Portland State University

PDXScholar

Dissertations and Theses

Dissertations and Theses

7-16-2019

A Secure Anti-Counterfeiting System using Near Field Communication, Public Key Cryptography, Blockchain, and Bayesian Games

Naif Saeed Alzahrani
Portland State University

Follow this and additional works at: https://pdxscholar.library.pdx.edu/open_access_etds

Part of the [Computer Sciences Commons](#)

Let us know how access to this document benefits you.

Recommended Citation

Alzahrani, Naif Saeed, "A Secure Anti-Counterfeiting System using Near Field Communication, Public Key Cryptography, Blockchain, and Bayesian Games" (2019). *Dissertations and Theses*. Paper 5038.
<https://doi.org/10.15760/etd.6914>

This Dissertation is brought to you for free and open access. It has been accepted for inclusion in Dissertations and Theses by an authorized administrator of PDXScholar. Please contact us if we can make this document more accessible: pdxscholar@pdx.edu.

A Secure Anti-Counterfeiting System using Near Field Communication, Public Key
Cryptography, Blockchain, and Bayesian Games

by

Naif Saeed Alzahrani

A dissertation submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy
in
Computer Science

Dissertation Committee:
Nirupama Bulusu, Chair
Wu-chang Feng
Charles Wright
Feng Liu
Vivek Shandas

Portland State University
2019

Abstract

Counterfeit products, especially in the pharmaceutical sector, have plagued the international community for decades [56]. To combat this problem, many anti-counterfeiting approaches have been proposed [43, 79, 88, 99]. They use either Radio Frequency Identification (RFID) or Near Field Communication (NFC) physical tags affixed to the products. Current anti-counterfeiting approaches detect two counterfeiting attacks: (1) modifications to a product’s tag details, such as changing the expiration date; and (2) cloning of a genuine product’s details to reuse on counterfeit products. In addition, these anti-counterfeiting approaches track-and-trace the physical locations of products as the products flow through supply chains.

Existing approaches suffer from two main drawbacks. They cannot detect *tag reapplication attacks*, wherein a counterfeiter removes a legitimate tag from a genuine product and reapplies it to a counterfeit or expired product. Second, most existing approaches typically rely on a *central* server to authenticate products. This is not scalable and creates tremendous processing burden on the server, since significant volumes of products flood through the supply chain’s nodes. In addition, centralized supply chains require substantial data storage to store authentication records for all products. Moreover, as with centralized systems, traditional supply chains inherently have the problem of a single-point of failure.

The thesis of this dissertation is that a robust, scalable, counterfeiting-resistant

supply chain that addresses the above drawbacks and can be simultaneously achieved by (i) using a combination of NFC tags on products and a distributed ledger such as blockchain for reapplication-proof, decentralized, and transparent product authentication (ii) a novel game-theoretical consensus protocol for enforcing true decentralization, and enhancing the protocol’s security and performance.

In this dissertation, we first propose a new Tag Reapplication Detection (TRD) system to detect reapplication attacks using low-cost NFC tags and public key cryptography. To detect reapplication attacks, TRD tracks the number of times a tag has been read in the supply chain using a ‘central’ authentication server. Second, leveraging the blockchain technology, we propose the Block-Supply Chain, a transformation of TRD into a decentralized supply chain. In this chain, each node maintains a blockchain (distributed public ledger) per product. This blockchain comprises chained blocks, where each is an authentication event. The Block-Supply Chain can detect *tag reapplication attacks* and can replace the centralized supply chain design, thus overcoming the *centralization* issues.

One of the fundamental characteristics of blockchain technology is the consensus protocol. Consensus protocols ensure that all nodes in the blockchain network agree on the validity of a block to be included in the public ledger. The first and most popular of the existing consensus protocols is Proof of Work (PoW). However, PoW requires massive computational effort, resulting in high energy and computing resources consumption. Alternatively, Byzantine Fault Tolerance (BFT) protocols, such as Tendermint [9, 47], were adapted in blockchain technology to be efficient and easy to implement. Nevertheless, not all of BFT protocols guarantee *true decentralization*, and they are mostly based on fixed-validators. BFT fixed-validators protocols typically rely on fixed, static validators responsible for validating all newly proposed blocks. This opens the door for adversaries to launch several attacks on these valida-

tors, such as Distributed Denial of Service (DDoS) and Eclipse attacks. In contrast, a truly decentralized protocol ensures that variable sets of anonymous validators execute the blocks' validations. Building on this observation, we propose the TrueBFT, a truly decentralized BFT-based consensus protocol that does not require PoW and randomly employs a different set of validators on each block's proposal. TrueBFT is designed for permissioned blockchains (in such blockchains, the participants who can transact on the blockchain are limited, and each participant is required to have permission to join the system). Our simulations show that TrueBFT offers remarkable performance with a satisfactory level of security compared to the state-of-the-art protocol Tendermint.

Another issue with current consensus protocols, particularly the BFT, is that the majority of them do not take *the number of employed validators* into consideration. The number of validators in a blockchain network influences its security and performance substantially. In response, we integrate a game theoretical model into TrueBFT that analyzes the risk likelihood of each proposer (i.e., the node that creates and proposes the new block). Consequently, each time a new block is proposed, the 'number of validators' becomes proportional to the risk likelihood block's proposer. Additionally, the game model reinforces the honest behavior of the validators by rewarding honest validators and punishing dishonest ones.

Together, TRD, Block-Supply Chain, and the game-theoretical TrueBFT consensus protocol enable robust, scalable, decentralized anti-counterfeiting supply chain that is resistant to tag reapplication attacks, as well as attacks to consensus protocols such as DDoS and Eclipse attacks.

*Dedicated to my loving parents, my supportive wife, and my lovely daughter for
always being with me and supporting me. You are the reason that I am where I am
now*

Acknowledgements

Foremost, I would like to express my sincere gratitude to my advisor, Professor Nirupama Bulusu, for the continuous support of my Ph.D. program and research, for her patience, motivation, enthusiasm, and immense knowledge. Her guidance helped me in all the time of research and writing of this dissertation. I could not have imagined having a better advisor and mentor for my Ph.D. study. Over the years, she has dedicated her support to help me to cross the finish line.

Second, I would like to thank the rest of my dissertation committee: Professor Wu-chang Feng, Professor Charles Wright, Professor Feng Liu, and Professor Vivek Shandas for serving on my dissertation committee. I appreciate all of their valuable feedback.

In addition, I would love to acknowledge the financial support granted to me by my sponsor and employer, Jeddah University. Jeddah University has fully funded my Ph.D. program. This work would not be if not for the grants and support provided by Jeddah University.

I want to thank my friends Hisham Benotman, Huy Tran, and many more for sharing knowledge, joining my user studies and rehearsals as well as for other activities we have had together.

Last but not least, I would like to thank my family: my parents, my brothers, and sisters for supporting me spiritually throughout my life.

Table of Contents

Abstract	i
Dedication	iv
Acknowledgements	v
List of Tables	x
List of Figures	xi
1 Introduction	1
1.1 Motivation	1
1.2 Problems and Challenges	2
1.3 Thesis Statement	5
1.4 Solutions	6
1.5 Contributions	9
1.6 Dissertation Overview	11
2 Tag Reapplication Detection (TRD) System	13
2.1 Introduction	13
2.2 Background	15
2.2.1 RFID Tags	15
2.2.2 NFC Tags	16
2.2.3 Public key cryptography:	17
2.2.4 Digital Signature	17
2.3 Related Work	19
2.3.1 Track-and-Trace Approaches	19

2.3.2	Cryptographic Approaches	21
2.4	Proposed Approach	23
2.4.1	Initialization Phase	25
2.4.2	Verification Phase	27
2.5	Analysis and Evaluation	32
2.5.1	Security Analysis	33
2.5.2	Economic Analysis	35
2.6	Discussion and Conclusion	36
3	Block-Supply Chain	38
3.1	Introduction	38
3.2	Background	40
3.2.1	Blockchain Technology	40
3.2.2	Traditional Centralized Supply Chains	42
3.2.3	Decentralized Supply Chains	43
3.3	Related Work	44
3.4	Proposed Block-Supply Chain	47
3.4.1	Initialization Phase	47
3.4.2	Verification Phase	50
3.5	Discussion and Conclusion	54
4	True Decentralized Consensus Protocol	55
4.1	Introduction	55
4.2	Background	60
4.2.1	Blockchain Consensus Protocols	60
4.2.2	Broadcast and Consensus	61
4.2.3	Byzantine Fault Tolerance (BFT) in Distributed Systems	62
4.3	Related Work	63
4.3.1	Proof of Work (PoW)	63
4.3.2	Proof of Stake (PoS)	64
4.3.3	Byzantine Fault Tolerance (BFT)	66
4.3.4	Non Byzantine Fault Tolerance	72
4.4	The Proposed TrueBFT Consensus Protocol	74
4.4.1	Initialization Phase	75

4.4.2	Verification (Validation) Phase	79
4.5	Experiments and Evaluation	88
4.5.1	Experiments Setup	89
4.5.2	Performance	90
4.5.3	Security	93
4.6	Limitations and Concluding Remarks	100
5	Game Theoretical Consensus Protocol	101
5.1	Introduction	101
5.2	Background	103
5.2.1	Definition	104
5.2.2	Classification of Game Theories	105
5.2.3	Bayesian Game	106
5.2.4	Nash Equilibrium (NE)	106
5.2.5	Mixed Strategies	107
5.3	Related Work	107
5.4	The Proposed Two-Stage Bayesian Game Model	110
5.4.1	First-Stage Game	110
5.4.2	Second-Stage Game	118
5.5	Security Analysis	121
5.5.1	Malicious or Lazy Validation-Leading	122
5.5.2	Block Withholding Attack [78]	122
5.5.3	Denial of Service Attack [18]	122
5.5.4	Malicious Block Proposing	123
5.6	Conclusion	123
6	Dynamic Proposers-Leaders Mapping	125
6.1	Introduction	125
6.2	Dynamic Proposers-Leaders Mapping	126
6.2.1	Initialization Phase	126
6.2.2	Dynamic Mapping During the Verification Phase	128
6.2.3	Proposers/Leaders' Legitimacy Checking	135
6.2.4	Liveness Issues	137
6.2.5	Limitations	138

6.3	Discussion on Safety and Liveness	139
6.3.1	Safety	140
6.3.2	Liveness	146
6.4	Conclusion	149
7	CONCLUSION	151
7.1	Research Contributions	151
7.2	Future Directions	154
	Bibliography	156

List of Tables

2.1	The notation of the online authentication protocol.....	29
2.2	Comparison of TRD to existing product authentication approaches. . .	36
5.1	The first-stage game notation.	112
5.2	Strategic form of the first-stage Bayesian game (player x is malicious) .	112
5.3	Strategic form of the first-stage Bayesian game (player x is regular)....	112
5.4	Strategic form of the second-stage Bayesian game (player x is malicious)	119
5.5	Strategic form of the second-stage Bayesian game (Player x is regular) .	119
7.1	Contribution summary.....	154

