 shows the effect of our attack upon "legitimate" messages received and validated with respect to the number of malicious nodes and keys per malicious nodes. Also here, with an attack payload of just 500 blocks (10 malicious nodes with 50 malicious keys), the percentage of "legitimate" messages received/validated starts to degrade considerably, which eventually lead to step failures.

It is important to mention that an adversary can tune our attack on the number of malicious nodes/keys to prevent the detection while still creating significant disturbance in the network. Moreover, the size of the blocks is another important parameter to tune the attack. The results presented in the Figures 1 and 2 were obtained using 1 MB of attack payload. Figures 3 and 4 show the impact of attack payload's size upon the average round time and the percentage of "legitimate" messages received as well as validated in due time, respectively. Our results show that the size of the block does not significantly affect the performance of honest nodes in the absence of the attack. However, increasing size of the attack payload severely affect the performance of honest nodes in term of

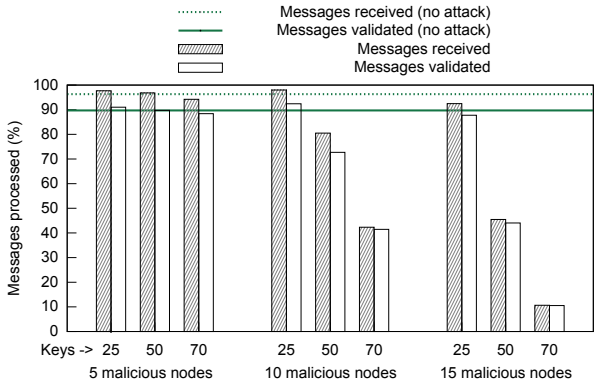


Figure 2: Effect of the number of malicious nodes and keys per malicious nodes upon percentage of "legitimate" messages received and validated

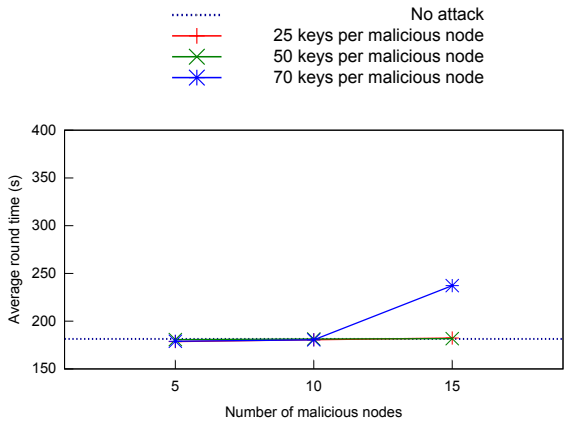
both the average round time and messages processed. It happens because the block's size directly affects the download time of a block, which is further worsened by the number of malicious nodes and the keys per malicious nodes involved in the attack.

5 FEASIBILITY OF THE ATTACK

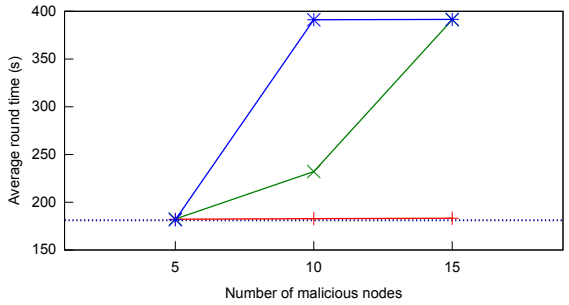
The attack costs practically nothing to an adversary. However, the adversary has to overcome one minor challenge, i.e., to connect at least one malicious node with the honest target. This malicious node can then flood the target with an arbitrarily large number of block proposals - one for each public key controlled by the adversary. However, the recipient can monitor and label connections from which he receives a suspiciously large number of messages for a round. Then, it is likely that the honest nodes would simply drop (or, at least temporarily stop listening) from such a connection. Hence, the success of our attack depends on how many malicious connections an attacker is able to establish with the target node.

