3-2019

A New Product Anti-Counterfeiting Blockchain Using a Truly Decentralized Dynamic Consensus Protocol

Naif Alzahrani
Portland State University, nalza2@pdx.edu

Nirupama Bulusu
Portland State University, nbulusu@pdx.edu

Citation Details
Alzahrani, N, Bulusu, N. [Post-print version] A new product anti-counterfeiting blockchain using a truly decentralized dynamic consensus protocol. Concurrency Computat Pract Exper. 2020; 32:e5232. https://doi.org/10.1002/cpe.5232

A New Product Anti-Counterfeiting Blockchain Using a Truly Decentralized, Dynamic Consensus Protocol

Naif Alzahrani, nalza2@pdx.edu
Nirupama Bulusu, nbulusu@pdx.edu

Abstract

The growth of counterfeit goods has plagued the international community for decades. Nowadays, the battle against counterfeiting remains a significant challenge. Most of the current anti-counterfeiting systems are centralized. Motivated by the evolution of blockchain technology, we propose (Block-Supply), a decentralized anti-counterfeiting supply chain that exploits NFC and blockchain technologies.

This paper, also, proposes a new truly decentralized consensus protocol that, unlike most of the existing protocols, does not require PoW and randomly employs a different set of different size of validators each time a new block is proposed. Our protocol utilizes a game theoretical model to analyze the risk likelihood of the block's proposing nodes. This risk likelihood is used to determine the number of validators involved in the consensus process. Additionally, the game model enforces the honest consensus nodes' behavior by rewarding honest players and penalizing dishonest ones.

Our protocol utilizes a novel, decentralized, dynamic mapping between the nodes that participate in the consensus process. This mapping ensures that the interaction between these nodes is executed anonymously and blindly. This way of mapping withstands many attacks that require knowing the identities of the participating nodes in advance, such as DDoS, Bribery, and Eclipse attacks.

1 Introduction

1.1 Motivation

Explosive growth in pharmaceuticals and other products across the world has led to growth in large, globalized, digital supply chains [1]. The structure of modern supply chains is mostly centralized, in which central authority stores and manages products' authentication records. In this typical structure, each node authenticates a product upon its arrival. This authentication is carried out by executing authentication protocols between a supply chain node and an

authentication server. Track-and-trace based supply chains function similarly, where a centralized tracking server tracks the physical locations of products and updates their records.

Recently, blockchain technology has emerged as a proper solution for many different industries beyond cryptocurrency [2]. Blockchain offers better transparency, security, global peer-to-peer interaction, and decentralization [3]. Primarily, a blockchain is a distributed public ledger, which contains chained blocks. Each block consists of several transactions. The blocks are validated globally and transparently by some selected nodes in the network to assure security [1].

Despite the potential of blockchain technology to better establish supply chain provenance, there have only been a few projects that examine leveraging blockchain technology in anti-counterfeiting supply chains. In a recent review published in 2017 [1], Mackey and Nayyar state that "we were only able to extract a single 2016 IEEE non-research article that summarized a few blockchain projects initiated by different organizations and explored it as a potential solution for fake medicines among other healthcare problems." This has motivated us to utilize this promising technology to battle counterfeit goods by transforming our previous centralized supply chain [4] to a decentralized Block-Supply chain.

Blockchain was introduced to solve the problem of reaching agreements on a state among distributed nodes without a coordinating third party. These nodes are trustless and may contain faulty nodes (e.g., malicious, crashed). Reaching a consensus on a proposed block in the presence of faulty nodes requires a consensus protocol executed by some selected nodes called validators or miners. In response to this, many research has been conducted to come up with robust yet efficient consensus protocols that guarantee consensus in the blockchain.

Consensus protocols ensure that all nodes in the blockchain network agree on the validity of a block to be included in the public ledger. It also guarantees that all nodes have the same order of blocks in their blockchains. This is of significance because blockchains are trustless distributed nodes which need a way to synchronize their copies of stored data. Hence, a consensus protocol is designed to accomplish the reliability of a network that has multiple unreliable nodes [5].

Over the last few years, a considerable number of consensus protocols have been proposed. Nonetheless, not all of them guarantee *true decentralization*, in which the blocks' validation is executed by anonymous, variable sets of validators to strengthen the protocol's robustness. Instead, they rely on fixed, known validators selected at the genesis state. This opens the door for various risk threats, which will be discussed shortly. Additionally, most of the current consensus protocols do not take the number of validators or how to select them into consideration. Motivated by this argument, we aim in this paper to introduce a new consensus protocol which enjoys the true decentralization features and offers high efficiency.

1.2 Problems

1.2.1 Centralization of Current Anti-Counterfeiting Supply Chains

Current anti-counterfeiting supply chains aim to mitigate counterfeiting attacks. Lehtonen et al. [6] define three kinds of product counterfeiting attacks. First, the modification attack, in which an adversary modifies a product's details on a tag, such as changing the expiration date. Second, the cloning attack, where an adversary clones a genuine product's details to use on counterfeit products. The third attack is the tag reapplication attack, in which the legitimate tag is removed from a genuine product and reapplied to a counterfeit product. Several approaches have been proposed; however, none currently detect all of these attacks. In other words, the **tag reapplication attack** remains a threat in many product authentication systems [6], and requires a manual complex verification to detect it [7]. In addition, most of the existing anti-counterfeiting supply chains are **centralized and depend on a central authority** to authenticate products. Despite their potential to detect counterfeit products, their centralization dependency introduces many problems. First, there is an enormous processing burden on the server, since significant numbers of products flood through the supply chain nodes. Second, substantial storage is required to store authentication records for all products. Third, as with centralized systems, traditional supply chains inherently have the problem of a single point of failure. Additionally, they do not offer transparency as they do not allow supply chain nodes to verify the authenticity of a product's data.

1.2.2 Non-True Decentralization of Current Consensus Protocols

One of the fundamental characteristics of blockchain technology is the consensus protocol. The nodes responsible for validating the blocks and executing consensus protocols are the validators (or miners in some blockchains). There exist a considerable number of existing consensus protocols. Nevertheless, not all of them guarantee **true decentralization**, and they are PoW (Proof of Work) based, or fixed-validators based. PoW requires massive computational effort, which results in high energy and computing resources consumption. Alternatively, fixed-validators' protocols rely on fixed, static validators responsible for validating all newly proposed blocks, which opens the door for adversaries to launch several attacks such as DDoS, bribery, and eclipse attacks on these validators. In contrast, a truly decentralized protocol ensures that the blocks' validations are executed by anonymous, variable sets of validators, which results in greater security.

Another issue with the current consensus protocols is that most of them do not take **the number of validators or how to select them** into consideration. The number of validators in a blockchain influences its security substantially. The optimal number of validators, which achieves the optimal security, would be all nodes in the network except the proposing node (i.e., the node which creates and proposes the new block). However, this choice results in: (1) substantial validating work as of $O(n)$ for each validation event, (2)

high communication overhead to reach consensus with $O(n^2)$ in protocols like PBFT [8] (Practical Byzantine Fault Tolerance) and Tendermint [5], and (3) large block size due to including each validator's signature as an evidence of its validity.

An alternative choice to the $n-1$ validators is to rely on a fixed static number of validators chosen at the genesis state (e.g., Tendermint and Hyperledger Fabric [9]). Although the fixed-validators approach is efficient, it has several limitations. First, it relies on an extreme trust assumption that the majority of validators are honest; nevertheless, it is possible for a powerful adversary to corrupt or bribe most of them over time [10]. Second, a fixed committee of validators is vulnerable to adversarial attacks, since they are known and fixed. For example, an adversary can launch a DoS attack against the validators, preventing them from validating new blocks or receiving messages from each other. Third, the validators are selected based on their voting power. In a network where the nodes have equal voting power, this approach does not guarantee the fairness of selection. Fourth, although this approach is efficient, utilizing a relatively small number of validators in a large network with a massive amount of transactions or blocks can bottleneck the performance.

1.3 Contributions

The contribution of this paper is of threefold.

1.3.1 Block-Supply Chain

To overcome the problem discussed in Section 1.2.1 (i.e., current centralization anti-counterfeiting supply chains), we propose the *Block-Supply chain*. Block-Supply is a decentralized supply chain that exploits blockchain technology. In this chain, each node maintains a blockchain for each product. This blockchain is comprised of chained blocks where each is an authentication event. A new block is proposed to the network by the node that currently has the product (i.e., the proposing node). This newly proposed block is then validated by a number of other nodes we call validators, to ensure that the new block is valid. Upon successful validation, all nodes in the Block-Supply chain network add this block to their copies of the blockchain.

Block-Supply chain eliminates the need for a centralized authentication server utilized in most existing supply chains. Instead, it involves the nodes in the supply chain to do the authentication. Additionally, it can trace-and-track products without a centralized tracking server. Moreover, it detects the three counterfeiting attacks (modification, cloning, and tag reapplication) by involving the supply chain nodes transparently.

1.3.2 True Decentralized Consensus Protocol

To overcome the problem discussed in Section 1.2.1 (i.e., non-true decentralized consensus protocols), we propose a *truly decentralized consensus protocol* that

does not require PoW and which randomly employs a different set of validators on each block's proposal.

In our proposed protocol, at the genesis state, the blockchain initiator randomly maps each *proposing node* to four *validation-leader node*s. The validation-leaders are responsible for randomly selecting the *validator* nodes for this particular proposer. The selected validators are responsible for validating the block proposed by the proposer. Our protocol achieves true decentralization by anonymous mapping between the proposers and their leaders. In other words, no proposer knows its leaders, nor does the leader know its proposer or its other peer leaders. They, however, communicate with each other via private secrets assigned to them at the geniuses state. This way, their identities remain anonymous, preventing powerful attackers from launching attacks such as DDoS and eclipse, since the attackers do not know these nodes in advance. Each leader randomly selects a portion of the validators without any communication with the other leaders. The set of selected validators are different for each proposed block due to exploiting the randomness. More on this in Section 4.4.

Our proposed protocol is based on Tendermint, but it deals with the **the number of selected validators** issue by randomly selecting a relatively small number of validators proportional to the total number of nodes in the network. This way, we avoid employing all nodes in the system as validators. As mentioned in Section 1.2.2, using all nodes as validators yields optimal security, since every node participates in the validating of new blocks. Nevertheless, this choice degrades the consensus efficiency (performance) regarding the time taken to reach a consensus on a proposed block. This is due to the validation work performed by every node and the communication overhead between validators to reach the consensus. Tendermint is a fixed-validators decentralized protocol, which as mentioned earlier may suffer from attacks such as DDoS, eclipse, and validators bribing (these attacks will be discussed in detail in Section 6.3). To overcome the issue of **the number of selected validators and how to select them**, we integrate a *game theoretical model* to our protocol which can determine the risk likelihood ($risk$) of each proposing node dynamically. The number of selected validators is proportional to this $risk$. Hence, in a less hostile network environment, fewer validators are employed.

1.3.3 Dynamic Anonymous Proposers-Leaders Mapping

In this paper, we propose a novel dynamic anonymous proposers-leaders mapping mechanism. We mentioned in the previous section that the blockchain initiator anonymously maps each proposer to four leaders (i.e., static mapping). However, this mapping is valid only for *one* proposing round. That is when a proposer needs to repropose again, the identities of its leaders are revealed and, hence, they could be exposed to the previously mentioned attacks. To overcome such an issue, we introduce a new dynamic mapping approach that does not rely on the blockchain initiator and guarantees the anonymous mapping.

In our dynamic mapping approach, a new mapping is created every time a new block is proposed. This new mapping is utilized once in one of the next

round's proposing slot (i.e., one slot represents one new proposed block). Our approach uses secret nodes' identities to anonymize the mapping.

2 Related Work

The related literature is of threefold: (1) existing anti-counterfeiting approaches, (2) works that integrate blockchain and supply chain technologies, (3) the current widely used consensus protocols.

2.1 Anti-Counterfeiting Approaches

The current anti-counterfeiting works falls into two general camps: (1) works that track-and-trace products to mitigate counterfeiting, and (2) works that utilize cryptography to detect counterfeiting attacks.

2.1.1 Track-and-Trace Approaches

This kind of approach uses Radio-Frequency Identification (RFID) tags to track the physical locations of a product, which are then stored in a centralized database. Koh et al. [11] proposed one of the first track-and-trace approaches, which uses Electronic Product Codes (EPC) to uniquely identify and track products in pharmaceutical supply chains. This approach can detect cloning attacks because the EPC of a counterfeit product will appear at least twice in the database. For example, a product with an EPC registered in Switzerland for sale cannot be registered in America at the same time. Nevertheless, the problem with Koh et al.'s approach is that it cannot detect tag reapplication attacks.

In 2007, the EPCglobal and US Federal Drug Administration (FDA) designated the drug e-Pedigree to be used in securing pharmaceutical supply chains in the USA [12]. Each product is linked to an RFID tag that has a unique EPC. e-Pedigree is an electronic certified record that contains the product information (e.g., serial number, name, expiration date), manufacturer information, transaction information (e.g., transaction ID, time, location), distributor information, recipient information, and signatures. Products' e-Pedigrees are stored in a central database accessible to the supply chain nodes. In this track-and-trace approach, the product's manufacturer performs the following:

- Creates an e-Pedigree for the product, which has the product and manufacturer information.
- Digitally signs the e-Pedigree using the manufacturer's private key.
- Stores the e-Pedigree along with its digital signature in the central database.

As the product moves through the supply chain, each node performs the following:

- Uses the EPC of the product's tag to query the central database about the product's e-Pedigree.
- Verifies the signature of the e-Pedigree.
- If valid, updates the e-Pedigree by adding the node information (distributor information), the transaction information, and the recipient information.
- Digitally signs the updated e-Pedigree using the node's private key.
- Stores the updated e-Pedigree along with its digital signature in the central database.

The benefit of e-Pedigree is to quickly track products through the supply chain and make counterfeiting more difficult [13]. This approach can detect cloning attacks by tracking and tracing the product with a complete product e-Pedigree as the product moves from the manufacturer to retailers [14]. In addition, it can detect modification attacks by using digital signatures. However, tag reapplication attacks remain a challenge for e-Pedigree-based approaches. This is because if a counterfeiter removes a tag from a genuine product and reapplies it to a counterfeit one, the e-Pedigree will remain valid, and the system therefore will not detect this attack.