List of Figures

2.1	An NFC tag, designed by NXP, diameter 21mm. Source: [68].	17
2.2	Public key digital signature process. The sender sends the plain message and its signature to the receiver. The receiver knows the sender's public key; given the message, its signature, and the public key, the receiver can determine if the signature is valid.	18
2.3	TRD architecture. Three parties are involved: (1) the manufacturer and its authentication server, (2) the supply chain nodes, and (3) the final consumers. There are two product authentication protocols: (1) online between the nodes and the authentication server, and (2) offline by a consumer's phone when the product is on the market.	24
2.4	Product's manufacturer initialization. The manufacturer digitally signs the product's details (<i>PDetails</i>), and the number of the number of times the tag is read (<i>ReadingsOnTag</i>), and writes them on the tag. The read-only <i>Counter</i> value is equal to the value of <i>ReadingsOnTag</i>	26
2.5	Online authentication protocol. In all steps, if not pass or no received <i>Ack</i> , abort; the product is counterfeit. This protocol first authenticates the product, and secondly updates the readings value. The Authenticate process ensures that the values of <i>ReadingsOnDB</i> , <i>ReadingsOnTag</i> , and (<i>Counter</i> - 1) are equal. After successful authentication, the Update process updates the values of <i>ReadingsOnDB</i> , and <i>ReadingsOnTag</i> to hold the same value of <i>Counter</i> after the authentication. The two keys (<i>pk</i> , <i>sk</i>) utilized in the protocol are for the manufacturer.	30
2.6	Offline customer authentication protocol. In all steps, if not pass, abort; the product is counterfeit. The public key (<i>pk</i>) utilized in the protocol is for the manufacturer.	31
3.1	Traditional supply chain authentication. As products ship from node to node in the supply chain, each node authenticates the products by contacting the authentication server.	42
3.2	A general overview of the decentralized products' authentication architecture. It is a blockchain where the product's authentication records are grouped into a public ledger (i.e., blockchain) seen by every node in the supply chain. Each block includes an authentication record (<i>Record</i>), in addition to the hash (<i>H</i>) of the previous block.	43

3.3	Blocks' structure and chaining them. Two blocks are chained by including the hash of the previous block in the new block.	49
4.1	An example of a Byzantine fault problem. This is a network of three nodes, in which nodes A and B are honest (regular) nodes, and node C is malicious (Byzantine). Source : [62].	63
4.2	Normal case operation of PBFT. Source: [12].	67
4.3	Our consensus mechanism. The first step is "Propose." After this step, validators only advance after hearing from more than two-thirds of M other nodes. By the end of the consensus, either the block is committed and the blockchain height increases, or the proposed block is rejected and the protocol aborts.	85
4.4	Consensus latency for committing one block.	91
4.5	Scalability: TrueBFT's consensus latency with $\log n$ validators increases gradually, while Tendermint's time with $n - 1$ validators increases significantly on networks of sizes between 100 - 200 nodes.	92
4.6	Event of Interest, where A is the set of n elements, B is the subset of $\log n$ elements, and C is the subset of $\frac{\log n}{3}$ elements.	95
4.7	Detection Rate (DR) with 95% Confidence Intervals.	99
5.1	Extensive form of the first-stage Bayesian game.	113
5.2	Extensive form of the second-stage Bayesian game.	119

1 Introduction

1.1 Motivation

The World Health Organization (WHO) estimates that globally, 10% of medicines are counterfeit, rising to 30% in developing countries [13]. Moreover, the Counterfeiting Intelligence Bureau (CIB) claims that counterfeit goods comprise up to 7% of world trade [84]. In 2011, MarkMonitor Inc. reported that counterfeit sales cost the legitimate market \$135 billion in online shopping around the world [22]. In a recent review published in 2017 [56], the authors mentioned that "studies have estimated that as high as 40–50% of antimalarials may be counterfeit" in areas like Southeast Asia and sub-Saharan Africa. Furthermore, in 2018, the Pharmaceutical Security Institute (PSI) reported that counterfeit incidents increased by 60% over the past five years and the majority of these incidents occurred in North America [36].

The problem of counterfeit products is of significant concern. In particular, counterfeit or expired pharmaceutical products can cause severe illness or even death. Additionally, counterfeit high-value products can result in serious economic ramifications for both businesses and customers. In response to this situation, several product anti-counterfeiting approaches have been proposed in recent years. The problems with these existing approaches are: (1) they cannot detect tag reapplication attacks, and (2) most of the existing works are centralized and rely on a trusted server to coordinate and manage product authentication.

Most traditional centralized supply chains utilize tags such as Radio Frequency Identification (RFID) or Near Field Communication (NFC) and an authentication server to detect counterfeiting attacks. Nevertheless, this typical architecture of centralized supply chains still experiences single-point processing, storage, and failure. To overcome such issues, blockchain technology stands out as a potential framework to establish a modernized, decentralized, trustworthy, accountable, transparent, and secure supply chain against counterfeiting attacks [56]. Despite the potential of blockchain technology to elevate security, work and storage distribution, as well as the transparency of supply chains, there have been only a few projects that examine the integration of anti-counterfeiting supply chains with blockchain technology [56]. This has motivated us to exploit this promising technology to combat counterfeit goods to ensure transparency, information security, and quality assurance.

1.2 Problems and Challenges

In this dissertation, we first address the problems of current anti-counterfeiting supply chains. Then, we tackle some of the drawbacks of the existing blockchain consensus protocols (in particular, the BFT protocols).

Lehtonen et al. [51] define three kinds of product counterfeiting attacks: (1) the modification of a product’s details on a tag, such as changing the expiration date, (2) the cloning of a genuine product’s details to reuse on counterfeit products, and (3) the removal of a legitimate tag from a genuine product and its reapplication to a counterfeit product. Although several approaches have been proposed in the past few years, none currently detects all of these attacks. In other words, the *tag reapplication attack* remains a threat in many product authentication systems [51], and requires manual, complex verification to detect it [69], as will be discussed in Chapter 2.

The second problem pertains to the limitations of the *current centralized anti-*

counterfeiting supply chains. Most existing anti-counterfeiting supply chains rely on central servers to authenticate products. Despite their potential in mitigating counterfeit products, their centralization dependency imposes several limitations. First, they impose a tremendous processing burden on the server, since significant volumes of products flood through supply chain nodes. Second, substantial data storage is required to store authentication records for all products. Third, as with centralized systems, traditional supply chains inherently suffer from 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. One potential solution to address the limitations of centralization is to use distributed ledger technology (i.e., blockchain) to store products records. However, applying blockchain technology is non-trivial. Using blockchain technology enables supply chain nodes to authenticate products and maintain a public ledger per product that consists of valid, chained blocks (each of which represents an authentication record) without central servers.

Although the use of blockchain technology in product supply chains potentially eliminates the need for centralized authentication servers, the nodes in the supply chain need a mechanism to authenticate products and agree on their authenticity efficiently and securely. To achieve this, blockchain technology requires robust, yet scalable consensus protocols. Consensus protocols ensure that all nodes in a blockchain network agree on the validity of a block to be included in a public ledger. Consensus protocols also guarantee that all nodes have the same order of blocks in their copies of public ledgers. This is of significance because blockchains are trustless, distributed nodes that need a way to synchronize their copies of stored data. The nodes responsible for validating the blocks and executing consensus protocols are the *validators*.

There is a considerable number of existing consensus protocols. The first and most popular of them is Proof of Work (PoW) (Bitcoin [63]). However, PoW re-

quires massive computational effort, which results in high energy and computing resources consumption. Alternatively, Byzantine Fault Tolerance (BFT) protocols were adapted in blockchain technology to be efficient and easy to implement (e.g., Tendermint [9,47], Hyperledger Fabric [34], and Ripple [83]). Nevertheless, not all of BFT protocols guarantee *true decentralization*, and they are mostly based on fixed-validators. BFT fixed-validators protocols typically rely on fixed, static validators responsible for validating all newly proposed blocks, which exposes these validators to several attacks such as DDoS, Bribery, and Eclipse (these attacks will be discussed in Chapter 4). In contrast, a truly decentralized protocol ensures that the blocks' validations are executed by variable sets of anonymous validators, which results in greater security, since the identities of validators are anonymous. In this dissertation, we mainly focus on the BFT consensus protocols. We aim to exploit their efficiency, taking into consideration the security and efficiency trade-off.

Another issue with current consensus protocols, particularly the BFT, is that the majority of them do not take *the number of employed validators* into consideration. The number of validators in a blockchain network influences its security and performance substantially, especially in a fully decentralized blockchain with no special nodes. The larger the number of validators, the more secure the system, but the poorer the performance. The optimal number of validators in such blockchains that achieves optimal security would be all nodes in the network except the proposing node (i.e., the node that creates and proposes the new block). However, this choice results in: (1) substantial validating work as of $O(n)$ for each validation event (n is the number of nodes in the blockchain network), (2) high communication overhead to reach consensus with $O(n^2)$ in protocols like Practical Byzantine Fault Tolerance (PBFT) [12] and Tendermint [9], and (3) large block size due to including each validator's signature in the block as evidence of its validity. An alternative choice to the

$n - 1$ validators is to rely on a fixed, static number of validators (e.g., Tendermint and Hyperledger Fabric [34]). This choice, however leads to: (1) validators' vulnerability to attacks such as DDoS since the identities of the fixed validators are known, and (2) partial centralization.

The last problem that we address in this dissertation is the *validators' diversion from honest behavior*. For example, a dishonest validator might perform a block withholding attack [78], in which this validator either does not participate in the validation process or does not reveal the result of the verification in favor of a malicious proposing node. This attack can result in undermining the consensus process. Thus, a reward and punishment mechanism is needed to reinforce good behavior. Additionally, the *always-validation* (i.e., validators always validate even if the risk likelihood of a block's proposer is low) might be unnecessary computational work particularly found in blockchains with low hostility. Hence, the validators should validate with some probability proportional to the proposers' risk likelihoods, which would enhance protocol efficiency.

1.3 Thesis Statement

The thesis of this dissertation is that a robust, scalable, counterfeiting-resistant supply chain that addresses the above problems and can be simultaneously achieved by (i) using a combination of NFC tags on products and a distributed ledger such as blockchain for reapplication-proof, decentralized, and transparent product authentication (ii) a novel game-theoretical consensus protocol for enforcing true decentralization, and enhancing the protocol's security and performance.

1.4 Solutions

In this section, we briefly introduce the four proposed solutions that build on our thesis. To detect *tag reapplication attacks*, we first propose a new Tag Reapplication Detection (TRD) authentication system that uses low-cost NFC tags and public key cryptography. In this system, each product instance is affixed with an NFC tag containing the product’s details such as a serial number, a name, and an expiration date. Each NFC tag has a read-only counter that increases each time the tag is read and a read-only, unique tag ID. TRD relies on a centralized product authentication server. Upon receiving a product, each node in the supply chain contacts the authentication server to verify its authenticity. The authentication server has a database in which each product’s details are stored, including how many times the tag has been read in the supply chain (authenticated using an online authentication protocol). This approach enables TRD to detect reapplication attacks. When tagged products are in the market, TRD allows customers to use NFC-enabled cell phones to authenticate products using an offline authentication protocol. TRD also can detect (1) *modification attacks* by signing the details of the products on their tags, and (2) *cloning attacks* by including the value of the read-only tag ID in the signed details. We will discuss the proposed system in detail in Chapter 2.

Although TRD can detect all three counterfeiting attacks (i.e., modification, cloning, and tag reapplication), its architecture inherits the *centralization limitations* discussed in Section 1.2. To overcome this, we propose the Block-Supply Chain, a decentralized supply chain that exploits blockchain technology. In this chain, each node maintains a blockchain (i.e., public ledger) for each product. This blockchain comprises chained blocks, each of which represents 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 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. The Block-Supply Chain is a permissioned blockchain that can track-and-trace products and their tags' reads without a centralized server. Moreover, it detects the three counterfeiting attacks (modification, cloning, and tag reapplication) by involving the supply chain nodes transparently. In the Block-Supply Chain, the validators play the role of the centralized authentication server in TRD, and each node in the network stores the authentication records of a product as a blockchain.