The other important aspects of our attack are: (1) since TCP connections are required to be established, IP spoofing is not an option[12] and real addresses should be used; (2) the target's *max-connections* parameter defines the maximum number of connections allowed towards it. We believe that the former issue can be solved by using botnet or relying on hidden services, such as Tor, to protect the address identity inside the network. However, the *max-connections* parameter is out of the adversary's control. Hence, it is considered as an attack variable in our attack scenario. Consequently, establishing and maintaining a high number of connections with the target nodes is somehow a bit more challenging. But, the goal is achievable due to the current design of the protocol, where peers are not weighted on their stake.

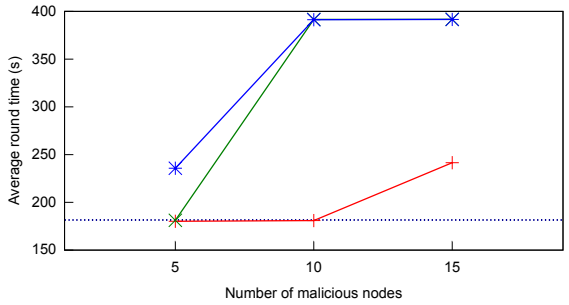
[12]Establishing TCP connections via IP spoofing is problematic as one cannot complete the TCP three-way handshake protocol. In particular, SYN-ACK packets from the target nodes are not sent to the real location of the malicious node. To make this happen, the adversary must have control over at least one router on the path between the fake address and the target, or the target node's network has allowed source-routing [33] of the messages, or the Initial Sequence Number (ISN) in the handshake protocol is vulnerable to the prediction attack. All of these speculations are quite strong assumptions.



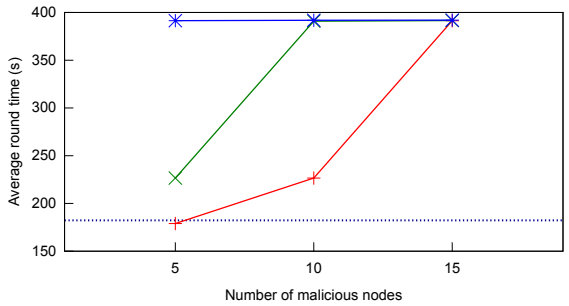
(a) Block size = 0.5 MB



(b) Block size = 1.0 MB

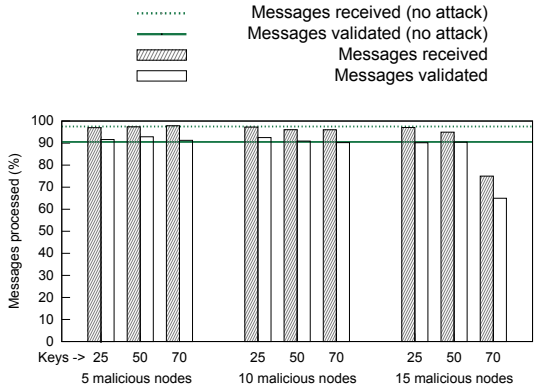


(c) Block size = 1.5 MB

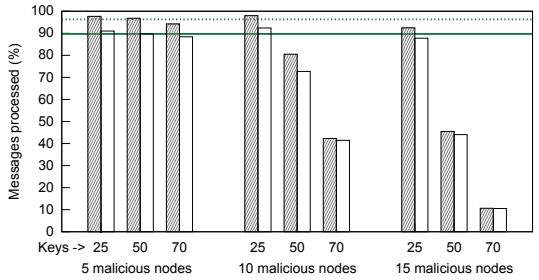


(d) Block size = 2.0 MB

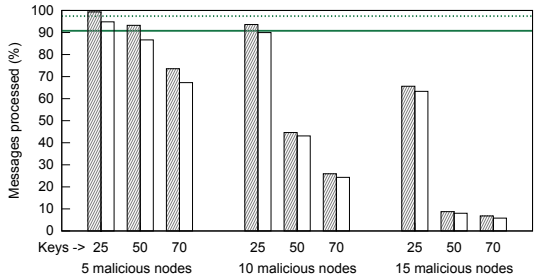
Figure 3: Effect of the attack payload's size upon average round time