2.1.2 Cryptographic Approaches

Cryptographic approaches use public or private key cryptography to authenticate products. Saeed et al. [15, 16] propose an offline approach that uses NFC tags and is based on public key cryptography. This approach allows customers to check the authenticity of products using their cell phones and does not require access to a database. Their approach assigns each instance of a product a unique public/private key pair, and uses a challenge-response protocol between the customer's phone and the tag on the product. The tag contains the private key in a secure location that is accessible only to the tag's processor. The corresponding public key is stored on the tag too, but can be obtained by the customer's phone. The main benefit of this approach is that it involves customers in product authentication. However, this approach does not detect tag reapplication attacks, and requires expensive NFC tags that have processors, secure storage, and support encryption.

Recently, TagPrint [17], an offline cryptographic approach, was proposed by Yang et al. to detect counterfeit products using RIFD. According to the authors, TagPrint is the first RFID-based offline approach in existing anti-counterfeiting systems. This approach involves three parties: a tag

provider, a product manufacturer, and a customer. First, the tags provider fingerprints its RFID tags by extracting some physical layer information to identify each tag. The tags and their fingerprints are offered to the product manufacturer. Second, the manufacturer attaches a group of tags (at least four) to each product in randomized geometric locations. The manufacturer encrypts the tags' fingerprints and geometric relationships and stores them in the tags' memories. Third, the customer employs an RFID tag's reader, which contains the manufacturer's public key. The reader reads and decrypts the encrypted fingerprints and geometric relationships from the tags' memories. After that, the reader obtains new tags' fingerprints and geometric relationships and compares them to the decrypted ones to check if they are the same and hence determine the authenticity of the product. TagPrint can detect modification and cloning attacks using passive low-cost RFID tags and can be executed offline. However, as acknowledged by the authors, TagPrint cannot detect tag reapplication attacks. In addition, TagPrint is based on RFID tags, that require specialized readers, making this approach unsuitable for ordinary consumers.

2.2 Block and Supply Chains

In this section, we only present the works that have integrated blockchain technology into supply chains. Although blockchain technology has gained considerable attention from the computer science community, the use of this technology in products' supply chains is limited [1].

Tian [18] proposed a conceptual framework of an agri-food supply chain, which uses RFID tags and blockchain technology. The proposed supply chain is designed to trace agri-food "from farm to fork." The author analyzed the advantages of using blockchains and stated that is it can be used to fight fake products.

Saveen et al. [19] discussed the potential benefits of using blockchain technology in manufacturing supply chains. They presented a structure for a manufacturing supply chain for cardboard boxes. This structure involves blockchain as a platform to collect, store and manage product details of each product throughout its life cycle. The work showed a general overview of replacing the centralized system with a decentralized one using blockchain.

Korpela et al. [20] conducted a study on integrating blockchain in digital supply chains. The use of blockchain would provide better access to customers by sharing information about products effectively. Besides, products and service deliveries can be tracked to ensure visibility in the supply chain. Their analysis showed that many of the digital supply chains' functionalities could be embedded in blockchain technology. In addition to that, blockchain provides security and cost-effective transactions in supply chains networks with no central system.

2.3 Consensus Protocols

In this section, we will examine related existing consensus protocols. The related literature falls into four general camps: (1) Proof of Work (PoW) based protocols, (2) Proof of Stake (PoS) protocols, (3) Byzantine Fault Tolerance (BFT) protocols, and non-Byzantine Fault Tolerance protocols.

2.3.1 Proof of Work (PoW)

In 2009, Satoshi Nakamoto introduced Bitcoin [21]; the first known implemented blockchain. Bitcoin utilizes the PoW consensus mechanism to reach an agreement on a proposed block. In PoW, transactions are grouped into a block, which then validated and confirmed by 'miners.' The miners are required to solve a challenge by computing cryptographic hashes. They achieve this by making trial and error computations until a consensus is reached. The way PoW works is that all miners compete to find a 'nonse,' so when combined with the proposed block, the block will hash to a target value determined by the mining difficulty. The successful miners are then rewarded due to their consumed computational power.

Blockchains that involve and rely on PoW mining to ensure consensus, such as Bitcoin, impose the following drawbacks:

1. Time-consuming: confirmation of transactions is slow. For example, it can take an average of 10 minutes to commit a block in Bitcoin.
2. High consumption of resources: due to the significant computation to solve a challenge which requires computing cryptographic hashes.
3. High energy consumption: which results in massive expenses.
4. Security: can be difficult to quantify [22].
5. Specialized hardware: sometimes required to increase the mining power.

2.3.2 Proof of Stake (PoS)

The most common alternative to PoW is Proof of Stake (PoS). In 2012, King and Nadal [23] introduced PoS to solve the problem of Bitcoin mining's high energy and computation consumption. In PoS, mining new blocks depends on who holds the highest amounts of cryptocurrency, in which a deterministic algorithm selects nodes according to the number of coins each one has. Hence, instead of investing in expensive computational power to mine blocks, miners invest in the currencies of the system. As a result, a miner's likelihood of being chosen to create a block depends on the fraction of coins the miner owns in the system. For example, a miner with 400 coins is four times more likely to be picked as another miner with 100 coins.

PoS protocols require less energy consumption than PoW and mitigate the hardware centralization risk [24]. However, some argued that PoS is not ideal for

blockchains [25]. This is because of a problem called "nothing at stake," where miners have nothing to lose by voting for multiple blocks and claim various sets of transaction fees, since a participant with nothing to lose has no reason not to misbehave. This problem prevents the consensus from resolving.

Delegated Proof of Stake (DPoS)

Permissionless/public blockchains that utilize the tradtional PoS often face scalability issues. As a result, Delegated Proof of Stake (DPoS) [26], a variation of the PoS, was adapted by some blockchains such as Lisk [27], EOS [28], BitShares [29], and Ark [30] seeking to reach consensus more efficiently.

In DPoS, nodes *vote* to select *witnesses*. A witness is a node that has been selected (i.e., voted on) to validate transactions. Each node votes for the witnesses, whom it trusts. The top tier of witnesses (i.e., the nodes collected most of the votes) win the privilege to validate. Moreover, in DPoS, nodes are allowed to *delegate* their voting power to other nodes that they trust to vote for witnesses on their behalf. The votes for a witness are weighed based on the size of every voter's stake. As a result, a node does not need to have a significant stake to be in the top tier of witnesses. Instead, votes from nodes with large stakes can elevate a node, with a small stake, to be a member of the witnesses in the top tier. The witnesses in the top tier are responsible for validating transactions and creating blocks for these transactions, and as a result, are awarded the associated fees.

2.3.3 Byzantine Fault Tolerance (BFT)

In this section, we present the most widely used consensus protocols that can tolerate Byzantine faults. These protocols are known to reach consensus and maintain liveness even in the presence of Byzantine faults (nodes' crashes, or malicious nods). Our proposed protocol falls into this category.

PBFT (Practical Byzantine Fault Tolerance)

PBFT [8] is a replication algorithm that can tolerate Byzantine faults. PBFT was first introduced in 1999, after many Byzantine Fault Tolerance (BFT) protocols were proposed to improve its robustness and performance [5]. PBFT works in asynchronous environments that might contain byzantine faults such as the Internet [8].

PBFT progresses through a chain of views. Each view has a primary (i.e., proposer,) which is selected in round-robin order. The other nodes (replicas) in the view are called backups. A client sends a request to the primary. The primary assigns the request a sequence number and multicasts a signed *pre-prepare* messages to the other backups. The *pre-prepare* contains the view and sequence numbers. If the backups did not already accept a *pre-prepare* message for the same view and sequence numbers, they accept the *pre-prepare*. After accepting the *pre-prepare*, a backup broadcasts a signed *prepare* message. For a replica to be prepared for a given request, the replica must receive $2f$ *prepare*

messages for that request (where f is the is the maximum number of replicas that may be faulty), with the same view and sequence number. When a replica is prepared, it multicasts a signed *commit* message. After a replica receives and accepts a *commit*, it checks the client request to the state machine and sends back the result to the client.

PBFT uses a timeout mechanism to deal with faulty primaries. Backups keep a timeout that starts every time they receive a new request. This timeout finishes when the request *pre-prepare* is received. Hence, if backups did not receive a *pre-prepare* message within the timeout for a request, they execute a *view change* protocol. PBFT has the following issues [5]. First, changing the view in case of a faulty primary is subtle and a bit complicated. Second, all previous clients' requests, since the last commit, migrated to the new view.

Tendermint

Tendermint [5, 22] is a protocol used to deliver *security* and *consistency* for replicating an application on multiple nodes. Tendermint guarantees the *security*, as it can work even if up to one-third of the nodes in the network fail in arbitrary ways. The *consistency* means that every non-faulty node can view the same transaction log and compute the same state. Tendermint is a consensus protocol that does not include Proof of Work mining, which overcomes the energy and resource consumption issues and speeds up blocks' validations [22].

Tendermint is based on PBFT, and it involves three stages of voting to reach consensus (*propose*, *prevote*, and *precommit*). A proposer *propose*s a new block, then the validators *prevote* on the block and only proceed to *precommit* if they receive more than two-thirds of *prevote*s. Likewise, validators only accept the block if more than two-thirds of *precommit*s are received. Tendermint, as mentioned earlier, is a fixed-validators protocol; that is, it requires a fixed known set of validators.

Voting on a block proceeds in rounds, where each round has a new proposer. The validators vote on whether to commit the block or advance to the next round. Tendermint is notable for its simplicity, performance, and fork-accountability [31]. However, the number of validators yields a powerful influence on Tendermint's performance. This is due to the communication overhead created by the two stages of voting (i.e., *prevote* and *precommit*). This creates a trade-off between performance and security, where more validators strengthen security. Our protocol is based on Tendermint and inherits all the features offered by Tendermint. However, it deals with the validators' selection issue by selecting a different, random set of validators on each block proposal (i.e., true decentralized).

Hyperledger Fabric

Hyperledger Fabric employs PBFT as its consensus algorithm [9]. Thus, it can tolerate up to one-third byzantine nodes in a blockchain network. In Fabric v0.6, there exists a fixed number of *validation peers* responsible for executing the consensus protocol. A proposer can submit a transaction to any of them.

Then, the chosen peer broadcasts this transaction to the other peers. One of the validation peers is selected as a *leader*.

When generating a block, the leader orders the block's transactions and broadcasts this ordered list of transactions to all validation peers. When a validation peer receives the ordered list, it proceeds as follows. First, the validation peer executes the transactions orderly. Second, after executing all transactions, it calculates the hash for the newly created block (i.e., for the executed transactions plus final state of the blockchain). Third, it broadcasts the resulting hash to all other peers and begins counting their responses. Finally, if two-thirds responses were received with the same hash, it commits the new block to its local ledger. Hyperledger Fabric, like Tendermint, suffers partial centralization since it employs a fixed known number of validation peers.

Ripple

Ripple [32] was introduced in 2012 to enable global payments using 'XRP' cryptocurrency by connecting payment providers, banks, and businesses via a network called 'ripplenet.' Ripple uses a BFT consensus protocol to maintain a valid, distributed ledger.

In Ripple consensus, a proposer collects new transactions initiated by end users and combines them into a list known as the "candidate set" (i.e., block). Each node in the network has a Unique Node List (UNL). UNL is a list of other nodes whom the node trusts. The nodes in a UNL are so-called validating nodes. Each node amalgamates the candidate sets of all nodes on its UNL, and votes on the integrity of all transactions. After 50% of the nodes approve the transactions, the candidate set is pushed further for a higher round of voting. The final round of voting needs a minimum of 80% of a node's UNL in agreement on a transaction. All transactions that fit this requirement are then written to the ledger.

Ripple, however, requires $4/5.n$ of all n validating nodes to be correct for maintaining correctness [33]. This corresponds to tolerating $f < n/5$ faulty nodes. Additionally, Ripple requires a minimal overlap among the Unique Node Lists to avoid forking. Moreover, as argued by Cachin and Vukolic [33], Ripple "is by far not as decentralized as advertised" since Ripple offers a default list of validating nodes operated by Ripple and third parties.

Algorand

Recently, Algorand [10], a new fast Byzantine fault-tolerant consensus, has been introduced to avoid the scalability and power-hungry issues presented by PoW. It combines a revamped Byzantine Agreement protocol (BA*) with a cryptographic method so-called 'Cryptographic Sortition' to propose and agree on new blocks.

Algorand advances in rounds. In each round, to propose a block, each node in the network executes cryptographic sortition to determine if it is selected to propose the block. The sortition ensures that a small number of nodes are randomly selected based on their account balance. Each selected node has a

priority and a proof. For each round, multiple nodes are selected and propose the block for this round, but the one with the highest priority is adopted.

To agree on the proposed block, Algorand uses BA* consensus. Each node initializes BA* with the highest-priority block that it received. The cryptographic sortition is, also, used to select a committee of verifiers. The verifiers are the nodes responsible for validating the proposed block and voting on it. Each verifier is required to broadcast proof of selection so any other node can verify this proof. These steps repeat until, in some phase of BA*, enough nodes in the committee (a threshold level of votes) reach consensus.

Unlike most of the previously discussed protocols, Algorand does not rely on fixed known validators, which makes is robust against DDoS and eclipse attacks. Nevertheless, Algorand provides no incentive and does not guarantee that the verifiers will adhere to the protocol. It is possible for attacks like 'denial of service' and lazy verifiers to appear in Algorand. Our protocol, in contrast, avoids this issue by leveraging a rewarding mechanism based on a game theoretical model (Section 4.2).

2.3.4 Non-Byzantine Fault Tolerance

Paxos

Paxos [34] is an asynchronous consensus protocol and is quite similar to PBFT. However, it requires only $2f + 1$ nodes to tolerate f faults [5]. In Paxos, there two parties: (1) proposer (leader), which proposes a value (block), and (2) acceptors, which accepts the value. A client can connect to the proposer to add a transaction (value) to the log (ledger). The proposer proposes the value to the acceptors and counts the votes for acceptance of the majority. The value is accepted when there is majority/quorum.

Raft

Raft [35] is a consensus protocol which has been introduced to be easy to understand. It is similar in spirit to Paxos in fault-tolerance. Raft works by electing a leader that coordinates some followers. When a client sends a request to the leader, the leader instructs its followers to append the entry. The entry is committed only when at least the majority of the followers have confirmed the appending command. Raft is used by many blockchains, such as R3 Corda [36], and Kadena [37].