The Block-Supply Chain eliminates the need for a centralized party to orchestrate the authentication process. Instead, a proposing node, which currently has the product, proposes a block of the current product's details to all nodes in the network. Then, the validators validate this block to ensure the product's authenticity and communicate their decisions to the other nodes. Reaching an agreement on these decisions (votes) requires a consensus protocol. As previously discussed in Section 1.2, the current consensus protocols do not offer *true decentralization*.

In response, we propose the TrueBFT, a truly decentralized consensus protocol that does not require PoW and randomly employs a different set of validators on each block's proposal. Our TrueBFT protocol is a BFT-based protocol and is designed to work only with permissioned blockchains. Additionally, the proposed TrueBFT leverages the consensus mechanism offered by the state-of-the-art consensus protocol Tendermint [9, 47]. Nevertheless, the validators' set in Tendermint can be known in advance, which makes them exposed to attacks such as DDoS. In TrueBFT, the set of validators is replaceable and selected randomly from the set of "all nodes" each time a new block is proposed. As a result, TrueBFT achieves the true decentralization and withstands powerful adversaries.

TrueBFT employs three types of nodes: (1) the proposer which creates and proposes the new block, (2) the leader which determines the number of validators and randomly selects them, and (3) the validator which is responsible for validating the newly proposed block. The key idea of our solution is to anonymously map each proposer in the network to some leaders in a way that none of them knows its corresponding peer. However, these nodes communicate with each other via secrets assigned to them at the genesis state by the blockchain initiators. When the proposer proposes a new block, its leaders select the validators at random. This way, the identity of each node (i.e., leader/validator) remains unknown and only revealed after it accomplishes its job.

To overcome the limitations of having a *fixed number of selected validators*, the proposed TrueBFT decides the 'number of validators' according to a hostility risk factor called *risk*. In our premier version of the protocol and for illustration purpose, we set *risk* so that the protocol *randomly* employs $\log n$ validators, rather than the optimal choice of $n - 1$ validators (excluding the proposing node). The reasoning that has led to this specific value (i.e., $\log n$) is to reduce the validation work from $O(n)$ to $O(\log n)$, and the communication overhead from $O(n^2)$ to $O((\log n).n)$. This reduction results in a considerable performance enhancement and reduction of the communication and the storage overhead. Our simulations show that the proposed TrueBFT offers remarkable performance with a satisfactory level of security compared to Tendermint. Nevertheless, we improved TrueBFT by making the number of validators dynamic and variable based on the hostility of the network, as will be discussed shortly.

Despite the security provided by true decentralization, TrueBFT, like most existing protocols, does not guarantee that the *validators are always honest and do not deviate from the protocol*. To overcome such a vulnerability, we integrate a

game theoretical model into TrueBFT to reward honest validators and punish dishonest validators that do not adhere to the protocol. Additionally, the game-theoretical-based TrueBFT eliminates the *always-validation*. Alternatively, it utilizes the theoretical game outcomes to enable the validators to validate with some probability proportional to the proposers' risk likelihoods (i.e., *risk*). Furthermore, TrueBFT exploits the proposers' *risks* to make the 'number of the selected validators' dynamic and variable. In other words, instead of selecting a fixed number of validators, TrueBFT selects a different number of validators (on every block's proposal) proportional to the block's proposer *risk*.

It is worth mentioning that TrueBFT protocol is designed to work with permissioned blockchains. In addition, TrueBFT is highly dependent on the consensus mechanism used in Tendermint; that is, how the validators communicate with each other to reach a consensus on a proposed block. TrueBFT only addresses the issue of the validators' selection in terms of *validators' replacement* and *validators' number* to enhance security with respect to efficiency. Moreover, in this dissertation, we compare our proposed TrueBFT protocol to the BFT protocols. We mainly focus on their issues due to relevance, and offer a light touch on other consensus mechanisms.

1.5 Contributions

The contribution of this dissertation is fourfold:

1. **TRD:** We propose a new authentication scheme that detects *tag reapplication attacks* and has the following features:
 - It detects tag reapplication attacks automatically, without the need for manual verification.
 - It uses low-cost NFC tags that do not have special features, such as pro-

processors (to perform cryptographic operations) or inaccessible memory.

- It involves the end customers in the authentication process using their NFC-enabled cell phones and offline authentication protocol.

2. **Block-Supply Chain:** We propose a decentralized supply chain that exploits the blockchain technology, thus overcoming the *centralization limitations* previously mentioned in Section 1.2. The Block-Supply Chain is a permissioned blockchain that can track-and-trace products without a centralized server. Moreover, it detects the three counterfeiting attacks (modification, cloning, and tag reapplication) by involving the supply chain nodes transparently.

3. **TrueBFT** (True decentralized consensus protocol): We propose a BFT-based consensus protocol (TrueBFT) that achieves true decentralization by randomly replacing the *set of validators* on every block's proposal. Also, the proposed protocol addresses the problem of *how to select the number of validators*. Our solution is to select the number of validators based on a hostility factor called *risk*. TrueBFT has the following advantages:

- When we set *risk* to employ $\log n$ validators, TrueBFT is very efficient and scalable. In particular, it achieves a considerable improvement in performance when compared to the optimal secure Tendermint with $n - 1$ validators.
- With $\log n$ validators, TrueBFT offers a satisfactory level of security. For example, it was able to detect the *Bribery attack* with a detection rate of 99.8% in a blockchain network with 200 nodes, where 33% of them were malicious.

4. **Game-theoretical-based TrueBFT:** We integrate a game theoretical approach into TrueBFT to a) analyze the risk likelihood of the proposing nodes in order to decide the *number of validators*, and b) mitigate *attacks presented by dishonest nodes*. The game theoretical approach exploited by TrueBFT has the following benefits:

- It reinforces the honest behavior of the consensus participants by rewarding honest ones and penalizing dishonest ones.
- The *number of the selected validators* is dynamic and variable. Hence, instead of selecting a fixed number of validators, TrueBFT utilizes the outcomes of the proposed theoretical game to select a *different number* of validators (on every block's proposal) proportional to the risk likelihood of the proposing node (i.e., *risk*).
- It eliminates the *always-validation* mode. Instead, the validators validate with probability proportional to the proposing node *risk*.

To the best of our knowledge, this is the first work to propose: (1) a robust, scalable, decentralized anti-counterfeiting supply chain that is resistant to tag reapplication attacks, as well as (2) a BFT truly decentralized consensus protocol that determines the number of validators based on the risks of the propping nodes, and mitigates attacks like DDoS, Bribery, and Eclipse attacks.

1.6 Dissertation Overview

Chapter 2 covers the problem of *tag reapplication* attacks and proposes the TRD system. Chapter 3 discusses the *centralization* problem and introduces the Block-Supply Chain. Chapter 4 presents the TrueBFT and addresses the *validators' number* problem. Chapter 5 introduces the game theoretical model to our TrueBFT consensus

protocol and discusses how this approach overcomes the problems of *dishonest validators*. Chapter 6 introduces a new *proposers-to-leaders mapping mechanism* to make TrueBFT suitable for blockchain applications other than the Block-Supply Chain. In Chapter 7, we summarize the dissertation and describe directions for future work.

The content of the dissertation is based on four publications. Our first contribution (i.e., TRD) was published in [3]. Our second contribution (i.e., Block-Supply Chain) was published in [4]. Our third contribution (i.e., TrueBFT) was published in [5]. Our fourth contribution (i.e., game-theoretical model) was published in [2].

2 Tag Reapplication Detection (TRD) System

In this chapter, we present our first contribution (Section 1.5). This chapter describes in detail the problem of tag reapplication attacks and our proposed TRD that overcomes this problem.

2.1 Introduction

Product counterfeiting has increasingly become a threat to the world economy [56]. In response to this threat, the people in academia and industries put considerable effort into combat counterfeiting. However, the battle against counterfeiting remains a significant challenge because products flow from their manufacturers to end users through many nodes. Not all of these nodes are trustworthy. This has created and encouraged the growth of counterfeit goods [65].

The current digital anti-counterfeiting systems utilize RFID and NFC tags to track-and-trace products and ensure their legitimacy. In such systems, each product is given a tag containing details such as serial number, name, and expiration date. An authenticating entity (e.g., a supply chain node, or customer) reads these details and checks if the product is genuine via an authentication mechanism. Counterfeiters, however, can perform three kinds of attack on the tagged products [51]. First, they can modify the product's details on the product's tag (modification attack) such as changing the expiration date. Second, counterfeiters can clone a genuine product's details to reuse on counterfeit products' tags (cloning attacks). The third attack is

tag reapplication, in which a counterfeiter removes a legitimate tag from a genuine product and reapplies it to a counterfeit product.

The problem of tag reapplication has been addressed by many researchers. Nochta et al. [69] point out that the link between a tag and an object is an adhesive bonding. As a result, “a tag can be removed from an original article and attached to another object, thereby compromising the security system” [69]. Staake et al. [87] recognize the need to consider tag reapplication attacks, stating that “the consideration of this attack is of importance especially when protecting high-value goods”. Potdar et al. [76] address tag reapplication attacks on the pharmaceutical industry, giving an example of replacing a tag from a cheaper pharmaceutical product with a tag from an expensive one.

Whilst tag reapplication does occur in developed countries, it is more prevalent in developing countries [13]. The motivation for such counterfeiting attacks is profit, especially when products are expensive. An example of tag reapplication is replacing the tag of an unconsumed but expired product, or a counterfeit product, with the tag of a consumed but unexpired product. This type of attack can occur anywhere in the supply chain, but it is more likely to happen when products are on the market. This is why we aim to involve end consumers in products authentication.

Tag reapplication attack is hard to detect because it mostly requires manual inspection [51]. Also, the link between a tag and a product, as mentioned, is only an adhesive bonding that can be easily removed and reapplied or used [69]. To the best of our knowledge, none of the existing anti-counterfeiting approaches have digital means to detect tag reapplication. Therefore, we propose the Tag Reapplication Detection (TRD) approach that uses NFC tags and public key cryptography. The use of NFC tags makes TRD user-friendly, since a large number of existing cell phones are NFC-enabled [79]; therefore, end consumers can use their phones to authenticate

products.

TRD has two authentication protocols. The first is an online protocol within the supply chain, between the supply chain nodes (e.g., distributor, warehouse, retailer) and an authentication server. This protocol requires each node to contact the authentication server to authenticate a tagged product. The second is an offline protocol between a customer’s NFC-enabled cell phone and the tag on the product. The offline protocol does not require the customer to contact the authentication server to authenticate the product.

TRD tracks the number of times a tag has been read to determine whether the tag has been reapplied. Tracking the number of times a tag has been read indicates that the product arrived at the final destination through certified nodes, and was not exposed to a tag reapplication attack between these nodes. At the end of the supply chain (i.e., the final node), the tag is set to be verified exactly once; if an attempt is made to verify it a second time, the customer is warned. In addition to reapplication detection, TRD detects modification and cloning attacks, using a digital signature. Our security analysis of TRD shows that it can detect tag modification, cloning, and reapplication attacks.