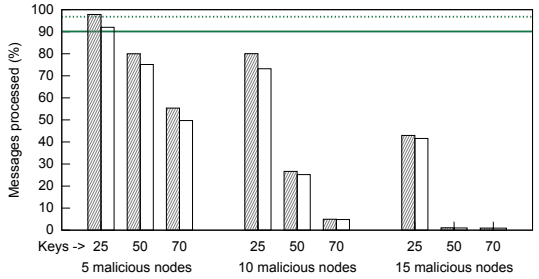
(a) Block size = 0.5 MB



(b) Block size = 1.0 MB



(c) Block size = 1.5 MB



(d) Block size = 2.0 MB

Figure 4: Effect of the attack payload's size upon the percentage of "legitimate" messages received and validated

6 CONCLUSION AND FUTURE WORKS

The Algorand protocol was proposed to overcome the limitations of conventional blockchain technologies. The protocol has great potential and is in under active development. In this paper, we present a particular security flaw of Algorand protocol that exploits its process of validating messages. In particular, we evaluated a possible DDoS-like flooding scenario under practical assumptions, where honest nodes suffer significant delays in the execution of protocol. Furthermore, we also discuss the major factors that make this attack scenario more challenging for the honest nodes. Finally, we also present possible solutions to prevent such an attack. The rigorous formalization of these countermeasures is kept as our future work.

ACKNOWLEDGMENTS

Ankit Gangwal is pursuing his Ph.D. with a fellowship for international students funded by Fondazione Cassa di Risparmio di Padova e Rovigo (CARIPARO). This work was supported in part by EU LOCARD Project under Grant H2020-SU-SEC-2018-832735, and in part by Huawei Project "Secure Remote OTA Updates for In-Vehicle Software Systems" under Grant HIRPO 2018040400359-2018.