Casper "The Friendly GHOST"

Casper [38] is the consensus mechanism in Ethereum [39]. It is an adjustment of some of the principles of the GHOST protocol [40] (Greedy Heaviest-Observed Sub-Tree). Casper in Ethereum was presented with security-deposit based economic consensus protocol. In other words, each node which wishes to participate in the validation and consensus process needs to have a security deposit that reflects how much stake it has. These nodes are known as "bonded validators" and

must place their security deposits prior to participating in the consensus. This way, Ethereum addresses the "nothing at stake" problem since each bonded validator has its deposit at stake. As a result, if the bonded validator misbehaves in an objectively verifiable manner, it will lose its security deposit.

In Casper, to produce a block that will be accepted by all nodes in the system, a validator bundles the new transactions in a block, validate them, and securely signs the block. Then, the validator places bets (security deposit) on the consensus process known as 'gambling on consensus.' The likelihood that this validator is chosen is directly proportional to the deposit it makes.

3 Proposed Block-Supply Chain

This section explains our Block-Supply chain in detail. The Block-Supply authenticates every product and detects counterfeit products without the need for a centralized authentication server. Rather, it involves the nodes in the authentication process utilizing blockchain technology. Every node in the blockchain has a unique pair of keys (public pk and secret sk) and is identified by its public key. There are three types of nodes in our protocol:

1. **Proposing (proposer):** This is the node which currently has the product. It creates, proposes, and broadcasts the new block to the network.
2. **Validator:** This node is responsible for validating the newly proposed block. Moreover, validators communicate their votes on the block to reach consensus.
3. **Idle:** This node does nothing except wait for the decision to be reached by validators on whether to accept or reject the proposed block. All other nodes in the network are idle.

The Block-Supply chain has two phases, the initialization phase, and the verification phase. The products' manufacturer executes the initialization phase, and the supply chain nodes execute the verification phase. Each product is occupied by an NFC tag, which contains the product's details such as serial number, name, and expiration date.

3.1 Initialization Phase

This first phase is responsible for initializing the details of each product, securing them and storing them on the product's NFC tag. Every NFC tag has a read-only unique tag ID (TID) and a read-only *counter*. The *counter* is increased automatically on each reading of the tag. This factor can be used to keep track of the number of times that the tag is read by the nodes.

The manufacturer executes this phase. It forms the product's details ($PDetails$), which includes the following:

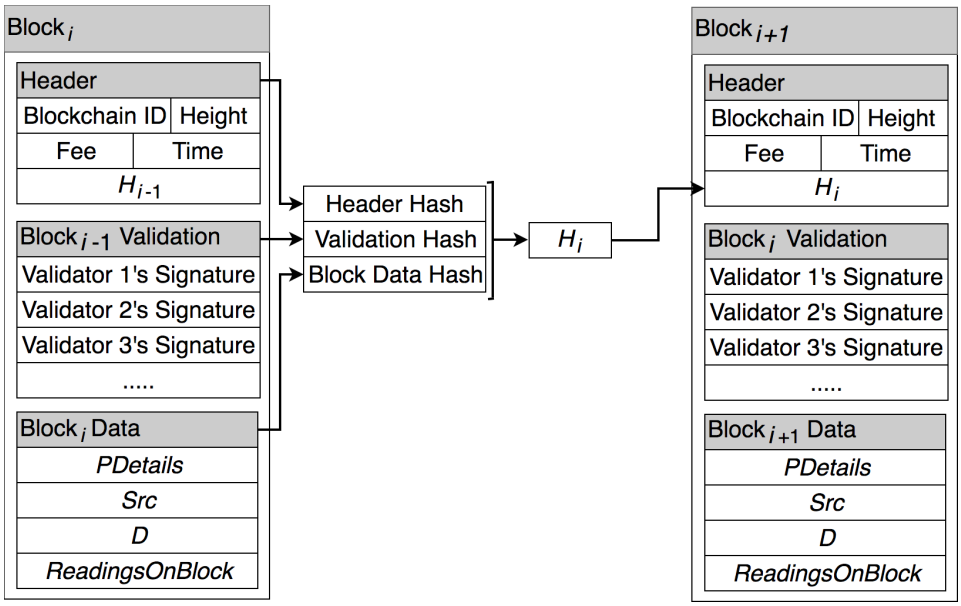

Figure 1: Blocks' structure and chaining them. Two blocks are chained by including the hash of the previous block in the new block.

- The unique product ID (PID).
- The product name ($PName$).
- The product expiration date ($PExpiryDate$).
- A field called ($ToSignTID$), that is equal to the read-only (TID).

Then, the manufacturer digitally signs $PDetails$ using its private key ($m.sk$) to produce the product's data digital signature ($SignedPDetails$). After that, it writes the product data ($PDetails$) and its signature ($SignedPDetails$) to the product's tag.

Once the tag is prepared, the manufacturer creates a *genesis block* for the product. Each block in our system represents an event of a successful product's authentication. A valid block is added to the blockchain if it is validated by a number of validators in the network. Figure 1 shows the block structure and how the blocks are chained. A block contains three parts:

1. The block header, which includes the following:
   - The blockchain ID (to identify each product's blockchain).
   - The block height (order) in the blockchain.
   - The fee to be paid to the block's validators.
   - The time stamp.

- The hash of the previous block (H_{i-1}).

2. The validation of the previous block to provide evidence of its validity, that is the data on that block is valid and sufficient signatures are included. This part includes the digital signatures of all the validators of the previous block.

3. The block data, which contains:

   - The product's details ($PDetails$).
   - The shipping source node address (Src).
   - The node the product is being shipped to address (D).
   - The current number of reads on the product's tag ($ReadingsOnBlock$) to track how many times the product's tag has been read (this field is for detecting the tag reapplication attack).

Finally, the manufacturer distributes the genesis block to every node in the network, and ships the product to the supply chain. The manufacturer is the initiator of the product's blockchain and is *no longer involved in authenticating the product.*

3.2 Verification Phase

This phase is executed by the supply chain nodes using blockchain. As a product flows throughout the supply chain, its blockchain grows and gets updated every time it leaves a node and moves to the next by adding new blocks to it. When a node (D) receives a product, two types of authentication are performed: local and global.

3.2.1 Local Authentication

The node D (i.e., the node that currently has the product) authenticates the product locally as follows:

1. Read the product's tag. Then, given the product data ($PDetails$), verify its signature ($SignedPDetails$) using the manufacturer's public key ($m.pk$) to detect **modification** on $PDetails$ as follows:

   $$Verify_{m.pk}(PDetails, SignedPDetails)$$

   If pass, proceed; if not, abort.

2. Check if the $PDetails$ on the tag is the same as the $PDetails$ on the last block in the node's own copy of the product's blockchain.

3. Check if the tag ID ($ToSignTID$) included in $PDetails$ is equal to the read-only tag ID (TID). Then, compare the $ToSignTID$ and PID on the tag to the corresponding ones on the last block in the blockchain. This is to detect **cloning** of $PDetails$. If pass, proceed; if not, abort.

4. Check if the value of the tag's read-only ($Counter - 1$) is equal to the value of $ReadingsOnBlock$ included in on the last block of the product's blockchain. This is to detect **tag reapplication**. Note that we subtract one from the $Counter$ value because of the node D's reading. If they are equal, then this means that no reads have been performed on the product's tag between Src and D. If pass, proceed; if not, abort.

If the local authentication succeeds, the product is authentic, and node D proceeds accordingly. If the product is counterfeit, node D broadcasts this finding to the other nodes in the network so further inspection can be conducted to find the origin of the counterfeiting attack. Unlike centralized supply chains that depends on an authentication server, the node in our Block-Supply, which has the product (i.e., proposer), can check for all three types of attacks (modification, cloning, and reapplication) by relying only on its own copy of the product's blockchain (assuming a valid blockchain). There is no need for a remote authentication at this point. This is because the product's blockchain serves as a distributed database that contains valid information about the product. Hence, it is safe for the node to consult this blockchain. The data on the product's tag and blockchain is sufficient to execute the local authentication algorithm and check for the three attacks. Ensuring that the blockchain is valid is the responsibility of the validators, which will be explained in the following section.

3.2.2 Global Authentication

After successful *local authentication*, and prior to dispatching the product to the next node, the node becomes a *proposing node* and proposes a new block. The proposed block contains the following:

- The new source node (Src), which is the address for the current proposing node.
- The destination node (D), where the product is going to be shipped to.
- The current number of reads of the tag's *counter* ($ReadingsOnBlock$).
- The hash of the previous block (H_{i-1}).
- The digital signatures of the previous block's validators.

The proposing node, then, broadcasts this new block to every node in the network. The validators validate the block globally and vote on it.

Each validator executes *global authentication*, which includes the following steps:

1. The validators ensure that the blocks in the product's blockchain are well-chained and have the appropriate order. This is done by hashing the previous block ($block_{i-1}$) and comparing it to the H_{i-1} that is included in the proposed block ($block_i$).

2. They check if $PDetails$ in the previous blocks is the same as the one in the proposed block.

3. They trace-and-track the product by ensuring that the source (Src) in a block ($block_i$) is equal to the destination (D) in its previous block ($block_{i-1}$). This is to ensure that the product has been supplied through certified, valid nodes in the supply chain.

If the three checks are successful, the block is valid, and it is safe to include it in the blockchain. The validators ensure that the blockchain always contains valid blocks so the next *proposing node* can safely rely on the last block when executing its *local authentication*.

This far, we have only discussed the role of validators, but how do we select them and how they interact to reach an agreement on a block validity? Our proposed consensus protocol answers these two questions in the following section.

4 The Proposed True Decentralized Consensus Protocol

In this section, we propose a new consensus protocol that exploits randomness and game theory to achieve true decentralization security with respect to efficiency. Our protocol is based on Tendermint and exploits its capability to overcome up to one-third of Byzantine faults. Unlike other protocols that rely on a fixed, static set of validators responsible for validating all proposed blocks, our protocol randomly selects a different set of different size of validators each time a new block is proposed. Thus, our protocol improves the security, since the validators are not known before proposing the new block. This factor makes the job more difficult for an adversary to attack or to bribe the set of validators.

Each node in the blockchain has a unique pair of keys (public pk and secret sk) and is identified by its public key. Moreover, each node has a public trust (reputation) value (R) where this value affects the selection of a node to be validator over time. There are four types of nodes in our protocol:

1. **Proposing (proposer):** This is the node which executes the *local authentication* algorithm in our Block-Supply chain (Section 3.2.1). After successful authentication, it creates, proposes, and broadcasts to the network the new block for the product.

2. **Validation-leader (leader):** This is the node responsible for selecting the random set of validators for the proposing node.

3. **Validator:** This node is responsible for validating the newly proposed block by executing the *global authentication* algorithm (Section 3.2.2). Moreover, validators communicate their votes on the block to reach consensus.

4. **Idle:** This node does nothing except wait for the decision to be made by validators on whether to accept or to reject the block. All other nodes in the network are idle.

Our protocol works in two phases, the initialization phase and the verification (validation) phase. The blockchain initiator executes the first phase at the genesis state, in which it randomly maps each proposer to its validation-leaders. In the second phase, each node becomes a proposer in a round-robin fashion. When a node is a proposer, it proposes a block, broadcasts it to all nodes, and its corresponding validation-leaders randomly select the validators to verify (validate) this block. Next two subsections present an in-depth description of how these two phases are executed.

4.1 Initialization Phase

This phase's main task is *mapping proposers to validation-leaders*. At the genesis state (i.e., when the genesis block is created), the blockchain initiator randomly maps four validation-leaders to each proposer in the network. The reasoning behind this choice is that four is the minimum number to provide tolerance to a single Byzantine fault [5]. As our protocol is based on Tendermint, it is assumed that a Tendermint network has two-thirds of non-Byzantine nodes. A simple approach is to employ only one validation-leader per a proposer; however, to ensure the safety and liveness of the consensus process, we need to utilize more. It is worth noting that this number (i.e., four) can be changed based on factors such as the network's size and hostility, or the blockchain application that utilizes our protocol. Our approach works with any number of validation-leaders per proposer other than four, but we utilize the minimum in favor of efficiency. Additionally, this number can be a random number to further increase robustness.

The mapping is executed randomly according to the nodes' weights (reputations R). As shown in Algorithm 1, we use the Weighted Random Sampling (WRS) algorithm [41]. The weights in our algorithm are the nodes' reputation values. Furthermore, this mapping is anonymous and done blindly; that is, no proposer knows its corresponding leaders and no leader knows its proposer until executing the consensus protocol. This way, we prevent a malicious proposer from corrupting or bribing its leaders and vice versa.

To accomplish the anonymous mapping, the blockchain initiator, **first**, includes a secret (S_1) in every node's genesis block, so it uses this secret when the node becomes a proposer. S_1 is a hash that includes the proposer's public key ($pr.pk$), all the four selected the leaders' public keys $[vl_1.pk - vl_4.pk]$, the blockchain ID ($BlockchainID$), and a random number ($Rand_1$) as follows:

$$S_1 \leftarrow hash(pr.pk||vl_1.pk||vl_2.pk||vl_3.pk||vl_4.pk||blockcahinID||Rand_1)$$

Note that there is only one proposer secret S_1. Each proposer in the network has its own (S_1). This secret is checked by each of the four validation-leaders. **Second**, the blockchain initiator generates a validation-leader's secret (S_2). S_2

is a hash that includes the proposer's secret S_1, and a random number ($Rand_2$) as follows:

$$S_2 \leftarrow hash(S_1||Rand_2)$$

Here, we use different $Rand_2$ for each leader to make S_2 different for each one of them. Note that $Rand_2$ is private and is only known to its particular leader node.

To ensure that a validation-leader is *legitimate* and that it has been elected by the blockchain initiator, we need to utilize a verifiable proof (π). This proof is a digital signature signed by the initiator using its private key ($in.sk$). The proof π includes the proposer's public key ($pr.pk$), the validation-leader's public key ($vl.pk$), and the blockchain ID ($BlockchainID$), as below.

$$\pi \leftarrow Sign_{in.sk}(pr.pk||vl_i.pk||blockcahinID)$$

The validation-leader must submit this proof to its elected validators so that each can verify π using the initiator's public key ($in.pk$) prior to involving in the validation and consensus process. This protects against 'malicious nodes' claiming that they are 'validation-leaders' for a proposer.