2.2 Background

In this section, we provide a brief background on the concepts used in our work to better understand it.

2.2.1 RFID Tags

Radio-Frequency Identification (RFID) tags are used in most identification applications to automatically identify and track tags affixed to objects. Each tag contains information about the object and requires a specialized reader to read this informa-

tion. RFID tags are either passive or active. Passive tags do not have onboard power supplies, and they draw power from the reader. However, the active tags are powered by a battery and automatically broadcast their signal [43].

2.2.2 NFC Tags

NFC (Near Field Communication) is a wireless technology that allows two NFC-enabled devices to communicate data, such as text or numbers. NFC technology is compatible with contactless smart cards (tags) [79]. Similar to RFID tags, NFC tags can be affixed to objects and store information about these objects. However, NFC tags do not require specialized readers to communicate stored information. As an alternative, an NFC tag can be read by a cell phone, which makes this technology user-friendly. The use of NFC technology is significantly increasing. According to a research study conducted by Information Handling Services (IHS) Technology Inc., “NFC was integrated into just 18.2 percent of the 1.5 billion cell phones shipped worldwide in 2013. In 2018, NFC penetration is rising to 64 percent” [90]. NFC tags are small in size (as shown in Figure 2.1), passive (i.e., powered by their readers), and are of four types [66]:

- **Types 1 and 2:** These types of tags are very similar. Memory size can be between 48 bytes and 2 Kbytes. Communication speed is 106 kbit/sec. These two types are cost-effective and ideal for many NFC applications.
- **Type 3:** Memory size can be up to 1 Mbyte. Communication speed with the tag is 212 kbit/sec. The unit price of this type of tag is high.
- **Type 4:** Memory size can be up to 32 Kbytes. Communication speed can be between 106 kbit/sec and 424 kbit/sec. The unit price of this type of tag is high because they contain cryptographic processors.

Figure 2.1: An NFC tag, designed by NXP, diameter 21mm. Source: [68].

2.2.3 Public key cryptography:

This cryptographic system uses two keys (private and public). The private key is known only to the owner and cannot be published to other parties in the system. The public key is published to all parties. To perform an encryption using public key cryptography, a sender encrypts a message using the receiver's public key, resulting in a cipher (encrypted) text. The receiver decrypts the cipher text, using its private key to obtain the original message [61].

2.2.4 Digital Signature

In this scheme, a message is signed by a sender, and verified by a receiver. In public key cryptography digital signatures, the message is signed by the sender's private key, and verified by the receiver using the sender's public key. A valid digital signature gives the receiver reason to believe that the message was created by a known sender (authenticity of the message), and the message was not modified (integrity of

Figure 2.2: Public key digital signature process. The sender sends the plain message and its signature to the receiver. The receiver knows the sender's public key; given the message, its signature, and the public key, the receiver can determine if the signature is valid.

the message) [28]. As shown in Figure 2.2, to digitally sign a message, the sender performs the following:

- Hashes the plain message using a hash function (takes a message as input and produces a unique hash as output in which two messages cannot have the same hash), resulting in *hash 1*.
- Encrypts *hash 1* using the sender's private key, resulting in a signature.
- Sends the plain message and its signature to the receiver.

The receiver knows the sender's public key. To verify the signature of the message, the receiver performs the following:

- Decrypts the signature using the sender's public key, resulting in *hash 1*.
- Hashes the plain message using the same hash function used by the sender, resulting in *hash 2*.

- Compares *hash 1* to *hash 2*. If they are the same, the signature is valid.

2.3 Related Work

In this section, we will examine some related existing works. The relevant literature 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.

2.3.1 Track-and-Trace Approaches

This kind of approach uses RFID tags to track the physical locations of a product, which are then stored in a centralized database. Koh et al. [43] 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. Similarly, TRD tracks the number of times a tag is read and updates this number in a manufacturer's database. However, 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 [21]. 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 [17]. 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.

2.3.2 Cryptographic Approaches

Cryptographic approaches use public or private key cryptography to authenticate products. These approaches usually use a challenge-response protocol. For example, Saeed et al. [79, 80] propose a cryptographic challenge-response protocol in which a tag's reader sends a challenge to the tag. The tag then signs this challenge with a private key stored on the tag and sends the signature to the reader. The reader verifies the signature using the corresponding public key, and can thus determine whether the tag is legitimate.

Cryptographic approaches are of two kinds. The first are **online approaches**, which use a centralized database to authenticate products. Staaake et al. [88] propose an online private-key challenge-response protocol in which each tag has a private key, and the centralized database stores all the tags' private keys. The tag's reader communicates the EPC of the tag to a Cryptographic Unit (CU), located at the manufacturer's site. The CU generates and sends a challenge to the reader, which in turn sends it to the tag. The tag then encrypts the challenge with its private key and sends the encrypted challenge (response) to the CU. The CU searches the centralized database for the tag's private key using the tag's EPC, and uses this key to verify the response. If the verification is successful, the product is genuine, or else the product is counterfeit. It is assumed that the tag's private key is only accessible to the tag's processor. This approach as analyzed by Saeed et al. [79] results in significant communication and computation overheads because the manufacturer stores the unique private key for each tag, generates a challenge for each received EPC request, and verifies the corresponding response. In addition, this approach cannot detect tag reapplication attacks.

The second are **offline approaches**, which authenticate products without con-

tacting a database. Offline approaches are suitable when products are on the market, and involve customers in the authentication process [79]. Lehtonen et al. claim that offline authentication needs further research and state, “Offline authentication remains unsolved as practically all existing techniques need online servers” [51]. Saeed et al. [79] 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 benefits of this approach are that it involves customers in product authentication, and it is offline. However, this approach does not detect tag reapplication attacks, and requires expensive NFC tags that have processors, secure storage, and support encryption. TRD is inspired by Saeed et al.’s approach, but aims to detect tag reapplication attacks and to use less expensive NFC tags that do not require these special features.

Recently, TagPrint [99], an offline cryptographic approach, was proposed by Yang et al. to detect counterfeit products using RFID. 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 affixes 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. In contrast, TRD is more user-friendly since it is based on NFC tags, which are integrated into an increasing number of cell phones and aims to detect reapplication attacks.

On the subject of tag reapplication attacks, Nochta et al. [69] propose a cryptographic approach based on RFID tags to detect such attacks by digitally signing the tag's ID and product-specific features. These features can be chemical or physical properties used to identify the product, such as the product's weight. Therefore, if a counterfeiter removes a tag from a genuine product and reapplies it to a counterfeit one, these features will not correlate. The verifier still needs to check these features manually after verifying the digital signature. Although this approach detects reapplication attacks, Lehtonen et al. [51] analyzed this approach, and recognized the complexity of verification. TRD eliminates manual verification, and simplifies the verification process by using a NFC-enabled cell phone.

2.4 Proposed Approach

This section describes the TRD approach in detail. The approach uses an authentication server located at the manufacturer's site to communicate with the supply

Figure 2.3: TRD architecture. Three parties are involved: (1) the manufacturer and its authentication server, (2) the supply chain nodes, and (3) the final consumers. There are two product authentication protocols: (1) online between the nodes and the authentication server, and (2) offline by a consumer's phone when the product is on the market.

chain nodes. The authentication server has access to the manufacturer's database in which the products' details are stored. When a product is in the supply chain, a supply chain node contacts the authentication server through a secure channel (i.e., the Internet), to authenticate the product using an online authentication protocol. After successful authentication, the authentication server updates the product's details in the database. When the product is on the market, an ordinary customer can authenticate the product by executing an offline authentication protocol between their cell phone and the NFC tag on the product. The overall architecture of the TRD approach is illustrated in Figure 2.3.

The product's manufacturer assigns each product with an NFC tag that contains the product's details such as serial number, name, and expiration date. TRD aims to use NTAG216F, the new NFC forum-compliant type 2 tag, developed by NXP [71]. These tags offer 888 bytes of user memory; each tag has a 7-byte read-only unique serial number (TID), and includes an NFC counter function. The counter function enables the tag to automatically increase a 24-bit counter value ($Counter$) each time the tag is read. The $Counter$ value cannot be otherwise adjusted or overwritten. In addition to the read-only memory, this kind of NFC tags has rewritable memory. TRD has two phases: initialization and verification.

2.4.1 Initialization Phase

This phase is responsible for initializing the details of each product, securing them, and storing them on the product's NFC tag. As mentioned earlier, each NFC tag has a read-only, unique TID and a read-only $Counter$, which is increased automatically on each reading of the tag. The product's manufacturer executes this phase using its unique private/secret key ($m.sk$) and public key ($m.pk$). As shown in Figure 2.4, for each product, the manufacturer performs the following:

1. Forms the product's details ($PDetails$) defined by:

$$PDetails \leftarrow PSN || PName || PExpiryDate || ToSignTID$$

Where:

- PSN is the unique product serial number, $PName$ is the product name, and $PExpiryDate$ is the product expiration date.
- $ToSignTID = TID$ (i.e., the tag's ID, that is attached to this product, is included in the product's details ($PDetails$) to be signed).

Figure 2.4: Product's manufacturer initialization. The manufacturer digitally signs the product's details ($PDetails$), and the number of the number of times the tag is read ($ReadingsOnTag$), and writes them on the tag. The read-only $Counter$ value is equal to the value of $ReadingsOnTag$.

- $||$ is a concatenation operator.

Note, we choose the product's details ($PDetails$) to be $(PSN, PName, ExpiryDate)$ because they are common features of most products.

2. Digitally signs $PDetails$ with $m.sk$ as follows:

$$SignedPDetails \leftarrow Sign_{m.sk}(PDetails)$$

Where:

- $SignedPDetails$ is the product's details digital signature.
- $Sign_{m.sk}(PDetails)$ is a digital signature function in which $m.sk$ is the signing key, and $PDetails$ is the data to sign.

3. Concatenates $PDetails$ and $SignedPDetails$ and writes them to the tag.
4. Reads the current value of the read-only tag's counter ($Counter$), that is, the value of how many times the tag has been read so far. We call this value $ReadingsOnTag$ defined by:

$$ReadingsOnTag \leftarrow Counter$$

5. Digitally signs $ReadingsOnTag$ with $m.sk$ as follows:

$$SignedReadingsOnTag \leftarrow Sign_{m.sk}(ReadingsOnTag)$$

Where: $SignedReadingsOnTag$ is the digital signature of $ReadingsOnTag$.

6. Concatenates $ReadingsOnTag$ and $SignedReadingsOnTag$ and writes them to rewritable memory on the tag.

After that, the product's details ($PDetails$) will be stored in the manufacturer's database. In addition the $PDetails$, the authentication server stores in the database the number of times the product's tag has been read (we will call it $ReadingsOnDB$). The values of $Counter$, $ReadingsOnTag$, and $ReadingsOnDB$ are equal. Then, the manufacturer affixes the NFC tag to the product and distributes the product through the supply chain.

2.4.2 Verification Phase

Once the product is distributed, two methods of verification phase can be used to authenticate the product. The first is executed in the supply chain, the second by the customer.

2.4.2.1 Verification in the Supply Chain

This verification is executed between the authentication server and the supply chain nodes. The authentication server uses the manufacturer's keys ($m.sk, m.pk$) to manage authentication in the supply chain. It checks and updates the value of *ReadingsOnDB* each time the product is authenticated by a supply chain node.