REFERENCES

[1] 2014. Enigma. A Private, Secure and Untraceable Transactions System for Cloackcoin. [online] https://www.cloakcoin.com/user/themes/g5_cloak/resources/CloakCoin_Whitepaper_v2.1.pdf.
[2] 2014. Whitepaper: Nxt. [online] https://nxtwiki.org/wiki/Whitepaper:Nxt.
[3] 2016. Hyperledger Sawtooth documentation. [online] https://intelledger.github.io/.
[4] 2018. Banking Is Only The Beginning: 42 Big Industries Blockchain Could Transform. [online] https://www.cbinsights.com/research/industries-disrupted-blockchain/.
[5] 2018. OpenStack. [online] www.openstack.org.
[6] Iddo Bentov, Rafael Pass, and Elaine Shi. 2016. Snow White: Provably Secure Proofs-of-Stake. *IACR Cryptology ePrint Archive* 2016 (2016), 919.
[7] Vitalik Buterin. 2015. Slasher: A Punitive Proof-of-Stake Algorithm. [online] https://ethereum.github.io/blog/2014/01/15/slasher-a-punitive-proof-of-stake-algorithm/.
[8] Vitalik Buterin and Virgil Griffith. 2017. Casper the Friendly Finality Gadget. *arXiv preprint arXiv:1710.09437* (2017).
[9] Jing Chen and Silvio Micali. 2016. Algorand. *arXiv preprint arXiv:1607.01341* (2016).
[10] Mauro Conti, Ankit Gangwal, and Sushmita Ruj. 2018. On the Economic Significance of Ransomware Campaigns: A Bitcoin Transactions Perspective. *Elsevier Computers & Security* 79 (2018), 162–189.
[11] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, et al. 2016. On Scaling Decentralized Blockchains. In *20th International Conference on Financial Cryptography and Data Security*. 106–125.
[12] Wei Dai. 1998. b-money. [online] http://www.weidai.com/bmoney.txt.
[13] Ittay Eyal. 2015. The Miner's Dilemma. In *36th IEEE Symposium on Security and Privacy (SP)*. 89–103.
[14] Hal Finney. 2004. RPOW - Reusable Proof Of Work. [online] http://cryptome.org/rpow.htm.
[15] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. 2017. Algorand: Scaling Byzantine Agreements for Cryptocurrencies. In *26th ACM Symposium on Operating Systems Principles*. 51–68.
[16] Stuart Haber and W Scott Stornetta. 1990. How to Time-stamp a Digital Document. In *Conference on the Theory and Application of Cryptography (CRYPTO)*. 437–455.
[17] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. 2017. Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol. In *Annual International Cryptology Conference (CRYPTO)*. Springer, 357–388.
[18] Sunny King. 2013. Primecoin: Cryptocurrency with Prime Number Proof-of-Work. [online] http://primecoin.io/bin/primecoin-paper.pdf.
[19] Sunny King and Scott Nadal. 2012. PPcoin: Peer-to-Peer Crypto-currency with Proof-of-Stake. [online] https://pdfs.semanticscholar.org/0db3/8d32069f3341d34c35085dc009a85ba13c13.pdf.
[20] Wenting Li, Sébastien Andreina, Jens-Matthias Bohli, and Ghassan Karame. 2017. Securing Proof-of-Stake Blockchain Protocols. In *Data Privacy Management - Cryptocurrencies and Blockchain Technology (DPM - CBT 2017)*. 297–315.
[21] Loi Luu, Ratul Saha, Inian Parameshwaran, Prateek Saxena, and Aquinas Hobor. 2015. On Power Splitting Games in Distributed Computation: The Case of Bitcoin Pooled Mining. In *28th IEEE Computer Security Foundations Symposium (CSF)*. 397–411.
[22] Silvio Micali, Michael Rabin, and Salil Vadhan. 1999. Verifiable Random Functions. In *40th IEEE Annual Symposium on Foundations of Computer Science*. 120–130.
[23] Satoshi Nakamoto. 2008. Bitcoin: A Peer-to-Peer Electronic Cash System. [online] https://bitcoin.org/bitcoin.pdf.
[24] Karl J O'Dwyer and David Malone. 2014. Bitcoin Mining and its Energy Footprint. (2014).
[25] P4Titan. 2014. Slimcoin. A Peer-to-Peer Crypto-Currency with Proof-of-Burn. [online] https://github.com/slimcoin-project/slimcoin-project.github.io/raw/master/whitepaperSLM.pdf.
[26] Colin Percival. 2009. Stronger Key Derivation via Sequential Memory-hard Functions. [online] https://www.tarsnap.com/scrypt/scrypt.pdf.
[27] Douglas Pike, Patrick Nosker, David Boehm, Daniel Grisham, Steve Woods, and Joshua Marston. 2015. PoST White Paper. [online] https://www.vericoin.info/downloads/VeriCoinPoSTWhitePaper10May2015.pdf.
[28] Serguei Popov. 2016. IOTA: The tangle.
[29] Larry Ren. 2014. Proof of Stake Velocity: Building the Social Currency of the Digital Age. [online] https://www.reddcoin.com/papers/PoSV.pdf.
[30] Meni Rosenfeld. 2011. Analysis of Bitcoin Pooled Mining Reward Systems. *arXiv preprint arXiv:1112.4980* (2011).
[31] Yonatan Sompolinsky, Yoad Lewenberg, and Aviv Zohar. 2016. SPECTRE: A Fast and Scalable Cryptocurrency Protocol. *IACR Cryptology ePrint Archive* (2016), 1159.
[32] Iain Stewart. 2018. Proof of Burn. [online] https://en.bitcoin.it/wiki/Proof_of_burn.
[33] Carl A Sunshine. 1977. Source Routing in Computer Networks. *ACM SIGCOMM Computer Communication Review* 7, 1 (1977), 29–33.
[34] Nick Szabo. 2005. Bit Gold. [online] https://unenumerated.blogspot.com/2005/12/bit-gold.html.
[35] Pavel Vasin. 2014. Blackcoin's Proof-of-Stake Protocol V2. [online] https://blackcoin.org/blackcoin-pos-protocol-v2-whitepaper.pdf.