As mentioned, for one proposer, there exist four leaders responsible for selecting the validators for the block proposed by this particular proposer. This raises a new problem of selection conflict, since each validation-leader selects the validators blindly without knowing its peer leaders. Consequently, the four leaders perform the validators' selection from the same pool of nodes without any communication or agreement between them. This can result in selecting a validator more than once by different leaders. Our protocol overcomes this problem by dividing the pool of nodes into four pools, each of which is assigned to a leader. Specifically, each validation-leader will have a range (g) to choose from determined at the genesis state. Note, we assume that all the nodes in the network have the same set of nodes in the same order. As shown in Algorithm 1, g is predetermined by the blockchain initiator and is defined below:

$$g \leftarrow [((i-1).\tfrac{n}{4}) + 1 \ , \ i.\tfrac{n}{4}]$$

Where $1 \leq i \leq 4$ and is the index of a validation-leader among its peers.

In Algorithm 1, there are three lists. The first list (A) is a population of n nodes each of which has a reputation value R. The second list (B) is a temporary list for a proposer to hold the public keys for the selected validation-leaders; this list is flushed after selecting the validation-leaders and initializing their secrets and proofs. The last list (C) is for a validation-leader. There exist four corresponding proposers for each validation-leader. Thus, C stores four tuples, and each of them corresponds to one proposer. Each tuple includes the secret (S_2), the random number ($Rand_2$), the proof (π), and the range (g). By the end of executing Algorithm 1, each node in the network will have exactly one proposer's secret (S_1) used when the node becomes a proposing node, and a list (C) used whenever this node becomes a validation-leader for one of its four proposing nodes. This concludes the initialization phase.

Algorithm 1: Proposers to leaders Mapping

Input : A population A of n nodes having reputation values

foreach $pr \in A$ **do**
  for $k \leftarrow 1$ **to** 4 **do**
    Try:
    $p_i(k) \leftarrow \frac{R_i}{\Sigma_{s_j \in A-B} R_j}$
    Randomly select vl_i **with** probability $p_i(k)$ **from** $A - B$
    if $C_i.size > 4$ **then**
      Go to **Try**
    else
      $B.add(vl_i.pk)$
    end
  end
  Randomly generate $Rand_1$
  $S_1 \leftarrow hash(pr.pk||vl_1.pk||vl_2.pk||vl_3.pk||vl_4.pk||blockcahinID||Rand_1)$
  Append S_1 to the pr's genesis block
  foreach $vl_i \in B$ **do**
    Randomly generate $Rand_2$
    $S_2 \leftarrow hash(S_1||Rand_2)$
    $\pi \leftarrow Sign_{in.sk}(pr.pk||vl_i.pk||blockcahinID)$
    $g \leftarrow [((i-1).\frac{n}{4}) + 1\ ,\ i.\frac{n}{4}]$
    $C_i.add(S_2||Rand_2||\pi||g)$
  end
  Flush B
end

4.2 Verification (Validation) Phase

This phase is executed upon proposing a new block. It is carried out by three parties (proposer, validation-leaders, and validators). The main purpose of this phase to decide the validity of the newly proposed block and to reach a consensus on this decision.

When a node becomes a proposer, it broadcasts its secret (S_1) to all nodes in the network. Every other node checks if it is a validation-leader for this proposer by looping through its list (C) and hashing the received S_1 and each private random number ($Rand_2$) it has. If the resulting hash matches its secret (S_2), then this node is a leader for this proposer as shown in Algorithm 2.

4.2.1 Deciding the Number of Validators (M)

Each validation-leader decides its number of validators (m), of which $m < n$ where n is the total number of nodes in the network. Our protocol utilizes a game theoretical approach to determine the risk likelihood threshold ($risk$) of each proposer. In this game, we treat a proposer as a potential attacker (payer x). The defenders in this game are the corresponding leaders (player y).

There are two benefits to this game theoretical approach. First, according to the outcomes of the game, the validation-leaders can assess the *risk* of their proposer. Consequently, based on this *risk*, they determine the number of validators to select (i.e., m), which is proportional to this *risk*. Second, the game rewards the honest players and penalizes the dishonest ones, which results in adhering to the protocol and mitigating the malicious behavior.

Game Model

In this game, player x could be of two type: a) *malicious*, or b) *regular*. Since the validation-leaders do not know the type of player x (i.e., regular or malicious), we model our game as a Bayesian game. Table 1 shows the notation used in our game theoretical approach. Table 2 shows the payoff matrix of the game when player x is of type *malicious*. For each cell in the payoff matrix, the first payoff is for player x and the second one is for player y. Table 3 shows the payoff matrix of the game when player x is of type *regular*. We assume that the players are rational and they play the best strategy they have to maximize their payoffs.

The Bayesian game introduces a third player called Nature (denoted by N), which determines the type of player x by assigning a probability (μ) to player x of being *malicious*. Figure 1 represents the Bayesian game extensive form. In this game, player x will try to play a strategy to minimize the chances of being detected, and player y will also try to play a strategy to maximize the chance of detecting the cheating without much cost.

This game does not have a *pure-strategy* **Bayesian Nash Equilibrium (BNE)**. To derive the game mixed-strategy BNE, we let p be the probability that player x plays $Cheat$, and q be the probability which player y plays $AddMoreValidators$. After some analysis, we found that player y plays $AddMoreValidators$, if:

$$p > \frac{w_y + 2\gamma}{\mu\beta(g_y - c_y) + \mu\gamma} \tag{1}$$

Algorithm 2: Validation-leader checking

Input : The node's list C, and the received proposing node's secret S_1
Output: A decision of weather or not this node is a validation-leader

```
decision ← false
foreach tuple_i ∈ C do
    if S_2^i = hash(S_1||Rand_2^i) then
        decision ← True
    end
end
Return decision
```

Table 1. The game notation.

Symbol	Definition
β	Importance of the proposer. We assume that some proposing nodes in the blockchain network have higher criticality than others. For example, in applications like supply chains, a node that supplies products (and proposes blocks for these products) to 10 other nodes is more important than a node that supplies products to one node. Thus, β influences the payoffs for both players.
γ	A reward that player y can get if it maintains the performance of the consensus process under a cretin threshold by playing $UseMinimumValidators$. However, player y can loose γ (i.e. deducted from his gain g_y) if it plays $AddMoreValidators$ and the performance violates the specified threshold. We assume that player y will not win γ in case of a successful attack (i.e. player x plays $Cheat$ and player y plays $UseMinimumValidators$).
w_x	Work done by the proposing node (player x) to play $Cheat$.
g_x	The gain for player x from a successful attack.
c_x	The cost (risk) for player x if captured.
w_y	Work done by the validation-leader (player y) to play $AddMoreValidators$.
g_y	The gain for player y from capturing a cheater, in case the validation-leader employed more validators.
c_y	The cost (risk) for player y if fails to capture a cheater.
μ	The probability of player x being malicious.
N	The nature node, which determines the type of player x.

Table 2. Strategic form of the Bayesian game (Player x is malicious)

Game Matrix		**player y (Validation-leader)**	
		$AddMoreValidators$	$UseMinimumValidators$
player x	$Cheat$	$(\beta.c_x) - w_x$, $[(\beta.g_y) - \gamma] - w_y$	$(\beta.g_x) - w_x$, $\beta.c_y$
	$NotCheat$	0, $-w_y - \gamma$	0, γ

Table 3. Strategic form of of the Bayesian game (Player x is regular)

Game Matrix		**player y (Validation-leader)**	
		$AddMoreValidators$	$UseMinimumValidators$
player x	$NotCheat$	0, $-w_y - \gamma$	0, γ

Player x plays $Cheat$, if:

$$q > \frac{w_x - (\beta g_x)}{\mu\beta(g_y - c_y)} \tag{2}$$

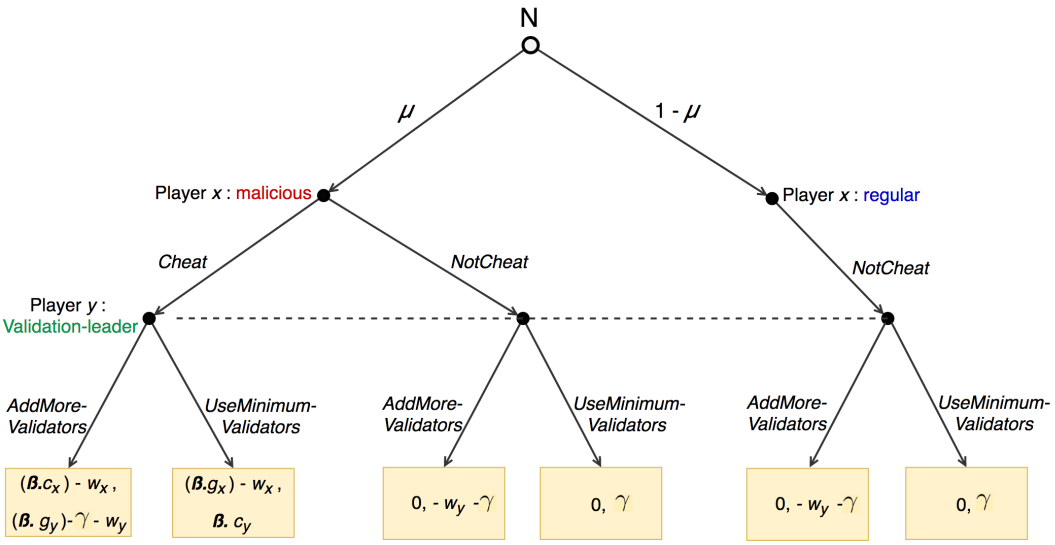

Figure 2: Extensive form of first-stage Bayesian game.

Thus, our game's mixed-strategy **BNE** is: ((*q* if malicious, $NotCheat$ if regular), p, μ).

p is the probability that the proposing node (player x) might attack (plays $Cheat$). In response to this probability, a validation-leader (player y) chooses the appropriate strategy that will maximize its payoff (i.e., whether or not to $AddMoreValidators$). Hence, we consider p as the "*risk likelihood*" (i.e., $risk = p$) of an attack.

In case a validation-leader chooses to play $AddMoreValidators$, the number of validators (m) will be a random number bound by the minimum number of validators (i.e. four) and a fraction of $\frac{n}{4}$ proportional to p (we choose $\frac{n}{4}$ because we have four validation-leaders). In other words, a validation-leader selects a random number between 5 (the minimum number of validators plus one) and $\frac{p.(n-2)}{4}$ (excluding the proposing and the validation-leader nodes).

After a validation-leader decides its m, it selects its validators, instructs them, and broadcasts m to all nodes. When a node in the network receives all the ms from the validation-leaders, it calculates the overall number of the validators involved in the protocol (M) as follows:

$$M = \Sigma_{i=1}^{4} m_i$$

This way the overall number of validators is proportional to $risk$. Note that our protocol inherits the Byzantine tolerance provided by Tendermint. In other words, the system can work with one faulty leader, of which M is the aggregation of only three ms. In case of more than one faulty leader, each node in the network waits for a time period named "*leader-time-out*" and then switches to "*all-validate*" mode. In this mode, every node in the network votes

Algorithm 3: Validators' Selection

Input : A population V of $\frac{n-2}{4}$ nodes having reputation values, AND the risk likelihood $risk$

Output: A set of validators/pre-voters PV and a set of pre-committers PC of size m

if $AddMoreValidators$ **then**
  $m = Random[5, \frac{risk.(n-2)}{4}]$
else
  $m \leftarrow 4$
end
for $k \leftarrow 1$ **to** m **do**
  $p_i(k) \leftarrow \frac{R_i}{\Sigma_{v_j \in V-PV} R_j}$
  Randomly select v_i **with** probability $p_i(k)$ **from** $V - PV$
  $PV.add(v_i)$
end
for $l \leftarrow 1$ **to** m_i **do**
  $p_i(l) \leftarrow \frac{R_i}{\Sigma_{c_j \in V-PC} R_j}$
  Randomly select c_i **with** probability $p_i(l)$ **from** $V - PC$
  $PC.add(c_i)$
end
Return PV **AND** PC

on the received M to agree on it. This mode is costly but preserves the consensus liveness.

4.2.2 Selecting Validators

Each validation-leader selects its set of m validators. The four sets of selected validators will be responsible for validating the proposed block. Our protocol is based on Tendermint which involves two steps of voting (pre-vote and pre-commit). The validators are selected randomly, and each set of selected validators is only known to their validation-leader. A validator is only known to the other nodes in the network when it contributes to one of the voting steps. Therefore, an adversary can observe the validators after revealing their identities in executing the first stage of voting (i.e., pre-voting). As a result, a powerful adversary might be able to attack or corrupt a sufficient number of them, which can result in not executing the second step of voting (i.e., pre-committing). In response to this issue, our protocol requires each validation-leader to select two sets of nodes of size m. The first set is the validators/pre-voters, and the second one is the pre-committers. The pre-voters are responsible for executing the first step of voting, and the pre-committers execute the second step. As a result, the adversary discovers a participating node in the voting only after giving its vote, which is unuseful knowledge. Algorithm 3 shows the process of selecting the validators/pre-voters and pre-committers.

After selecting the validators and pre-committers, each validation-leader needs to include a proof of eligibility (τ) for each selected node to prove that a legitimate validation-leader has selected this node. τ is a digital signature signed by the validation-leader's private key ($vl.sk$) and includes the validation-leader's public key ($vl.pk$), the selected node's public key ($pv.pk$ for a pre-voter and $pc.pk$ for a pre-committer), and the validation-leader's proof (π) as follows:

$$\tau \leftarrow Sign_{vl.sk}(vl.pk||pv.pk||\pi)$$

A node which receives a vote accompanied by τ from a voting node (i.e., pre-voter or pre-committer) needs to perform two verifications. First, it needs to verify τ using the validation-leader's public key ($vl.pk$). Second, after successful verification of τ, the node verifies π using the initiator's public key ($in.pk$).

4.2.3 Consensus Mechanism

Thus far, we have covered how the validators are selected. However, these validators need a way to reach consensus in the presence of Byzantine nodes. Our protocol is based on Tendermint and exploits its capability to overcome up to one-third of Byzantine faults. The validators in our protocol *pre-vote* on the proposed block, and when they hear from more than two-thirds of M other nodes, they *pre-commit* the block. The block is committed when more than two-thirds of M *pre-commit*s are received. The consensus algorithm is summarized as follows:

1. When a node in Block-Supply chain becomes a proposing node, this node broadcasts its secret S_1 to all nodes. After that, when this node is ready to *propose*, it creates the new block and broadcasts it to the network.