The purpose of this phase is to track the number of a tag's readings to ensure the tag has not been reapplied and has been authenticated by certified nodes. When node i receives a product, the node:

- Verifies the product's details signature (*SignedPDetails*) using the manufacturer's public key ($m.pk$) to ensure that the product's details have not been modified.
- Checks if the value of *ToSignTID* is equal to the tag's read-only *TID* to ensure that the product's details have not been cloned.
- Verifies the *SignedReadingsOnTag* using $m.pk$ to ensure that the value of *ReadingsOnTag* has not been modified.

The node then contacts the authentication server to check if the number of readings stored on the tag (i.e., *ReadingsOnTag*) is equal to the number of readings in the database (i.e., *ReadingsOnDB*). The values of *ReadingsOnTag* and *ReadingsOnDB* must be equal to the value of the tag's counter (i.e., *Counter*) before the authentication. Table 2.1 shows the notation of the online authentication protocol, and Figure 2.5 illustrates its process.

After successful authentication between the node and the authentication server, the node updates the value of the *ReadingsOnTag* on the tag to have the same value of *Counter* after the authentication. Also, the authentication server updates

Table 2.1: The notation of the online authentication protocol

Symbol	Definition
$m.sk$	The manufacture's private key
$m.pk$	The manufacture's public key
TID	The unique read-only tag ID
$Counter$	The read-only counter value, increased automatically each time the tag is read
$PDetails$	The product's details
$SignedPDetails$	The digital signature of $PDetails$
$ReadingsOnTag$	The value of how many times the tag has been read so far, stored on rewritable memory on the tag
$SignedReadingsOnTag$	The digital signature of $ReadingsOnTag$, stored on rewritable memory on the tag
$Verify_{pk}(D, S)$	A verification function that returns true if S is a valid signature on data D (with the public key pk) or false if not
$ToSignTID$	The tag's ID that is included in $PDetails$
PSN	The unique product serial number, which is included in $PDetails$
$Query$	The concatenation of PSN , $ReadingsOnTag$, and $Counter$, sent by node i to the authentication server
$ReadingsOnDB$	The value of how many times the tag has been read so far, stored in the database
$(Counter - 1)$	The value of the tag's $Counter$ minus 1, because we need the value of $Counter$ before node i 's authentication. We can substitute 1 by the value of node i 's readings ($reads$) if node i performs multiple readings prior to contacting the authentication server. $reads$ is then sent to the authentication server in the $Query$
Ack	An acknowledgement from the authentication server indicating that the values of $ReadingsOnDB$, $ReadingsOnTag$, and $(Counter - 1)$ are equal
$UpdatedCounter$	The value of $Counter$ after the product is authenticated by node i
$Sign_{sk}(D)$	A digital signature function where sk is the signing private key and D is the data to sign

the value of the $ReadingsOnDB$ in the database to hold the value of the updated $ReadingsOnTag$.

At the end of the supply chain, the product's tag is set to be authenticated offline without contacting the authentication server. This is achieved by writing the manufacturer's public key (i.e., $m.pk$) to the tag along with $ReadingsOnTag$,

Figure 2.5: Online authentication protocol. In all steps, if not pass or no received *Ack*, abort; the product is counterfeit. This protocol first authenticates the product, and secondly updates the readings value. The **Authenticate** process ensures that the values of *ReadingsOnDB*, *ReadingsOnTag*, and $(Counter - 1)$ are equal. After successful authentication, the **Update** process updates the values of *ReadingsOnDB*, and *ReadingsOnTag* to hold the same value of *Counter* after the authentication. The two keys (pk, sk) utilized in the protocol are for the manufacturer.

and *SignedReadingsOnTag*. The signature on the tag is stored as a “Signature Record” based on NFC Forum’s Signature Record Type Definition [67]. This record contains the message (i.e., *ReadingsOnTag*), the signature of this message (i.e., *SignedReadingsOnTag*), and a digital certificate that has the verification key (i.e., $m.pk$). The process at the final node ensures that the tag can be verified exactly once

Figure 2.6: Offline customer authentication protocol. In all steps, if not pass, abort; the product is counterfeit. The public key (pk) utilized in the protocol is for the manufacturer.

when the product is on the market.

2.4.2.2 Verification By the Customer

On receipt of the product, the customer executes this phase using their NFC-enabled cell phone. This phase is an offline authentication that does not require contacting the authentication server. As shown in Figure 2.6, when the customer scans the tagged product with their phone, the verification application will automatically execute the offline authentication protocol as follows:

1. Retrieve the product's manufacturer public key ($m.pk$), which is stored in the signature record on the tag.
2. Given the product data ($PDetails$), verify its signature ($SignedPDetails$) using $m.pk$ to detect **modification** on $PDetails$. If pass, proceed; if not, abort.
3. Check if the tag ID ($ToSignTID$) included in $PDetails$ is the same as the read-only tag ID (TID) to detect **cloning** of $PDetails$. If pass, proceed; if not,

abort.

4. Given the number of readings (*ReadingsOnTag*), which was updated by the final node, verify its signature (*SignedReadingsOnTag*) using *m.pk* to detect **modification** on *ReadingsOnTag*. If pass, proceed; if not, abort.
5. Check if the value of the tag's read-only (*Counter* - 1) is equal to the value of *ReadingsOnTag* to detect **tag reapplication**. Note that the protocol subtracts one from the *Counter* value because of the customer phone's reading. If pass, proceed; if not, abort.
6. Display the product details (the product serial number *PSN*, name *PName*, and expiration date *PExpiryDate*).

If the verification is successful, the product is authentic; if not, it is counterfeit. The product can be verified by only one customer. For example, if the product is sold and verified by Customer *A*, and an attempt is made by Customer *B* to verify this product, Customer *B* will get a warning message indicating that the product has been sold (used).

2.5 Analysis and Evaluation

In evaluating TRD, we followed the common practice of evaluating a product authentication system. Lehtonen et al. [51] stated that the security of a product authentication system can be evaluated by its ability to detect modified, cloned, and reapplied tags. However, active attacks, such as session-replay, and man-in-the-middle attacks in which a counterfeiter participates in the authentication session “are not considered realistic threats against a product authentication system” [51]. In addition to the security evaluation, the economic costs of the approach have to be considered.

2.5.1 Security Analysis

In this section, we analyze TRD from a security standpoint. The main goal of this approach is to detect tag reapplication attacks. In addition, it is possible to detect modification and cloning attacks.

2.5.1.1 Tag Reapplication Attacks

Tracking the number of times a product's tag has been read can successfully detect reapplication attacks. This is because the hardware *Counter* value is increased automatically each time the tag is read and cannot be adjusted. In addition, *ReadingsOnTag* and *ReadingsOnDB* are initialized by the product's manufacturer to be equal to the *Counter*, and are incremented each time the product reaches a supply chain node. Consider the following scenario:

- Product *P1* with tag *T1* has *Counter* = 0, *ReadingsOnTag* = 0, and *ReadingsOnDB* = 0 at the initialization phase at the product's manufacturer.
- Suppose that *ReadingsOnTag* at the final node is 20. The final node will write *ReadingsOnTag*, its signature *SignedReadingsOnTag*, and the manufacture's public key *m.pk* to the tag. Signing *ReadingsOnTag* protects it from modification.
- Assume that Product *P1* has been sold to Customer *A*. Customer *A* uses their phone to verify the product resulting in *Counter* = 21, as this value increases automatically due to the phone's reading. The verification application on Customer *A*'s phone executes the offline authentication protocol (Section 2.4.2.2) and indicates that the product is authentic. Now, *Counter* = 21, but *ReadingsOnTag* = 20, as this value is not affected by the verification process.

- Assume that a tag reapplication attack occurs —for example, a counterfeiter removes the tag of Product $P1$ (after Product $P1$ has been consumed) and reapplies it to a counterfeit Product $P2$.
- The victim customer (i.e., Customer B) buys the counterfeit Product $P2$ and uses their phone to authenticate it. The verification application will read the tag resulting in $Counter = 22$.
- The protocol requires the application to verify $SignedReadingsOnTag$ to ensure that $ReadingsOnTag$ is not modified. After that, the application will compare the $(Counter - 1)$ value, which in this case is 21 because the tag has been read at least once, to the $ReadingsOnTag$ value, which is 20. This results in detecting the reapplication attack.

2.5.1.2 Modification Attacks

The product's details ($PDetails$) are digitally signed by the product's manufacturer's private key ($m.sk$). The offline authentication verifies this signature ($SignedPDetails$) using the manufacturer's public key ($m.pk$) (step 2 in Figure 2.6). Therefore, the signature is invalid if any of the product's details is modified, such as the product's expiration date.

2.5.1.3 Cloning Attacks

Each NFC tag has a read-only unique TID , which cannot be modified or overwritten. This TID is included in the product's details ($PDetails$), as $ToSignTID$. $PDetails$ is digitally signed by the manufacturer's private key ($m.sk$). Verifying this signature involves reading the TID . Consider the following scenario:

- The details of Product $P1$ are stored on a tag that has a $TID = T1$. Therefore, $ToSignTID = T1$.
- A counterfeiter copies these details and uses them for a counterfeit product by writing them on another tag which has a $TID = T2$.
- When the offline authentication is executed, the verification application reads the counterfeit tag's TID (i.e., $T2$), the product's details ($PDetails$), and the product's signature ($SignedPDetails$). The application uses the manufacturer's public key ($m.pk$) to verify $SignedPDetails$, given $PDetails$. If the counterfeiter modifies $PDetails$ to have $ToSignTID = T2$, verifying $SignedPDetails$ will detect this modification (step 2 in Figure 2.6). If the counterfeiter does not modify $PDetails$, checking $ToSignTID = TID$ (step 3 in Figure 2.6) will compare the $ToSignTID$ (which holds the value $T1$) to TID (which holds the value $T2$). This results in cloning detection.

2.5.2 Economic Analysis

The economic costs of an approach can be evaluated by considering the cost of the used tags and the cost of their readers [51]. Since TRD can be executed using NFC-enabled cell phones and does not require specialized readers (such as RFID readers), we will not consider the readers cost. The cost of the tag is a key factor in using it for certain applications. For example, product authentication systems require a very large quantity of tags, which makes the price of the tag an essential factor. The price of NFC tags is influenced by several factors, such as memory capacity, and other features (e.g., processors, secure storage). The tags used for TRD are of type 2, which are low in cost compared to types 3 and 4 [16].

Table 2.2 compares TRD to the existing related approaches in Section 2.3 in terms

Table 2.2: Comparison of TRD to existing product authentication approaches.

Approach	Modification detection	Cloning detection	Reapplication detection	Cost of tag
Koh et al. [43]	Yes	Yes	No	Low
e-Pedigree [21]	Yes	Yes	No	Low
Staake et al. [88]	No	Yes	No	Low
Saeed et al. [79] [80]	Yes	Yes	No	High
TagPrint [99]	Yes	Yes	No	Low
Nochta et al. [69]	Yes	No	Yes	Low
TRD	Yes	Yes	Yes	Low

of ability to detect modification, cloning, and tag reapplication attacks. In addition to these security aspects, the cost of the tag is compared.

2.6 Discussion and Conclusion

We have described the Tag Reapplication Detection (TRD) approach to detect tag reapplication attacks based on low-cost NFC tags and public key cryptography. We have shown that TRD can detect a tag reapplication attack by tracking the number of times the tag has been read. The approach employs a database at the manufacturer to track the number of times a tag has been read in the supply chain. At the end of the supply chain, the final node sets the tag to be verified exactly once. After the product is distributed, the approach requires customers to authenticate products using their phones. Once a tag has been authenticated, it cannot be authenticated a second time, so reapplication is useless.