2. Upon receiving S_1, each node in the network checks if it is a validation-leader, as mentioned earlier. If a node is a validation-leader, then it determines how may validators to select (i.e., m) based on the outcomes of the played game. Then, it executes the "validators' selection" algorithm to randomly select m *validators/pre-voters* and *pre-committers*. After that, it sends a *validate* message to the selected validators, and a *pre-commit* message to the selected pre-committers. In addition, each leader broadcasts its m to all nodes.

3. After receiving all the ms from the leaders, the remaining nodes wait for a "*validator-time-out*" period. The *validator-time-out* is the time that every node waits to hear a *validate* message from the validation-leader node, or a *pre-vote* message from a validator. There are three possibilities a node might act in this step:

   (a) If the node receives a *validate* message, then it acts as a validator and carries on the validation process.

   (b) If the node receives a *pre-vote* message, then it is an idle node, and it waits to hear the remaining two-thirds *pre-votes*.

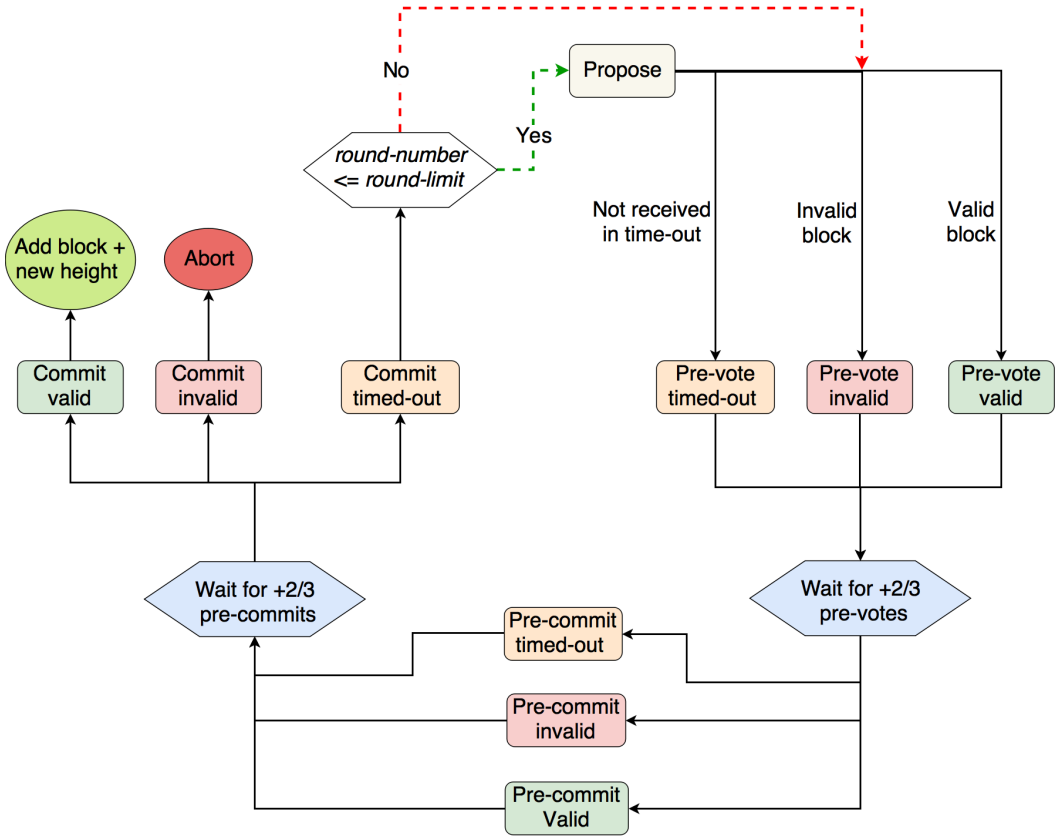

Figure 3: Our consensus protocol. The first step is "Propose." After this step, validators only advance after hearing from two-thirds or more of M other nodes. By the end of the cycle, either the block is committed, and the blockchain height increases, or the proposed block is rejected and the protocol aborts.

(c) If the node receives neither, then it acts as an "*all-validate*" validator and carries on the validation process. *All-validate* is an alternative mode to the M-validators mode that we described so far. This mode allows all nodes to participate in reaching a consensus. It is a particular case that preserves the liveness of the protocol.

4. After receiving the *validate* message, the validators wait for a *proposer-time-out* to receive the proposed block. This time-out protects the protocol's liveness from faulty proposing nodes. The validators begin this step by initializing the voting's *round-number* to zero. Each validation event (i.e., executing the consensus protocol) is a round. The validators' votes depend on two factors: a) whether or not they receive the proposed block within the *proposer-time-out*, and b) whether or not the proposed

block is valid. If a validator receives the proposed block in the *proposer-time-out*, it validates the block, and *pre-vote*s 'valid' if the block is valid or 'invalid' otherwise. However, if the *proposer-time-out* terminates, then the validator *pre-vote*s 'timed-out' as illustrated in Figure 3.

5. When a validator receives more than $\frac{2.M}{3}$ *pre-vote*s, it *pre-commit*s 'valid', 'invalid', or 'timed-out' according to the received *pre-vote*s type.

   Note that our consensus algorithm is based on Tendermint and exploits its capability to overcome up to one-third of Byzantine faults; that is up to $\frac{1.M}{3}$ of faulty validators. If more than $\frac{1.M}{3}$ validators fail to pre-vote, then the consensus will stall and the pre-committers will not be able to proceed. As a result, Tendermint introduced the concept of *validation rounds* to overcome such vulnerability. For example, when $\frac{2.M}{3}$ pre-votes are not received on time, a new round is initiated to maintain the protocol's liveness. Put differently, the proposer will start a new round by proposing the block again giving a chance for the network's nodes to reach consensus on the block.

6. When a validator receives more than $\frac{2.M}{3}$ *pre-commit*s, it *commit*s 'valid', 'invalid', or 'timed-out' according to the received *pre-commit*s type. The remaining nodes *commit* when they receive more than $\frac{2.M}{3}$ *pre-vote*s followed by $\frac{2.M}{3}$ *pre-commit*s.

7. There is a final subsequent step follows the *commit* step, and it is of three types. First, if a node *commit*s 'valid,' then it adds the proposed block to the blockchain and extends it to a new height. Second, if the node *commit*s 'invalid,' then it aborts the protocol. Third, if the *commit* is of type 'timed-out,' the idle and validation-leader nodes do nothing, they wait to hear again from the validators. On the other hand, the validators check the *round-number* against a rounds' counter we call *rounds-limit*. If the *round-number* is less than or equal to the *rounds-limit*, the validators: a) increase the *rounds-limit* by one, b) increase the *proposer-time-out* based on the network conditions, and c) start a new round giving a chance for the proposing node to re-propose. However, if the *round-number* is greater than the *rounds-limit*, the validators *pre-vote* 'invalid'.

It is worth mentioning that the proposed block and all types of messages (i.e., *validate*, *pre-vote*, and *pre-commit*) are digitally signed by the sender using its private key, and verified by the receiver using the sender's public key.

5 Dynamic Proposers-Leaders Mapping

Our proposed protocol (Section 4) was initially designed for our Block-Supply chain application (Section 3). In this protocol, the *static* proposers-leaders' mapping, performed at the genesis state, guarantees the anonymous mapping

only for one *cycle* of proposing. In other words, when a proposer proposes a block for the second time, its leaders' identities are revealed. Hence, they could be vulnerable to attacks like DDoS, eclipse, and bribery. Consequently, we need to involve the blockchain manufacturer to execute the proposers-leaders mapping at the beginning of each cycle. This is suitable for our Block-Supply chain use case, since each product flows through the supply chain nodes and gets authenticated only once by every node. As mentioned in Section 3, one successful authentication event corresponds to one block added to its the product's blockchain. When the product reaches the market (i.e., end customer), none of the previous nodes in the supply chain will repropose a new block for this product. Thus, for an application that strict their blockchain's nodes to propose only once the *static mapping*, which is executed by the blockchain initiator, is adequate and guarantees the anonymous mapping.

In blockchain applications that allow the nodes to propose more than once, relying on static mapping is tedious and yields partial centralization. This is due to involving the blockchain initiator or some special nodes in the network to map the proposers to their leaders on every cycle of proposing (i.e., when nodes need to repropose again). As a result, to make our protocol suitable for any other blockchain applications, we need to make the proposers-leaders mapping dynamic in a way that preserves the anonymity.

In this section, we propose a new dynamic proposers-leaders mapping mechanism that does not rely on any centralized proposers-leaders mapper. Instead, the regular nodes in the network perform the mapping dynamically and anonymously. Our dynamic mechanism is executed in two phases: 1) initialization at the genesis state, and 2) during the verification phase of our protocol.

5.1 Initialization Phase

As in our protocol, the blockchain initiator executes this phase at the genesis state. This phase is performed only once. After the genesis state our protocol proceeds utilizing the dynamic proposers-leaders' mapping in a distributed manner. It is worth mentioning that this dynamic approach is executed along with executing our protocol (Section 4). The blockchain initiator uses its pair of keys (i.e., public $in.pk$ and private $in.sk$) to perform this phase.

Each node in the network has two pairs of keys. The first one (pk_1, sk_1) is to identify the nodes, communicate with other nodes, and sign proposals and votes. The second pair (pk_2, sk_2) is used for the dynamic anonymous proposer-leaders' mapping. At the geniuses state, the blockchain initiator executes this phase as follows:

- Generates a secret ID (SID) for each node. This ID is appended to the node's geniuses block and is only known to this node. This SID is associated with the public key of the second pair of keys (i.e., pk_2).
- Creates a publicly known list that holds all the SIDs and their corresponding pk_2s; we call this list $SList$. Each node in the network has this list.

Algorithm 4: Dynamic Mapping Initialization Phase

Input : A population A of n nodes having their pk_1s and pk_2s

foreach $node \in A$ **do**
  Generate a unique SID
  Append SID to the $node$'s genesis block
  $SList.add(node.pk_2, node.SID)$
end
Create $CycleSlots$, where $r = 2$ and $s = [1, n]$
foreach $node \in A$ **do**
  Append $SList$ to the $node$'s genesis block
  Append $CycleSlots$ to the $node$'s genesis block
end

- Creates another list ($CycleSlots$) that represents the **proposing slots** in the **next cycle**. Each cycle consists of many slots. For example, a cycle may have slots equal to the number of nodes in the network. For each slot, a new block is proposed. Therefore, each cycle slot (cs) in this list can be selected only once, that is in a cycle slot, only one block is proposed. A cycle slot has two fields:
  - The cycle number (c): Each cycle has a number. This number is used to identify a cycle. The number of cycles can be finite or infinite.
  - The slot number within a cycle (s): As mentioned earlier, each cycle has a finite number of slots, each of which is identified by a number.

  An example of a cycle slot (cs) is (3,10), which identifies the slot number 10 in cycle 3. When a proposer is selected for that cycle slot, the proposer will propose a new block. Upon selecting a cs, we exclude this cs from the list $CycleSlots$.

Note that this initialization phase is only executed once. In this approach, each cycle of proposing/validating gets its anonymous proposers-leaders' mapping from the previous cycle as will be explained in the next section. Nevertheless, the anonymous mapping for the first cycle is static. After the first cycle, the protocol proceeds independently utilizing the dynamic approach to assign the anonymous mapping. Algorithm 4 illustrates the process of the dynamic mapping's initialization phase. Note that we choose the number of slots in our system to be equal to the number of nodes (n).

5.2 Dynamic Mapping During the Verification Phase

This phase takes place during the verification phase of our protocol. As mentioned earlier, each cycle obtains its mapping from the previous one. That is, the *current* proposers and leaders propose the mapping for the next *new* cycle and the *current* validators agree on this mapping. Note, we refer to the nodes

that are currently executing the protocol (i.e., proposers, leaders, and validators) as *current*. On the other hand, we refer to the nodes that will execute the protocol for the next cycle as *new*. Three entities execute the dynamic mapping:

1. The *current* proposer: This node selects and proposes a *new* proposer for a cycle slot (cs) in the next cycle.
2. The *current* corresponding leaders: They are responsible for selecting the *new* leaders of this cs for the newly selected proposer.
3. The *current* validators: These nodes assign the mapping for the *new* selected proposer/leaders and vote on it.

Following subsections explain in detail how each of these entities performs their role.

5.2.1 New Proposers Selection

In our protocol, the proposing nodes propose the new blocks, their leaders select the validators for that block, and the chosen validators validate the new block and reach consensus on it. The mapping between the proposers and their leaders is anonymous which contributes to greater security. In this section, we add a new task for the current proposers, which is selecting a new proposer for a slot in the next cycle. Algorithm 5 shows the anonymous new proposer selection, in which the current proposer does the following:

1. Randomly select a new proposer's secret ID ($PrSID$) from its the public list that contains all the secret IDs (i.e., $SList$). Once the secret ID is selected, obtain the the second public key ($PrSID.pk_2$) for this selected $PrSID$. Note that each tuple in $SList$ has two fields: a) the secret ID (SID), and b) its corresponding second public key (pk_2).
2. Generate a random number ($Rand_0$).
3. Randomly, select a cycle slot (cs) form the list $CycleSlots$.
4. Produce an anonymous new proposer ($NewPr$) by encrypting the $PrSID$, $Rand_0$, and cs with the new selected proposer's second public key ($PrSID.pk_2$) as below:

$$NewPr \leftarrow E_{PrSID.pk_2}(PrSID||Rand_0||cs)$$

Note that no one can reveal the real identity of this selected proposer, since the nodes are only identified by their first public keys (i.e., pk_1s), which is not involved in the dynamic proposers-leaders mapping at all. Although the second public keys are publicly published, the correlation between the second public key (pk_s) and the first public key (pk_1) for a node is invisible. The only node in the network that can reveal the identity of this $NewPr$ is the node that has the second private key (sk_2), which corresponds to the selected second public key in step 1 (i.e., $PrSID.pk_2$). Furthermore, sk_2 is private and only known to the owner node.

Algorithm 5: Selecting a New Proposer and its Round Slot

Input : $SList$ and $CycleSlots$
Output: $NewPr$, cs, and $[gs_1, gs_4]$

Let $PrSID = null$
Randomly select SID from $SList - current.SID$
$PrSID \leftarrow SID$
Randomly select cs from $CycleSlots$
Generate $Rand_0$
$NewPr \leftarrow E_{PrSID.pk_2}(PrSID||Rand_0||cs)$
for $i \leftarrow 1$ **to** 4 **do**
  $gs_i \leftarrow [((i-1).\frac{SList.size}{4}) + 1\ ,\ i.\frac{SList.size}{4}]$
end
Return $NewPr$, cs, and $[gs_1, gs_4]$

5. As discussed in Section 4.1, to avoid selecting a *validator* by more than one *leader*, we assign each node a pool of nodes (i.e., g) to choose from when it becomes a leader, where each g contains different nodes. Similarly, in our dynamic mapping approach, each node will have a pool that includes the secret ID's (SIDs) for the nodes from whom this node can select. We call this list sg. sg is introduced in the dynamic mapping to avoid conflict in choosing the leaders for the new cycle slot selected cs. As a result, the current proposer creates four ranges (i.e., four sgs) each of which is assigned to one of its leaders as follows:

   $$gs_i \leftarrow [((i-1).\frac{SList.size}{4}) + 1\ ,\ i.\frac{SList.size}{4}]$$

   Where $1 \leq i \leq 4$ and is the index of the current leader among its peers. Each leader is numbered by i. For example, the range gs_3 is for the leader number 3 to avoid conflict.

   A current leader can only select its new leader from the range in the list $SList$ that has been assigned to it to avoid choosing one new leader by more than one current leader.

6. Sign and broadcast $NewPr$, cs, and the four ranges (i.e., $[gs_1, gs_4]$).

It is worth mentioning that the anonymous new proposer selection is made simultaneously with proposing the current block.

5.2.2 New Leaders Selection

After receiving the newly selected proposer and its cycle slot, each current leader of the current proposer anonymously selects a new leader for this new proposer. As stated earlier, each current leader chooses a new leader from a range of nodes different from the ones its peer leaders have to evade selecting the same

Algorithm 6: Selecting a New Leader for the Selected Proposer

Input : $SList$, $NewPr$, and cs
Output: $NewVl_i$, $S_{i.1}$, and $Rand_{i.2}$

Let $VlSID = null$
Randomly select SID from $SList - current.SID$
$VlSID \leftarrow SID$
Generate $Rand_0$
$NewVl_i \leftarrow E_{VlSID.pk_2}(VlSID||Rand_0||cs)$
Generate $Rand_{i.1}$
$S_{i.1} \leftarrow hash(NewPr||NewVl_i||Rand_{i.1})$
Generate $Rand_{i.2}$
Return $NewVl_i$, $S_{i.1}$, and $Rand_{i.2}$

new leader twice. The mechanism for choosing new leaders is performed in a way similar to selecting a new proposer. Algorithm 6 illustrates how to select new leaders anonymously, where each current validation-leader i conducts the following:

1. Randomly selects a new validation-leader's secret Id ($VlSID$) from the current leader's range sg_i in the list $SList$.

2. Generate its own random number ($Rand_0$).

3. Produce an anonymous new leader ($NewVl_i$) by encrypting the $VlSID$, $Rand_0$, and cs with the new selected leader's public key $VlSID.pk_2$ as below:

$$NewVl_i \leftarrow E_{VlSID.pk_2}(VlSID||Rand_0||cs)$$

4. Randomly generate a $Rand_{i.1}$. This random number is similar to the one we utilize in the static mapping (Section 4.1).

5. Partially create secret 1 ($S_{i.1}$) for the new selected proposer $NewPr$ as follows:

$$S_{i.1} \leftarrow hash(NewPr||NewVl_i||Rand_{i.1})$$

   Note that every current leader participates in creating this secret. This is because secret 1 (S_1) used in the static mapping (Section 4.1) consists of all the four leaders' public keys.

6. Randomly generate a $Rand_{i.2}$. This is to make a different $Rand_2$ for each new leader in order to make its S_2 different from the others new leaders' S_2s.

7. Sign and broadcast $NewVl_i$, $S_{i.1}$, and $Rand_{i.2}$.

Algorithm 7: Mapping and Validating the New Proposer and Leaders

Input : $NewPr$, $NewVl_{[1-4]}$, $S_{[1-4].1}$, $Rand_{[1-4].2}$, $CycleSlots$, and cs
Output: ($NewPr$, S_1) and ($NewVl_{[1-4]}$, $S_{[1-4].2}$, $Rand_{[1-4].2}$)

```
if Check cs is in CycleSlots then
    CycleSlots ←CycleSlots - rs
    S_1 ← S_1.1||S_2.1||S_3.1||S_4.1.
    foreach NewVl_i do
        S_i.2 ← hash(S_1||Rand_i.2)
    end
    Return (NewPr, S_1) and (NewVl_[1-4], S_[1-4].2, Rand_[1-4].2)
else
    Return Invalid cycle slot
end
```

Since the current leaders do not know each other and have no means to communicate with each other, selecting the new leaders is performed anonymously and independently. Moreover, the current leaders have no clue of what the new proposer is. Hence, a *current* leader cannot select a *new* leader that can **collude** maliciously with the *new* selected prosper.

5.2.3 New Proposer-Leaders' Mapping and Validating

The current validators are responsible for finalizing the mapping and voting on it, so the other nodes in the network can commit this mapping. The primary purpose of this part of the dynamic mapping is for the validators to form the secret 1 (S_1) for the new proposer and secret 2 ($S_{i.2}$) for each new leader. In addition to that, they need to vote on the mapping the same way they vote on the proposed block. Algorithm 7 shows the task for every validator. Each validator does the following:

1. Validate that cs is an existing valid cycle slot in the $CycleSlots$.
2. Exclude cs form $CycleSlots$ (i.e., $CycleSlots - rs$). This step to avoid forming two maps for the same cycle slot.
3. Construct S_1 as follows:

$$S_1 \leftarrow S_{1.1}||S_{2.1}||S_{3.1}||S_{4.1}.$$

4. For each new selected validation-leader $NewVl_i$, do:

$$S_{i.2} \leftarrow hash(S_1||Rand_{i.2})$$

5. Sign and vote on:
   - The new selected proposer and its secret ($NewPr$ and S_1).
   - Each new selected leader $NewVl_i$, its secret, and its random number ($NewVl_i$, $S_{i.2}$, and $Rand_{i.2}$).

Algorithm 8: Checking if a Node is a Proposer or a Leader

```
   Input  : NewPr, NewVl_[1−4]
 1 Let IsPr ← D_{own.sk_2}(NewPr)
 2 if IsPr.PrSID == own.SID then
 3 |   own.cs ← IsPr.cs
 4 |   own.S_1 ← IsPr.S_1
 5 end
 6 foreach NewVl_i do
 7 |   Let IsVl ← D_{own.sk_2}(NewVl_i)
 8 |   if IsVl.VlSID == own.SID then
 9 |   |   own.cs ← IsVl.cs
10 |   |   own.S_2 ← IsPr.S_2
11 |   |   own.Rand_2 ← IsVl.Rand_2
12 |   |   own.C.add(own.S_2, own.Rand_2, IsVl.cs)
13 |   end
14 end
```

5.2.4 Checking if a Node in the New Mapping

After reaching consensus on the proposed proposer-leaders' mapping, each node in the network verifies if it has been selected as a new proposer or one of the new leaders. A node can check if it is the $NewPr$ by decrypting the $NewPr$ using its second private key (sk_2) as below:

$$D_{sk_2}(NewPr)$$

If the outcome matches its own SID, then this node is the proposer for the slot s in the next cycle c. If it is the proposer for the next cycle slot (cs), this node broadcasts the assigned S_1 when that cycle slot comes. Similarly, each node checks if it is a new leader for that cycle slot by decrypting all the $NewVl$s. Algorithm 8 illustrates this process.

Each time a new mapping is proposed, a cycle slot cs is excluded from the list $CycleSlots$. By the end of the cycle (c), the $CycleSlots$ will be empty. At the beginning of the next cycle, every node in the network populates its own copy of $CycleSlots$. The population mechanism is simple; only increase c by 1, while the slots numbers in the cycle $c+1$ remain the same as in cycle c. We mentioned earlier that the proposing/validation cycles have a number of slots. This number could be any finite number and remain the same for every cycle (we choose it to be equal to the number of nodes (n) in the network).

5.3 Proposers/Leaders' Legitimacy Checking

At the initialization phase of our protocol, and to ensure that a proposer is *legitimate* and has been selected by the blockchain's initiator to propose

a block for a cycle slot cs, the initiator assigns each proposer a verifiable proof we call π'. This proof is a digital signature signed by the initiator using its private key ($in.sk$). The proof π' includes the cycle slot number (cs) and is defined as below:

$$\pi' \leftarrow Sign_{in.sk}(cs)$$

The proposer must submit this proof to the network along with its secret (S_1), so that other nodes can verify π' using the initiator's public key ($in.pk$) prior to involving in the validation and consensus process. This protects against 'malicious nodes' claiming that they are 'proposers' for a certain cycle slot.

Similarly, each validation-leader will be assigned a proof we call π defined as follows:

$$\pi \leftarrow Sign_{in.sk}(cs)$$

Since each cycle slot cs is uniquely identified, each proposer or leader can only be involved in the cycle slot that they have been assigned to. Each other node in the network can easily verify that by decrypting the π' or π using the initiator's public key ($in.pk$) to check the legitimacy of their claims.

To apply the same proposers/leaders legitimacy checking approach after the initialization phase, each *current* proposer that selects and proposes a *new* proposer for a cycle slot (cs) in the next cycle will generate the verifiable proof π' for this new selected proposer. The new π' is a digital signature signed by the *current* proposer's private key ($current.sk_1$). This new proof π' includes the *current* proposer's proof ($current.\pi'$) as a proof of its legitimacy, and the cycle slot (cs) as below:

$$new.\pi' \leftarrow Sign_{current.sk_1}(current.\pi'||cs)$$

The $new.\pi'$ is then attached to the new selected proposer and its secret (i.e., $NewPr$ and S_1). The new selected proposer ($NewPr$) needs to submit $new.\pi'$ along with its secret (S_1) as a proof of its legitimacy to propose for cs.

Likewise for the *new* leaders, each *current* leader is responsible for generating a proof π for the new leader whom it selects. π is a digital signature signed by the current leader's private key ($current.sk_1$), and includes the current leader's proof ($current.\pi$), and the cycle slot (cs) as below:

$$new.\pi \leftarrow Sign_{current.sk_1}(current.\pi||cs)$$

This way, we protect the consensus form malicious nodes claiming that they are a proposer or leaders for a cycle slot. Each node in the system that claims it is involved in a cycle slot has to submit its proof, so that each other node can verify it.

5.4 Protocol Liveness

In the current version of our protocol, we employ only one proposer per *cs*. This because our protocol (using the static proposer-leaders' mapping) was designed to the Block-Supply use case, in which only the node that has the product can propose a new block. Nevertheless, we introduced the dynamic mapping to make our protocol suitable for any blockchain application.

Having one proposer per *cs* could threaten the protocol liveness. This because the selected proposer for a particular slot may fail to propose due regular crashes or maliciousness. To overcome such an issue, the dynamic mapping requires each *current* proposer to select and propose more than one *new* proposer per a cycle slot. The corresponding *current* leaders, in turn, select the *new* leaders in a way that every of the *new* proposer would have four *new* leaders. The number of selected new proposers varies based on the blockchain network's conditions such as the number of the nodes, environment hostility, and how likely crashes occur.

Although such an approach can fix the issue of liveness, it arises a new problem. That is, if more than one proposer proposes a block for the same slot, which one of these proposers should we select as the *primary* proposer. Note that we are only accepting one proposer per a cycle slot to avoid forking. Our protocol overcomes this problem by electing the new proposer with the highest reputation value (R). So, when the chosen proposers broadcast their secrets (S_1s) for a cycle slot *cs*, the nodes decide which one of them is the primary proposer based on R. When the nodes determine the primary proposer, they only wait to hear from its corresponding leaders. Moreover, any other proposer checks if its own R is less than any other R. If so, this proposer aborts the proposing process to avoid unnecessarily work and conflict. However, if a proposer with less R proceeds, its block eventually will be discarded.

5.5 Limitations

The dynamic proposers-leaders mapping achieves the anonymity and randomizes the mapping. A *new* selected proposer ($NewPr$) or leader ($NewVl$) is anonymous and only known to the node that has the second private key (sk_2). This sk_2 corresponds to the public key (pk_2) that was used to encrypt the secret ID (SID) for this $NewPr$ or $NewVl$. Note that the 'correlation' between a node's *public identity* (i.e., the first public key (pk_1)) and the *secret hidden identity* (i.e., second public key (pk_2) and secret ID (SID)) is private information and unknown to the other nodes. In other words, the hidden identities (i.e., pk_2s and SIDs) for all nodes are publicly published in the list $SList$ for the seek of new proposers and leaders selection, but no node in the network knows

which (pk_2, SID) relates to which node except the 'owner node.' In addition to that, no *current* proposer recognizes it leaders nor a leader knows its proposer or its peer leader. Therefore, colluding between these nodes to compose malicious *new* mapping is infeasible. Although the dynamic proposers-leaders mapping is anonymous, it suffers the following two limitations:

1. The *current* proposer may observe the identity (i.e., pk_1) of the *new* proposed proposer when this new proposer proposes its block in cs. Hence, it can correlates the hidden identity (pk_2, SID) to the public identity (pk_1). However, this is unuseful knowledge since the *current* proposer will only uncover this information after the *new* proposer has proposed its block. Additionally, this malicious *current* proposer does not know any of the corresponding *new* leaders since each one has been selected by a different *current* leader.

   However, one possible attack is for the *current* proposer to intently select the same *new* proposer (i.e., selecting the same $pr.SID$), when the malicious *current* proposer has the chance to repropose in future. Then, the malicious *current* proposer could launch a DDoS attack to prevent the new proposer from proposing. Nevertheless, having more than one proposer per a cycle slot (cs) can withstand such an attack. Also, we can enforce the random new proposer selection by applying a game theoretical rewarding mechanism to reward honest, adhering to the protocol nodes and penalize dishonest ones. The same above argument applies to malicious current leaders.