Although the proposed approach can detect tag reapplication attacks, detection failure is possible. This occurs when a counterfeiter removes and reapplies a tag without reading it. TRD cannot detect this attack, because the *Counter* value is not increased. However, the counterfeiter must buy the product to perform this attack, which conflicts with their financial motivation.

TRD can detect all three counterfeiting attacks (i.e., modification, cloning, and

tag reapplication). Nevertheless, this approach still suffers from the *centralization limitations* discussed in Section 1.2. The following chapter addresses this problem and presents a solution.

3 Block-Supply Chain

This chapter discusses the problem of centralized anti-counterfeiting supply chains (Section 1.2) and presents in detail our second contribution (the Block-Supply Chain) to overcome this problem.

3.1 Introduction

Explosive growth in pharmaceuticals and other products across the world has led to growth in large, globalized, digital supply chains [56]. The structure of modern supply chains is mostly centralized, in which a 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.

Centralized supply chains, however, face several limitations, as mentioned in Section 1.2. First, there is an enormous processing burden on the authentication server, since significant volumes of products flood through the supply chain nodes. Second, substantial storage is required to store authentication records for all products. Third, they 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. To overcome such limitations, blockchain technology stands out as a potentially promising solution.

Over the last few years, blockchain technology has been an attractive solution for many different industries beyond cryptocurrency [30]. The reasoning behind this is the transparency, security, quality assurance, global peer-to-peer transactions, and decentralization that the blockchain provides [74]. Fundamentally, a blockchain is a public, distributed ledger that contains chained blocks, each of which is made up of several transactions. These blocks are validated globally and transparently to guarantee security (i.e., they only comprise valid and correct transactions). The blocks are shared and synchronized across nodes via a peer-to-peer, distributed, and decentralized structure [56].

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 [56], Mackey and Nayyar stated 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 encouraged us to exploit this promising technology to combat counterfeit goods by transforming our previous centralized supply chain (i.e., TRD) to a decentralized Block-Supply Chain to ensure transparency, information security, and quality assurance.

In this chapter, we propose the Block-Supply Chain, a permissioned, decentralized supply chain that employs 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 blockchain 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 called validators to ensure that the block is valid. Upon successful validation, all nodes in the Block-Supply Chain network add this block to their copies of the blockchain, as will be explained in more detail in Section 3.4.

The Block-Supply Chain eliminates the need for the authentication server utilized in TRD. 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.

3.2 Background

3.2.1 Blockchain Technology

A blockchain is a distributed digital ledger of transactions. This ledger cannot be changed or tampered with because of the use of cryptographic methods [74]. The public/distributed ledger consists of blocks that are chained together. Each of these blocks has a hash. The hash is derived from the information included in the block. A new block is added to the blockchain by including the previous block's hash in it. This is how the blocks are bound together. Hence, if someone tampers with the previous block, its new hash will be different from the one included in the successive block. This mismatch will break the chain. As a result, the purpose of hashing is to ensure no one tampers with the blocks once they are validated and added to the chain. This chain of blocks is replicated on every node in the network.

There are three basic properties of a blockchain [30]:

1. It is **decentralized**: The blockchain's network is entirely run by its nodes, eliminating the need for a central authority that establishes trust. Thus, to add

a block to the ledger, this new block must be shared across the blockchain's peer-2-peer network. All nodes in the network keep their own local copy of the ledger.

2. It is **verified**: The blockchain nodes sign the transactions using public-private-key cryptography before broadcasting them to the network. Hence, only the owner of the private key can initiate them.
3. It is **immutable**: A blockchain network must utilize a consensus algorithm to preserve immutability. One or more transactions are grouped together to create a new block. This newly created block is proposed to the network by a node. Other nodes in the network can verify the transactions in the block. The consensus algorithm enables the nodes in the network to reach a decision on the validity of the new block. If consensus on the validity is reached, this indicates that all of the transactions in the block have been verified and are valid. Consequently, the block is added to the chain. Similarly, if no consensus is reached, the block is rejected. When a block is valid, a cryptographic hash is generated for that block to be included in the successive block.

There are two types of blockchain networks:

1. **Public blockchain**: This type is totally open, and anyone can join and participate in the blockchain network. In other words, any participant is able to read the chain and can write a new block into the chain. Typically, this type has an incentivizing mechanism to encourage more participants to join the blockchain [30]. Bitcoin [63] is one of the most known public blockchain networks. However, one of the shortcomings of this type is the substantial amount of computational power required due to maintaining a distributed ledger on a large scale [37].

Figure 3.1: Traditional supply chain authentication. As products ship from node to node in the supply chain, each node authenticates the products by contacting the authentication server.

2. **Private/permissioned blockchain:** This type of blockchain limits the parties who can participate in the blockchain network. Each participant must have permission to join the system. Participants can be determined at the start-up of the blockchain (genesis state), or by a set of rules put in place by the blockchain initiator [30]. This puts constraints on who is allowed to participate in the network, and on which transactions.

3.2.2 Traditional Centralized Supply Chains

Most of the existing products' supply chains rely on a third party (i.e., trusted server) to coordinate and manage the products' authentication as shown in Figure 3.1. In this typical architecture, each node authenticates a product upon its arrival. This authentication is carried out by executing some authentication protocols between a supply chain node, and the authentication server.

Figure 3.2: A general overview of the decentralized products' authentication architecture. It is a blockchain where the product's authentication records are grouped into a public ledger (i.e., blockchain) seen by every node in the supply chain. Each block includes an authentication record (*Record*), in addition to the hash (*H*) of the previous block.

3.2.3 Decentralized Supply Chains

Decentralized, blockchain-based supply chains maintain an open ledger in which every node in the supply chain can see and examine, as shown in Figure 3.2. This ledger shows the authentications history of a specific product. Hence, any counterfeit activity can be detected early by the supply chain nodes.

This type of supply chain structure authenticates products by involving the supply chain nodes without an authentication server. This is achieved by storing the product’s authentication records as blocks. The chain of blocks (i.e., the public ledger) is the chain of authentication events records, executed through the supply chain. Each event is created when a product is sent to the next node. Every node has the most recent ledger of the product. When a node tries to ship the product, this node proposes and broadcasts a new block to all nodes in the supply chain. This newly proposed block contains the recent product’s details such as the current location of the product in the supply chain. The new block is then chained to the product’s ledger by every node in the supply chain. This way, all nodes maintain the same copy of ledgers.

3.3 Related Work

In this section, we only present the works ¹ 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 [56].

Tian [91] proposed a conceptual framework of an agri-food supply chain using RFID and blockchain. This supply chain is designed to trace agri-food ”from farm to fork.” The author analyzed the advantages of using blockchains and stated that it can be used to fight fake products.

Saveen et al. [1] discussed the potential benefits of using blockchain technology in manufacturing supply chains. Then they proposed a framework for a manufacturing supply chain for cardboard boxes that involves blockchain as a platform to collect, store, and manage the details of each product throughout its life cycle. This work

¹Our dissertation is not strictly chronological. Several of the systems discussed here, such as TradeLens and the work introduced by Tseng et al. were proposed after we published our research on Block-Supply Chain.

introduced a general overview of replacing the centralized system with a decentralized system using blockchain technology.

In a study published by Hackius et al. [30], the authors conducted an online survey and asked professionals for their opinions on using blockchain in supply chain management. The authors found that most of the participants were positive about blockchain and the benefits that it can offer. This includes employing blockchain to fight counterfeit products.

Korpela et al. [44] conducted a study about integrating blockchain into 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 provide visibility in the supply chain. Their analysis showed that many of the digital supply chains' functionalities can be embedded in blockchain technology. In addition, blockchain provides security and cost-effective transactions in supply chains networks with no central system.

Recently, TradeLens [35] was introduced by IBM and Maersk to digitize current global supply chains using blockchain technology. TradeLens is a network of industry participants used to maintain a distributed, permissioned ledger. This ledger records the supply chain events, authority approval status, and full audit history. Each change or event creates a new, immutable block. TradeLens is a permissioned blockchain network that enables access only the parties participating in a specific shipment. TradeLens can track shipments in real time, and improve data sharing between shipping partners using the blockchain encryption and powerful permission-based sharing. However, TradeLens is a blockchain project that aims to transform the traditional track-and-trace supply chains, but does not undertake the hard-to-detect counterfeit attacks on products such as cloning and tag reapplication.

Tseng et al. [93] applied Gcoin blockchain's double-spending prevention mecha-

nism to mitigate the problem of counterfeit drugs. The authors exploited the Consortium Proof-of-Work in the Gcoin blockchain to tackle the counterfeiting problem in the pharmaceutical sector. Their approach tracks every pharmaceutical product in the supply chain, where the authors treated the shipping node as a seller and the receiving node as a buyer. The miners in the proposed approach are suggested to be the large manufacturers and government agencies, where the other nodes in the supply chain are full nodes, which are responsible for storing a backup of historical transactions about products. The proposed approach tracks and traces products using the records on the Gcoin blockchain, and protects against injecting new fake products into the supply chain. However, this approach fails to address the cloning and reapplication attacks on tagged products. In addition, it is partially centralized, since it does not shift the mining or validation responsibility away from the products' manufacturers.

In a more relevant work, Toyoda et al. [92] proposed a product ownership management system of RFID-affixed products to protect against counterfeiting in the post supply chain. This system utilizes RFID tags and Bitcoin's blockchain to prove the ownership of a product once the product is distributed to the market (i.e., post supply chain). The proposed work protects against cloning attacks by making the job for counterfeiters to clone genuine tags hard, since they cannot prove the possession of products. The main goal of the proposed system is to enable customers to reject purchasing counterfeit products, even with a genuine tag's Electronic Product Code (EPC), in case if the seller does not possess their ownership. Although this work was able to encounter the cloning attacks on products, it did not discuss the problem of reapplied tags.

3.4 Proposed Block-Supply Chain

This section explains our Block-Supply Chain in detail. The Block-Supply Chain is a 'permissioned' blockchain that comprise the nodes in the supply chain. It 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 Block-Supply Chain:

1. **Proposing (proposer):** This is the node that 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 has an NFC tag, which contains the product's details such as serial number, name, and expiration date.

3.4.1 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. We utilize the same NFC tags used

in TRD, where each NFC tag has a read-only unique tag ID (TID) and a read-only *Counter*. The read-only *Counter* is increased automatically on each reading of the tag and keeps track of the number of times that the tag is read by the nodes.

The manufacturer executes this phase in the same way described in TRD (Section 2.4.2). It forms the product's details ($PDetails$), which includes the following:

- 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 ($SignedPData$). After that, it writes the product data ($PDetails$) and its signature ($SignedPData$) to the product's tag.

Once the tag is prepared, the manufacturer creates a *genesis block* for the product. Each other 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 3.3 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}).

Figure 3.3: Blocks' structure and chaining them. Two blocks are chained by including the hash of the previous block in the new block.

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 address of the node the product is being shipped to (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 broadcasts the genesis block to all nodes in the supply

chain 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*.

3.4.2 Verification Phase

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

3.4.2.1 Local Authentication

When node i receives a product, the node authenticates the product locally by executing the *local authentication* algorithm, as shown in Algorithm 1. The local authentication involves the following:

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

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

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). This is to detect **cloning** of $PDetails$. If pass, proceed; if not, abort.