2. A *new* selected proposer ($NewPr$) for a cycle slot (cs) can be also selected as a *new* leader for itself by one of the *current* leaders. This is because there is no any communication between the *current* proposer and leaders. However, only one leader can select the $NewPr$ as a $NewVl$ because each leader has a range (sg) in the list $SList$ to choose from. Our protocol, however, can tolerate this issue since it is based on Tendermint and hence it can tolerate up to one-third of Byzantine fault assuming that the $NewPr$ is a Byzantine malicious node.

6 Experiments and Evaluation

In this section, we evaluate the performance and security of our proposed true decentralized consensus protocol. One of our most important design goals is to balance between performance and security. We chose Tendermint as a reference protocol due to its noteworthy performance, and ability to maintain liveness and safety in the presence of Byzantine nodes.

Our protocol achieves remarkable performance and at the same time maintains a reasonable level of robustness in a fully decentralized and distributed manner. The high performance and scalability are accomplished by **decreasing the number of validators** and **distributing the validation work** among

the blockchain nodes on every block proposal, instead of relying on the same set of validators. The security is achieved by the **anonymous leaders-proposers' mapping** and the **random validators' selection**. We set the *risk* value in our protocol so the employed validators would be *log n* nodes. This is only to show that even if we select a relatively small number of validators, our protocol can still be secured when compared to an optimal secured protocol where all the nodes in the network are selected to be validators. However, as mentioned before in Section 4.2.1, the number of validators in our protocol is dynamic, and it changes based on the proposing node risk likelihood.

6.1 Experiments Setup

In our experiments, we used Omnet++ as our simulation platform. OMNeT++ is a C++-based discrete event simulator for modeling communication networks [42]. It has gained wide-spread popularity in P2P protocols simulations [43], which makes it very suitable for our use case. Besides, OMNET++ proves to have better performance than some well-known simulators such as NS2 and OPNET [44]. Other advantages of OMNET++ are: it is well structured, highly modular, not limited to network protocol simulation, and source code is publicly available [45].

Neverthless, OMNET++ doesn't have a blockchain-based platform, and is not ideal for simulating a real-time application. However, we had to use it due to the lack of blockchain-enabled simulators. In addition, OMNET++ is not limited to network protocols simulation [44] and is known to its popularity in modeling p2p networks which is suitable for our blockchain use case as we have modeled our blockchain network as a p2p network. Creating a large testbed for evaluating an application like blockchain can result in high efforts [46]. Therefore, we utilized OMNeT++ to get a glimpse of how well our proposed protocol is performing before involving in costly, subtle implementations.

We simulated our Block-Supply chain as a peer-to-peer (p2p) network. The network is initiated in the genesis state. The two protocols (Tendermint and ours) where exactly simulated as they were described before (Sections 2.2.3 and 4). The only thing that we had to introduce is how to simulate the physical products and their movement between nodes. In this regard, we treated a product as a '*control*' message that propagates between nodes in the same way that the product might geographically flow in a real supply chain starting from the manufacturer (i.e., node 0) and ending to the last node in the supply chain (i.e., node $n - 1$). This newly introduced *control* message contains the data that are supposed to be on the product's NFC tag. It is assumed that there are fields in the *control* message that are read-only (i.e. *TID* and *counter*). However, without loss of generality, we had to update the tag's *counter* variable in a way that simulates updating it automatically upon readings in the real world.

The *control* message triggers the proposing nodes. In other words, when a node receives a *control* message, it becomes a proposing node, so, it sends its S_1 to all nodes. Similarly, when the product is valid, that is the consensus protocol was executed, and the new block is committed, the proposing node sends this *control* message to the next node in the network and becomes an idle node. This way we model our p2p network as a supply chain where real physical products are treated as *control* messages. As in our Block-Supply chain, where each product is uniquely identified by a PID, each corresponding control message is uniquely identified by the same PID.

It is worth noting that we modeled our blockchain network as a permissioned/private blockchain. This is because our Block-Supply chain application is defined on a set of nodes known in advance as in the real supply chains, where the products flow in a certain path from their manufacturers to the end consumers. Hence, we did not have to introduce the function/mechanism of how new nodes enter the network.

6.2 Products Verification against Counterfeiting

In our simulated blockchain network, when node i receives a *control* message (that represents a product in the real world), the node authenticates the product locally by executing the *local authentication* algorithm (Section 3.2.1). The local authentication involves the following:

1. Given the product data ($PDetails$) on the *control* message, verify its signature ($SignedPDetails$) using the manufacturer's public key ($m.pk$) to detect **modification** on $PDetails$. The manufacturer in our simulated network is the first node (i.e, node 0). Each other node in the network knows the public key of node 0. If pass, proceed; if not, abort.

2. Check if the $PDetails$ on the *control* message is the same as the $PDetails$ on the last block in the node's own copy of the product's blockchain.

3. Check if the tag ID ($ToSignTID$) included in $PDetails$ is equal to the tag ID (TID) on the *control* message. This is to detect **cloning** of $PDetails$. If pass, proceed; if not, abort.

4. Check if the value of the ($Counter$) on the *control* message is equal to the value of $ReadingsOnBlock$ stored on the last block of the product's blockchain. This is to detect **tag reapplication**.

If the local authentication succeeds, the product is authentic, and node i proceeds accordingly. Node i can check for all three types of attacks (modification, cloning, and reapplication) by relying only on its own copy

of the product's blockchain (assuming a valid blockchain). Ensuring that the blockchain is valid is the responsibility of the validators.

After successful *local authentication*, and before sending the *control* message to the next node in the network, node i creates a new block (represented by a message containing all the data on a block) and broadcasts this newly created block to all nodes in the blockchain network to be validated and agreed on by the validators. When a consensus is reached on the proposed block, node i sends the *control* message to the next node in the network simulating shipping a product to the next node in a real supply chain.

6.3 Performance

We have conducted several experiments to examine our protocol performance with different numbers of nodes. The question that we were trying to answer by conducting these experiments is: "*if we validate the same number of products on two blockchain networks, each one of them uses different consensus protocol (i.e., one uses ours and the another uses Tendermint), having the same number of nodes, under the same network conditions, which protocol will outperform the another in term of performance?*"

Our reference protocol was Tendermint employing $n-1$ nodes as validators. We chose $n-1$ validators for Tendermint because the only way that Tendermint can guarantee the true centralization, fairness of validators' selection, and optimal security is by involving $n-1$ nodes in the validation process.

We simulated our Block-Supply chain application to examine and compare the performance of Tendermint and our proposed protocol. In this section, we evaluate two performance metrics:

1. **Latency**: Measured as the time taken to commit one proposed block. Figure 4 shows that our protocol outperforms the optimal secured Tendermint.

2. **Scalability**: Measured as the changes of *latency* when increasing the number of nodes in the network. To better illustrate how scalable each protocol, we simulated five different sizes of blockchain networks using both Tendermint with $n-1$ validators and ours with $log\ n$ validators. We chose networks of 100, 125, 150, 175, 200 nodes respectively. We measured the total time taken to validate a product in its supply chain life cycle (i.e., from the first node to the last node). This way, we were able to observe a clear illustration of how a consensus protocol might affect the validation and consensus time. Figure 5 illustrates the comparison between our $log\ n$ validators protocol and the $n-1$ validators Tendermint.

Despite the overhead that we introduced by the anonymous proposers-leaders' mapping and the random the validators' section, our protocol outperforms Tendermint and shows great scalability. After analyzing the simulation results, we found that two factors influence the protocols' performance:

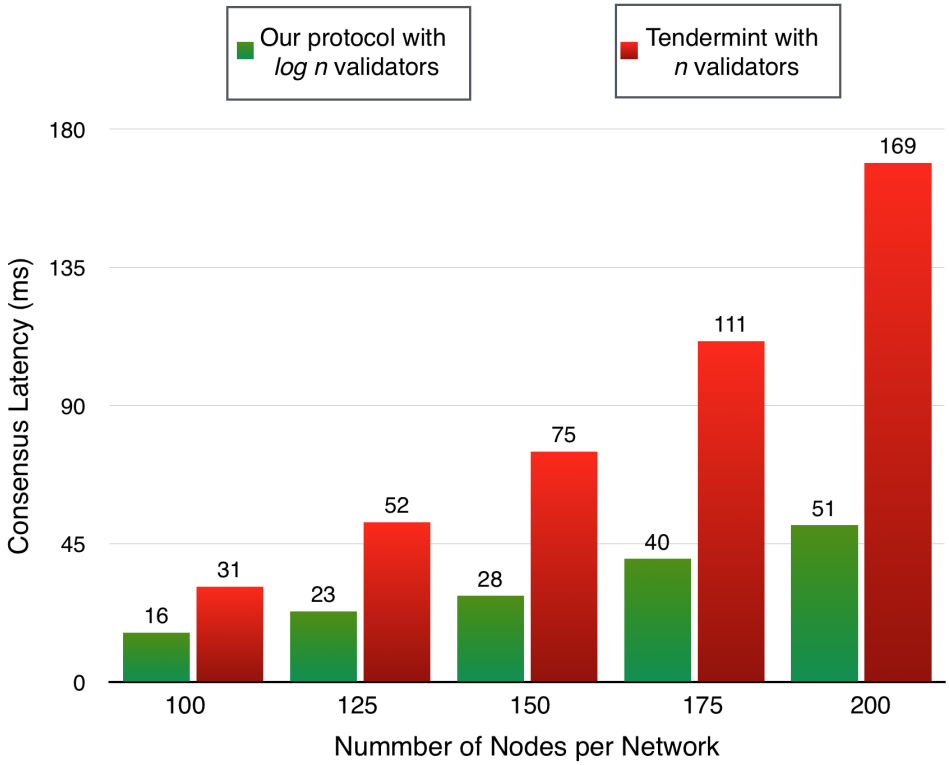

Figure 4: Consensus latency for committing one block.

1. Communication overhead: The number of exchanged messages to reach a consensus was the most dominant factor. In our protocol, this number was reduced due to a small number of communicating validators. This contributes significantly to lower latency compared to Tendermint.

2. The validation work: Another important factor was the computational cost resulted from validating a new block. This cost is proportional to the number of validators performing the validating computation.

6.4 Security

In our protocol, each proposer is blindly and randomly mapped to four leaders. Then, each of the leaders randomly selects a portion of validators without any communication with its peer leaders. This way of selection approach protects against the following attacks:

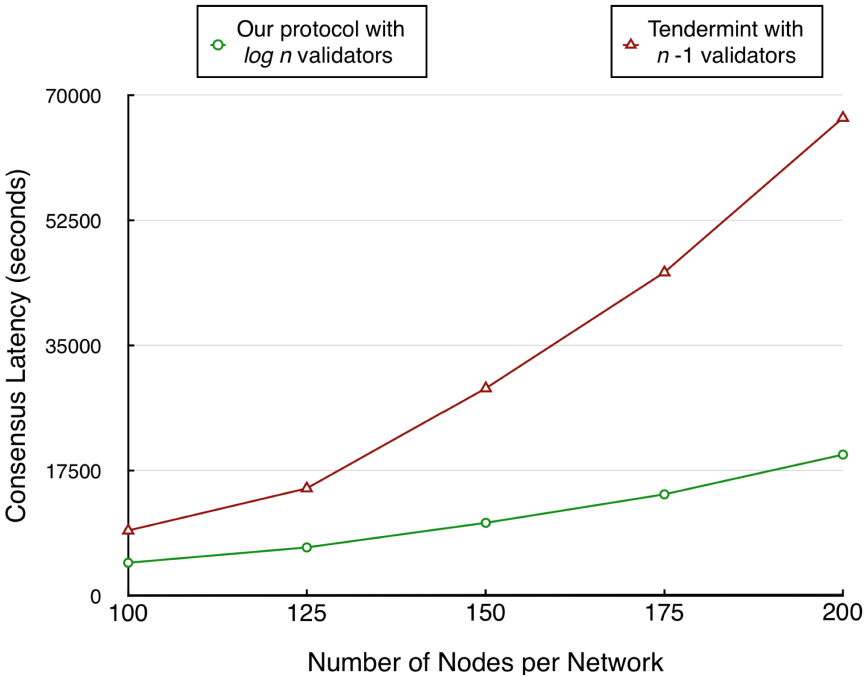

Figure 5: Scalability: our protocol's consensus latency with random $log\ n$ validators increases gradually, while Tendermint's time with $n-1$ validators increases significantly on networks of sizes between 100 - 200 nodes.

6.4.1 DDoS attacks

The DDoS attack is more likely to happen if the set of validators is known in advance. Such an attack can happen to undermine the blockchain and can be launched from inside or outside the network. Validators' replaceability and randomizing their selection can significantly mitigate this attack. This is because of the set of validators changes randomly, and their identities remain anonymous until they participate in the consensus voting. Besides, each step of voting has a different set of voters. Thus, launching a DDoS attack is almost impossible and requires to attack all the nodes in the network to undermine the system. Similarly, attacking the leaders is hard too since leaders are known only after completing their tasks (i.e., broadcasting the m value and instructing their selected validators/pre-voters and pre-commiters). Note that we just aim to protect the validation and consensus process form DDoS attacks.

6.4.2 Eclipse attacks

This attack is presented by Heilman et al. [47] and allows an attacker who controls an adequate number of IP addresses to manage all connections to and from a victim node. As a result, the adversary can utilize the victim nodes

for attacks on block validation and consensus system. As in the DDoS attack, an adversary mounting this attack needs to know the node participating in the validation and consensus process in advance. Introducing variable random validators on each block's proposal makes the adversary's job more difficult.

6.4.3 Bribery Attacks

Bonneau et al. [48] introduced a new attack on blockchains consensus so-called bribery attack. In this attack, a malicious node deliberately pays miners or validators to work on specific blocks and forks. The goal of such an attack is to generate an arbitrary fork which benefits this malicious node. The attack generally works by the adversary offering the miners a bribe to misbehave and deviate from the protocol. This bribe is higher than the fees/reward that the miners/validators obtain if perform correctly and adhere to the protocol. An example of this attack is when a malicious proposer bribes and convinces other leaders or voters to accept and vote for an invalid block. Performing such an attack requires knowing the identities of the targeted nodes. Our protocol anonymizes the interaction between the consensus and validation parties, which significantly mitigates such an attack.

6.5 Leaders denial of service attack [49]

In this attack, an omission fault occurs due to a leader avoiding to select validators as instructed by the protocol. This attack could happen for various reasons, but mostly that the fees associated with a proposed block does not worth working on it [49]. The game theoretical model that we integrate to our protocol mitigates such an attack. The game played between the proposer and it leaders incentivizes the leaders to adhere to the protocol due to the reward the game provides. Additionally, the punishment or the cost would disincentivize malicious leaders to perform this attack.

6.6 Experimental Evaluation

Proving a consensus protocol security experimentally requires examining all the possible strategies for an adversary, which is infeasible [10]. However, for illustration purpose, we choose the *bribery attack*. In this attack, a malicious proposing node tries to corrupt the nodes responsible for executing the consensus (i.e., leaders and validators) by bribing them. In other words, a malicious proposer (i.e., the bribing party) proposes an invalid block, and the corrupted nodes (i.e., the bribed parties) agree and vote on this block. The incentive for such an attack is financial for both the bribing and bribed parties. We model the attacks and evaluate the protocols as follows:

- We randomly select a 0.33% fraction of the nodes as malicious from each network.

The reason for this choice is that Tendermint and our protocol tolerate up to 1/3 of Byzantine nodes, which is in this case malicious. To evaluate a critical aspect such as the security, we need to maximize the risk to the highest limit. In our case, the highest we can do is the 0.33% of the network being malicious. Byzantine Fault Tolerance (BFT) protocols such as Tendermint usually can work in environments where less than one-third of their nodes are faulty or malicious. However, it is possible for a blockchain network to have more than one-third (i.e., 0.33%) malicious nodes, and we believe that our protocol can handle such a case. Hence, our future work includes conducting more experiments where a higher percentage of nodes are malicious.

- We assume, without loss of generality, that these malicious nodes are bribable (corruptible). On the other hand, the remaining fraction of nodes are honest and would not accept a malicious bribe.
- Our security metric is Detection Rate (DR), in which undetected (successful) attack is when a malicious node proposes an invalid block and the other malicious consensus nodes agree on it.
- For each experiment, we select the 0.33% random malicious nodes for each network of the five networks (i.e., the 100, 125, 150, 175, or 200 nodes networks). Then, we evaluated each of the following protocols:
  - Tendermint with $n - 1$ validators.
  - Our protocol without validation-leaders (i.e., fixed validators).
  - Our protocol with validation-leaders, but without anonymous proposer-leaders' mapping.
  - Our protocol with validation-leaders' and anonymous mapping.

  The reason that we select these four types of protocols is that we aim to illustrate how our true decentralized protocol evolves, and to show the importance of applying the anonymity and the validators' replaceability to our protocol.
- In each of the above protocols a successful attack is:
  - Tendermint with $n - 1$ validators: an adversary (i.e., malicious proposer) needs to corrupt/bribe all the nodes in the network, which is infeasible since the security of blockchains is based on the assumption that the majority of the nodes are honest.
  - Fixed-validators: the malicious proposer would try to bribe the fixed set of validators over time. If more than two-thirds of these validators are malicious (i.e., corruptible), the attack succeeds.

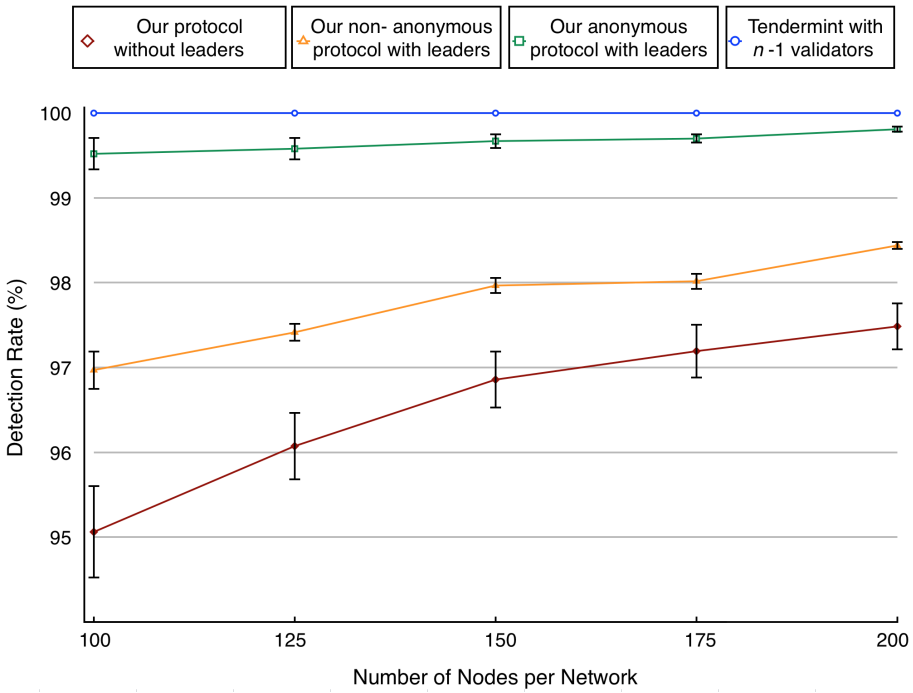

Figure 6: Detection Rate with 95% Confidence Intervals.

- Our protocol leaders *without* anonymous proposer-leaders' mapping: the malicious proposer would try to bribe a *chosen set* of $log\ n$ nodes over time. Then, it bribes its corresponding *known* leaders to select these nodes as validators (note, we are not applying the anonymous mapping).
- Our protocol leaders *with* anonymous proposer-leaders' mapping: the malicious proposer bribes a *chosen set* of $log\ n$ nodes over time. However, in this protocol, the proposer does not know its corresponding leaders in advance, and they are anonymous. Hence, it can only try to blindly select a set of nodes hoping they are its leaders and bribes them. We call this set the 'lottery choice.'

Figure 6 shows the DR with 95% Confidence Intervals (IC). This figure nearly offers a clear vision of how the true decentralization (anonymous mapping) contributes to the consensus protocol security. Our protocol with the anonymous proposers-leaders' mapping achieves high attacks detection rate, ranging from 99.5% when the network size is 100 nodes to 99.8% with 200 nodes network. Note we only examine one attack (bribery), but we argue that true decentralization withstands many other attacks such as DDoS and eclipse.

7 Conclusion

In this paper, we have proposed a new supply chain using blockchain technology. This Block-Supply chain can detect modification, cloning, and tag reapplication attacks in addition to tracking products without a centralized managing server. Furthermore, this paper has introduced a new truly decentralized consensus protocol utilizing anonymous, dynamic mapping and randomness. Our protocol randomly employs a different set of different size of validators on every block' proposal to protect against several real attacks mounted by powerful adversaries. Our simulations show that our new protocol is very scalable for large networks when utilizing a relatively small number of validators. At the same time, it maintains a satisfiable level of security.

This work, however, is in progress and has a few problems. First, OMNET++ is not ideal for simulating a real-time application. The reason that we used OMNeT++ is to test and evaluate only our proposed consensus protocol. Nevertheless, the implementation of our protocol is left for future work. Second, we only chose 0.33% as malicious nodes because when evaluating the security of our protocol. However, it is possible for a blockchain network to have more than one-third (i.e., 0.33%) malicious nodes, and we believe that our protocol can handle such a case. Hence, our future work includes conducting more experiments where a higher percentage of nodes are malicious.

References

[1] Tim K Mackey and Gaurvika Nayyar. A review of existing and emerging digital technologies to combat the global trade in fake medicines. *Expert opinion on drug safety*, 16(5), 2017.

[2] Niels Hackius and Moritz Petersen. Blockchain in logistics and supply chain: trick or treat? In *Proceedings of the Hamburg International Conference of Logistics (HICL)*. epubli, 2017.

[3] Marc Pilkington. 11 blockchain technology: principles and applications. *Research handbook on digital transformations*, 2016.

[4] Naif Alzahrani and Nirupama Bulusu. Securing pharmaceutical and high-value products against tag reapplication attacks using nfc tags. In *Smart Computing (SMARTCOMP), 2016 IEEE International Conference on*. IEEE, 2016.

[5] Ethan Buchman. *Tendermint: Byzantine fault tolerance in the age of blockchains*. PhD thesis, 2016.

[6] Mikko Lehtonen, Thorsten Staake, and Florian Michahelles. From identification to authentication–a review of rfid product authentication techniques.

In *Networked RFID Systems and Lightweight Cryptography*, pages 169–187. Springer, 2008.

[7] Zoltan Nochta, Thorsten Staake, and Elgar Fleisch. Product specific security features based on rfid technology. In *Applications and the Internet Workshops, 2006. SAINT Workshops 2006. International Symposium on*, pages 4–pp. IEEE, 2006.

[8] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OSDI*, volume 99, 1999.

[9] Hyperledger Community. *hyperledger/fabric*, 2018. https://github.com/hyperledger/fabric/tree/v0.6.

[10] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 51–68. ACM, 2017.

[11] Robin Koh, Edmund W Schuster, Indy Chackrabarti, and Attilio Bellman. Securing the pharmaceutical supply chain. *White Paper, Auto-ID Labs, Massachusetts Institute of Technology*, pages 1–19, 2003.

[12] Pedigree Ratified Standard EPCglobal. 1.0., epcglobal (2007).

[13] Lisa A.Daigle. Following pharmaceutical products through the supply chain.

[14] HH Cheung and SH Choi. Implementation issues in rfid-based anti-counterfeiting systems. *Computers in Industry*, 62(7):708–718, 2011.

[15] Muhammad Qasim Saeed, Zeeshan Bilal, and Colin D Walter. An nfc based consumer-level counterfeit detection framework. In *Privacy, Security and Trust (PST), 2013 Eleventh Annual International Conference on*, pages 135–142. IEEE, 2013.

[16] Muhammad Qasim Saeed and Colin D Walter. Off-line nfc tag authentication. In *Internet Technology And Secured Transactions, 2012 International Conference for*, pages 730–735. IEEE, 2012.

[17] Lei Yang, Pai Peng, Fan Dang, Cheng Wang, Xiang-Yang Li, and Yunhao Liu. Anti-counterfeiting via federated rfid tags' fingerprints and geometric relationships. In *Computer Communications (INFOCOM), 2015 IEEE Conference on*, pages 1966–1974. IEEE, 2015.

[18] Feng Tian. An agri-food supply chain traceability system for china based on rfid & blockchain technology. In *Service Systems and Service Management (ICSSSM), 2016 13th International Conference on*. IEEE, 2016.

[19] Saveen A Abeyratne and Radmehr P Monfared. Blockchain ready manufacturing supply chain using distributed ledger. 2016.

[20] Kari Korpela, Jukka Hallikas, and Tomi Dahlberg. Digital supply chain transformation toward blockchain integration. In *proceedings of the 50th Hawaii international conference on system sciences*, 2017.

[21] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[22] Jae Kwon. Tendermint: Consensus without mining. *Retrieved May*, 18, 2014.

[23] Sunny King and Scott Nadal. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. *self-published paper, August*, 19, 2012.

[24] Alexander Lielacher. Welcome to the world of blockchain consensus protocols, 2017.

[25] Andrew Poelstra. On stake and consensus. 2015.

[26] Daniel Larimer. Delegated proof-of-stake (dpos). *Bitshare whitepaper*, 2014.

[27] Guido Schmitz-Krummacher Max Kordek, Oliver Beddows. lisk: Access the power of blockchain.

[28] Ian Grigg. Eos, an introduction. *Whitepaper) iang. org/papers/EOS_An_Introduction. pdf*, 2017.

[29] Fabian Schuh and Daniel Larimer. Bitshares 2.0: General overview, 2017.

[30] KRISTJAN KOŠIČ, ROK ČERNEC, ALEX BARNSLEY, and FRANCOIS-XAVIER THOORENS. Building an open-source blockchain ecosystem with ark. 2018.

[31] Jae Kwon and E Buchman. Cosmos: A network of distributed ledgers, 2016.

[32] David Schwartz, Noah Youngs, Arthur Britto, et al. The ripple protocol consensus algorithm. *Ripple Labs Inc White Paper*, 5, 2014.

[33] Christian Cachin and Marko Vukolić. Blockchains consensus protocols in the wild. *arXiv preprint arXiv:1707.01873*, 2017.

[34] Leslie Lamport et al. Paxos made simple. *ACM Sigact News*, 32(4), 2001.

[35] Diego Ongaro and John K Ousterhout. In search of an understandable consensus algorithm. In *USENIX Annual Technical Conference*, 2014.

[36] Mike Hearn. Corda: A distributed ledger. *Corda Technical White Paper*, 2016.

[37] Will Martino. Kadena: The first scalable, high performance private blockchain, 2016.

[38] Vlad Zamfir. Introducing casper "the friendly ghost". *Ethereum Blog URL: https://blog. ethereum. org/2015/08/01/introducing-casper-friendly-ghost*, 2015.

[39] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151:1–32, 2014.

[40] Yonatan Sompolinsky and Aviv Zohar. Secure high-rate transaction processing in bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 507–527. Springer, 2015.

[41] Pavlos S Efraimidis and Paul G Spirakis. Weighted random sampling with a reservoir. *Information Processing Letters*, 97(5):181–185, 2006.

[42] András Varga and Rudolf Hornig. An overview of the omnet++ simulation environment. In *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, page 60. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

[43] Rupali Bhardwaj, VS Dixit, Anil Kr Upadhyay, and KIET ARSD. An overview on tools for peer to peer network simulation. *simulation*, 1(26):32, 2010.

[44] Xiaodong Xian, Weiren Shi, and He Huang. Comparison of omnet++ and other simulator for wsn simulation. In *Industrial Electronics and Applications, 2008. ICIEA 2008. 3rd IEEE Conference on*, pages 1439–1443. IEEE, 2008.

[45] OMNET. Omnet++.

[46] Christoph P Mayer and Thomas Gamer. Integrating real world applications into omnet++. *Institute of Telematics, University of Karlsruhe, Karlsruhe, Germany, Tech. Rep. TM-2008-2*, 2008.

[47] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. Eclipse attacks on bitcoin's peer-to-peer network. In *USENIX Security Symposium*, pages 129–144, 2015.

[48] Joseph Bonneau, Edward W Felten, Steven Goldfeder, Joshua A Kroll, and Arvind Narayanan. Why buy when you can rent? bribery attacks on bitcoin consensus. 2016.

[49] Stefano De Angelis. Assessing security and performances of consensus algorithms for permissioned blockchains. *arXiv preprint arXiv:1805.03490*, 2018.