Algorithm 1: Local authentication

Input : $Block_{(i-1)}$, *Data On Tag*
Output: *ProductStatus* (Authentic, or Counterfeit)
1 $ProductStatus \leftarrow \text{"Counterfeit"}$
2 **if** $Verify_{m.pk}(Tag.PDetails, Tag.SignedPDetails)$ **then**
3 **if** $Tag.PDetails = Block_{(i-1)}.PDetails$ **then**
4 **if** $Tag.TID = Tag.ToSignTID$ **then**
5 **if** $(Tag.Counter - 1) = Block_{(i-1)}.ReadingsOnBlock$ **then**
6 $ProductStatus \leftarrow \text{"Authentic"}$
7 **end**
8 **end**
9 **end**
10 **end**
11 **Return** $ProductStatus$

4. Check if the value of the tag's read-only ($Counter - 1$) is equal to the value of $ReadingsOnBlock$ stored 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 i '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 i proceeds accordingly. If the product is counterfeit, node i 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 TRD, the node that 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 remote authentication at this point 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.

3.4.2.2 Global Authentication

After successful *local authentication*, and before dispatching the product to the next node in the supply chain, node i becomes a *proposing node*, and proposes a new block. The proposed block contains some updating details as given below:

- 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.
- The new value of *ReadingsOnBlock* which is the current number of reads of the tag's read-only *Counter*.
- The hash of the previous block (H_{i-1}).
- The digital signatures of the previous block's validators.

The proposing node then broadcasts this newly created block to all nodes in the blockchain network. The *validators* validate the block globally and vote on it.

The validators, as shown Algorithm 2, execute the *global authentication* algorithm, which includes the following steps:

1. The validators ensure that the blocks 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 current proposed block ($block_i$).
2. They check if *PDetails* in the previous block is the same as the one in the proposed block.
3. They trace-and-track the product by ensuring that the source (Src) in the block ($block_i$) is equal to the destination (D) in its previous block ($block_{i-1}$). This is

Algorithm 2: Global authentication

Input : The proposed $Block_i$, current blockchain $[Block_1 - Block_{i-1}]$
Output: $BlockStatus$ (Valid, or Invalid)
1 $BlockStatus \leftarrow \text{"Invalid"}$
2 **if** $hash(Block_{i-1}) = Block_i.H(i-1)$ **then**
3 **if** $Block_{i-1}.PDetails = Block_i.PDetails$ **then**
4 **if** $Block_{i-1}.D = Block_i.Src$ **then**
5 $BlockStatus \leftarrow \text{"Valid"}$
6 **end**
7 **end**
8 **end**
9 **Return** $BlockStatus$

to ensure that the product has been supplied through certified, valid nodes in the supply chain.

If the three checks are successful, then the block is valid, and it is safe to include it in the blockchain. The validators guarantee that the blockchain always contains valid blocks, so the next *proposing node* (i.e., node $i + 1$) can safely rely on the last block in the blockchain (i.e., $Block_i$) when executing its *local authentication* algorithm.

3.4.2.3 Verification By Customers

Once the product is distributed to the market, the end consumer can authenticate the product. There are two options to consider for authenticating the product in this phase. First, *offline authentication*, in which we do not involve the blockchain technology in the authentication process. Instead, a customer, as in TRD, can simply check the validity of a product using their NFC-enabled cell phone. The number of the tag's reads (i.e., $ReadingsOnBlock$) needs to be stored on the tag as $ReadingsOnTag$ along with its digital signature, similar to TRD. The only difference here is that this value (i.e., $ReadingsOnTag$) will be signed by validators of the last proposing node in the supply chain. The second option is the *online authentication*, which treats the customers as nodes in the blockchain. However, this option will need applying new

roles to grant access to new members, since Block-Supply Chain is a permissioned blockchain.

3.5 Discussion and Conclusion

We have proposed a new decentralized supply chain (Block-Supply) utilizing blockchain and NFC technologies. The Block-Supply Chain is a transformation of our centralized TRD (Chapter 2), and is able to track-and-trace products and detect modification, cloning, and tag reapplication attacks without relying on an authentication server.

This far, we have only discussed the role of validators, but how do we select them, and how can they communicate to reach an agreement on a block's validity? Our proposed consensus protocol answers these two questions in Chapter 4.

4 True Decentralized Consensus Protocol

This chapter discusses the problem of the lack of true decentralization in the existing BFT consensus protocols (Section 1.2) and presents in detail our third contribution (TrueBFT: a truly decentralized consensus protocol).

4.1 Introduction

Blockchain technology was introduced to solve the problem of reaching agreements on a state among distributed nodes without a coordinating third party [9]. These nodes are trust-less 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, much research has been conducted to come up with robust yet efficient consensus protocols that guarantee consensus in blockchains.

As mentioned in Section 1.2, a consensus protocol ensures 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 [9]. Consensus protocols are significant because blockchains are trustless, distributed nodes that need a way to synchronize their copies of stored data. Hence, a consensus protocol is designed to achieve the reliability in a network that has multiple unreliable nodes [9].

There are a considerable number of existing consensus protocols. Among these

protocols, the Byzantine Fault Tolerance (BFT) protocols which are known for their efficiency (in terms of the time taken to reach a consensus on a proposed block). Nonetheless, despite the efficiency of BFT protocols, not all of them guarantee *true decentralization* in which variable sets of anonymous validators execute the blocks' validation. Instead, many of them rely on fixed, known validators (e.g., Tendermint [9, 47], Practical Byzantine Fault Tolerance (PBFT) [12], and Hyperledger Fabric [34]). This fixed-validators approach opens the door for various risk threats, that will be discussed shortly. In addition, most of the current consensus protocols, such as PBFT [12], Tendermint [9, 47], Hyperledger Fabric [34], and Stellar [60], do not take the *number of validators* into consideration. The number of selected validators greatly influences a protocol's security and performance. In this chapter, we mainly address the problem of lacking *true decentralization* in the current consensus protocol and, briefly, the problem of validators' selection in terms of *how many validators to select* (more on this problem on Chapter 5).

The first and most popular consensus protocol to secure and decentralize blockchains is the **Proof of Work (PoW)** (e.g., Bitcoin [63]). This protocol requires powerful nodes known as *miners* to validate the transactions in a blockchain and propose new blocks. This consensus approach, however, demands massive computational effort from the miners, which ultimately results in high consumption of energy and computing resources. Furthermore, such blockchains frequently fork. As a result, the blockchain nodes are not able to rely on a new block as soon as it appears. Alternatively, they must wait until this block is deep enough in the chain, which results in very high latency [29]. To overcome these issues, the blockchain community has introduced new alternative approaches in which a chosen set of nodes known as *validators* are responsible for handling the block validation process. Next, we discuss some of these approaches as well as their strengths and weaknesses.

An alternative approach that does not require the expensive PoW computation and therefore enhances *efficiency* is the **BFT Fixed-Validators Decentralization** (e.g., Tendermint [9, 47], Hyperledger Fabric [34], and Ripple [83]). In this approach, a small fixed number of nodes are chosen to be validators. This approach ensures the integrity of the blockchain as long as the majority of the validators are honest [9, 12, 34, 47]. The validators are typically selected at the genesis state, and they are usually selected based on the stake they have (e.g., Tendermint). However, the efficiency of such protocols is influenced by the number of selected validators. This is because each validator performs some work to check the validity of a block and communicates with other validators in the committee to reach a consensus. This incurs computation and communication overhead proportional to the committee size (i.e., number of validators). Besides efficiency, Fixed-Validators Decentralization approach enjoys the *ease of implementation*, since the validators are known in advance [48].

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 [26]. Second, a fixed committee of validators is vulnerable to adversarial attacks, since they are known and fixed. For example, an adversary can launch a DDoS attack against the validators, preventing them from validating new blocks or receiving messages from each other. Third, 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.

The second alternative approach to the PoW is the **Rotating-Validators Decentralization** (e.g., Casper [100], and the new version of Tendermint). In this approach, the fixed committee of validators is replaced with a *rotating-committee* every period of time. An example is where there are N sets of committees, and upon

every block proposal, one of these committees is selected to carry out the validation process. This approach is efficient based on the size of the committee, as discussed before. However, it also relies on a strong trust assumption as most committee members remain fixed over a long period of time. As a result, we must assume that they should remain honest for that time and are not corrupted by an adversary [26]. Moreover, this approach is vulnerable to attacks mounted by a powerful adversary. Clearly, an adversary can not attack most of the nodes in the network, but can definitely launch attacks like DDoS on the majority of the members of a few, small committees [26].

The third approach is **True Decentralization**, in which every node in the system can be chosen to be a member of the validators. In such an approach, a set of validators is selected randomly from the set of "all nodes" on every block's proposal. In other words, it does not require a single set of validators to validate all the proposed blocks. As a result, the true decentralization approach distributes the validation work among all nodes, and can withstand powerful adversaries. Note that the fixed-validators approach is defined on the same group of validators and do not support validators' replaceability. In response, we propose a novel, robust yet straightforward, truly decentralized consensus protocol (TrueBFT) that selects a different set of random validators on every block's proposal. TrueBFT mainly addresses the issues of a) the lack of true decentralization, b) the fixed number of validators in BFT consensus protocol. Besides, the proposed protocol is designed to work only with permissioned blockchains.

In TrueBFT, at the genesis state, the blockchain initiator randomly maps each *proposing node* to four *validation-leader nodes*. 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 this proposer. TrueBFT achieves true decentralization by, first, using 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 genesis state. Second, to further enhance true decentralization, TrueBFT requires each validation-leader to select a random set of validators without any communication with other peer leaders. Therefore, the identities of leaders and validators remain anonymous, preventing powerful attackers from launching attacks such as DDoS and Eclipse. This is because the attackers do not know these nodes in advance. Additionally, the set of selected validators are different for each proposed block due to exploiting randomness. More on this in Section 4.4.

The proposed TrueBFT protocol is BFT-based, and leverages the consensus algorithm utilized by Tendermint. Tendermint is notable for its simplicity, performance, and fork-accountability [48]. However, Tendermint’s performance depends on the number of validators [9]. The smaller the number of validators the higher the performance. This is due to the validation work performed by every validator and the communication overhead between validators to reach the consensus. Consequently, TrueBFT deals with *the number of selected validators* issue by selecting a relatively small number of validators proportional to the network’s hostility, and total number of nodes. Put differently, we allow TrueBFT to select a number of validators proportional to a risk threshold (*risk*). Hence, in a less hostile network environment, fewer validators are employed. This risk threshold, however, is static and set manually. We overcome this drawback (in Chapter 5) by integrating a game theoretical model into TrueBFT which can determine the risk likelihood of each proposing node dynamically.

To illustrate the trade-off between security and performance, we have simulated the proposed TrueBFT protocol. Our base reference protocol was Tendermint due

to its noteworthy performance. We chose Tendermint to be optimally secure (i.e., employs $n - 1$ of the nodes as validators), so that we can compare the security and performance of TrueBFT relative to this choice. We set the *risk*, so that TrueBFT employs $\log n$ validators, rather than the optimal choice of $n - 1$ validators. The reasoning that has led to this specific value (i.e., $\log n$) is to reduce the validation work from $O(n)$ to $O(\log n)$, and the communication overhead from $O(n^2)$ to $O((\log n).n)$. We found that this reduction resulted in a considerable enhancement in performance with a satisfactory level of security when compared to Tendermint, as will be shown in Section 4.5.

4.2 Background

4.2.1 Blockchain Consensus Protocols

A blockchain is a decentralized peer-to-peer network without a central authority. Although this architecture defeats corruption from a single source, it introduces a new problem. This problem can be illustrated in the following questions [81]:

- Who controls decisions? How to agree on them?
- Since the data is not stored in a single centralized storage, how to replicate this data on nodes? How to synchronize these replicas?

In a typical centralized architecture, all the decisions are made by a single controlling authority/leader or a committee of decision makers. This, however, is not possible in a blockchain network, since there is no leader or a group of leaders that controls the system. Therefore, making decisions in a blockchain requires a "consensus mechanism" to come to a consensus on any decision. Moreover, the data in a centralized organization is stored on a machine or a group of machines, so the nodes

in the network can read and write. In a decentralized blockchain, every node has its own copy of data, grouped in a ledger of blocks. The role of a "consensus mechanism" is to synchronize the copies of these ledgers, and ensure that they contain valid blocks.

A blockchain consensus mechanism/protocol is a series of actions and votes taken by a group of nodes to reach an agreement on a proposed block [81]. It guarantees that the blocks in a public ledger are always valid and have the same order. Consensus protocols must satisfy the following aspects [11]:

- **Liveness:** It means that something good *eventually* will happen. In distributed computing, it guarantees that the computation terminates. In consensus, all processes (nodes) will eventually agree on a value.
- **Safety:** Guarantees that something bad will *never* happen. In consensus, no two processes (nodes) decide on different values (i.e., correct decisions).

4.2.2 Broadcast and Consensus

In a fault-tolerant distributed system, an *atomic broadcast* is a broadcast where all correct nodes in the system receive the same set of messages in the same order [11]. Such a broadcast is either completes correctly at all nodes, or all nodes abort with no side effects. For a message m , atomic broadcast satisfies the following properties [10, 31]:

- **Validity:** If a correct node broadcasts m , it delivers m eventually.
- **Agreement:** If a correct node delivers m , then all correct nodes deliver m eventually.
- **Integrity:** Message m is received (delivered) by each node at most once, and only if it was previously broadcast.

- **Total order:** If the correct nodes p and q deliver m_1 and m_2 , then p delivers m_1 before m_2 if and only if q delivers m_1 before m_2 .

Intuitively, *consensus* and *atomic broadcast* perform remarkably alike. The only critical difference is that *atomic broadcast* is a continuous protocol, while *consensus* expects to terminate [9].

4.2.3 Byzantine Fault Tolerance (BFT) in Distributed Systems

Byzantine faults problem is one of the serious problems that face distributed systems, including blockchains. The goal of Byzantine fault tolerance in a distributed system, is to defend against failures, where parts of a system fail in arbitrary ways [62]. These failures could be crashing, processing requests incorrectly, or even acting maliciously. An example of a Byzantine fault problem is illustrated in Figure 4.1. In this example, each node has a sensor that reads a value. The value is then broadcast to all other nodes in the network. The nodes, however, need to agree on a single value (i.e., state) so each node will have the same value as the other nodes. Consider the following:

- Assume that the voting algorithm is to take the median of all broadcast values as the state value.
- Assume that node C is Byzantine. Hence, C sends value = 9 to node A and value = 12 to node B .
- A computes the median of $\{10, 11, 9\} = 10$, and B computes the median of $\{10, 11, 12\} = 11$.
- As a result, C 's malicious behavior causes A and B to work with inconsistent state values.

Figure 4.1: An example of a Byzantine fault problem. This is a network of three nodes, in which nodes *A* and *B* are honest (regular) nodes, and node *C* is malicious (Byzantine). Source : [62].

This problem can be defined in blockchains as how the distributed nodes in the network reach an agreement on a non-replicated block in the presence of Byzantine faults [31]. In this case, simple voting algorithms that involve all nodes will not be able to deal with Byzantine faults [62].

4.3 Related Work

In this section, we will examine related existing consensus protocols. The related literature falls into four general camps: (1) Proof of Work (PoW) protocols, (2) Proof of Stake (PoS) protocols, (3) Byzantine Fault Tolerance (BFT) protocols, and (4) non-Byzantine Fault Tolerance protocols.

4.3.1 Proof of Work (PoW)

In 2009, Satoshi Nakamoto introduced Bitcoin [63], 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 is 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 entail 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 [9].
2. High consumption of resources: due to the significant computation to solve a challenge that requires computing cryptographic hashes.
3. High energy consumption: which results in massive expenses.
4. Specialized hardware: which is sometimes required to increase the mining power.

4.3.2 Proof of Stake (PoS)

The most common alternative to PoW is Proof of Stake (PoS). In 2012, King and Nadal [42] 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 mine 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 [53]. However, some argued that PoS is not ideal for blockchains [75]. 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.

4.3.2.1 Delegated Proof of Stake (DPoS)

Permissionless/public blockchains that utilize the traditional PoS often face scalability issues [50]. As a result, Delegated Proof of Stake (DPoS) [50], a variation of the PoS, was adapted by some blockchains such as Lisk [59], EOS [16], BitShares [82], and Ark [45], 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 collecting 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, which are responsible for validating transactions and creating blocks for these transactions, and as a result are awarded the associated fees.

4.3.3 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 (crashing, or malicious nodes). Our proposed TrueBFT protocol falls into this category.

4.3.3.1 Practical Byzantine Fault Tolerance (PBFT)

PBFT [12] 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 [9]. PBFT works in asynchronous environments that might contain Byzantine faults such as the Internet [12].

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* message to the other backups. The *pre-prepare* message 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* message. After accepting, 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 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's request to the state machine and sends back the result to the client. Figure 4.2 illustrates this process.

PBFT uses a timeout mechanism to deal with faulty primaries. Backups keep

Figure 4.2: Normal case operation of PBFT. Source: [12].

a timeout that starts every time they receive a new request. This timeout finishes when the requested *pre-prepare* message 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 [9]. 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, migrate to the new view.

4.3.3.2 Tendermint

Tendermint [9,47] 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 [9]. 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 PoW mining, which overcomes the energy and resource consumption issues and speeds up blocks' validations [47].

Tendermint is based on PBFT, and it involves three stages of voting to reach consensus (*propose*, *prevote*, and *precommit*). A proposer *proposes* a new block, then

the validators *prevote* on the block and only proceed to *precommit* if they receive more than two-thirds of *prevotes*. Likewise, validators only accept the block if more than two-thirds of *precommits* are received. Tendermint, as mentioned earlier, is a fixed-validators protocol; that is, it has 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 [48]. 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, but degrade performance. TrueBFT 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). Furthermore, the ‘number’ of selected validators in TrueBFT is a fraction of the total number of nodes in the network proportional to the risk likelihood of the proposing nodes.

4.3.3.3 Hyperledger Fabric

Hyperledger Fabric [34] employs PBFT as its consensus algorithm. 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 in order. 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 of the 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.

4.3.3.4 Stellar Consensus Protocol (SCP)

SCP [60] is a consensus protocol that utilizes *quorums*, where a quorum is a set of nodes from a network sufficient to reach an agreement. SCP is based on Federated Byzantine Agreement (FBA), in which SCP exploits the concept of a *quorum slice*. A quorum slice (can be smaller than a quorum) is the subset of a quorum that can cause one particular node to reach an agreement. A node can have one or more quorum slices. The key idea in FBA is that every node chooses its own quorum slices. For example, a node might choose a reputable bank and other financial company to sign on all transactions. A node accepts a vote or a transaction when a threshold (e.g., two-thirds) of nodes in its quorum slice confirms it. However, SCP requires the quorum slices to overlap.

The previously discussed protocols such as PBFT, Hyperledger Fabric, and Tendermint employ a fixed and globally known set of nodes to reach a consensus. In contrast, SCP gives each node a choice to select one or more quorum slices, each of which might have different nodes. Despite the beauty of this design, it might result in undermining the consensus as quorum slices might not overlap [48]. If quorums are disjointed, quorum *A* might, for example, agree on a statement *X*, while quorum *B*

agrees on a statement Y . As a result, A and B independently agree on contradictory statements. Therefore, no consensus is reached. SCP relies on the assumption that every node is responsible for choosing a quorum slice that doesn't violate quorum intersection (i.e., non-empty intersection) [48].

4.3.3.5 Ripple

Ripple [83] was introduced in 2012 to enable global payments using 'XRP' cryptocurrency by connecting payment providers, banks, and businesses via a network called 'rippletnet.' 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 to 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 [11]. This corresponds to tolerating $f < n/5$ faulty nodes. Additionally, Ripple, like Stellar, requires a minimal overlap among the Unique Node Lists to avoid forking. Moreover, as argued by Cachin et al. [11], Ripple "is by far not as decentralized as advertised", since Ripple offers a default list of validating nodes operated by Ripple and third parties.

4.3.3.6 Tangaroa

Tangaroa [15] or BFTRaft is a Byzantine Fault Tolerant variant of the Raft [73] consensus algorithm. It was introduced to address the Byzantine behavior in which Raft is not tolerant to. Tangaroa, like Raft, works in views, each of which has a leader. The leaders are responsible for driving the consensus process. Tangaroa utilizes randomized timeouts to trigger the elections of leaders.

In Tangaroa, a client sends a request to the leader once a leader has been elected. When the leader receives a request from a client, it sends an append command to each replica/follower. As in PBFT, the client needs to wait for $f + 1$ matching replies to the request, in which $f < n/3$ where n is the total number of nodes in the network.

Cachin et al. [11] analyzed Tangaroa, and they argued that Tangaroa is neither live nor safe (we refer the reader to [11] for in-depth details). Another issue of Tangaroa is that the consensus process cannot be accomplished without a single leader for each view. This exposes the leaders to adversaries who could launch attacks on these leaders such as DDoS to undermine the consensus process.

4.3.3.7 Algorand

Recently, Algorand [26], a new fast Byzantine fault-tolerant consensus protocol, 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 a 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 it robust against DDoS and Eclipse attacks. Nevertheless, Algorand provides no incentive [46], and does not guarantee that the verifiers will adhere to the protocol. It is possible for attacks like block withholding and lazy verifiers to appear in Algorand. TrueBFT, in contrast, avoids this issue by leveraging a rewarding mechanism based on a game theoretical model (Chapter 5).

4.3.4 Non Byzantine Fault Tolerance

4.3.4.1 Paxos

Paxos [49] is an asynchronous consensus protocol and is quite similar to PBFT. However, it requires only $2f + 1$ nodes to tolerate f faults [9]. In Paxos, there are two parties: (1) the proposer (leader), which proposes a value (block), and (2) the acceptors, which accept 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 a majority/quorum.

4.3.4.2 Raft

Raft [73] is a consensus protocol that has been introduced to be easy to understand. It is similar in spirit to Paxos in fault-tolerance. Raft works by electing a leader who 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 a majority of the followers have confirmed the appending command. Raft is used by many blockchains such as R3 Corda [32], and Kadena [58].

4.3.4.3 Casper “The Friendly GHOST”

Casper [100] is the consensus mechanism in Ethereum [96]. It is an adjustment of some of the principles of the GHOST protocol [86] (Greedy Heaviest-Observed Sub-Tree). Casper in Ethereum was presented with a security-deposit based economic consensus protocol. In other words, each node that 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. In 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, a validator bundles the new transactions in a block, validates 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.

4.4 The Proposed TrueBFT Consensus Protocol

In this section, we propose the TrueBFT, a new consensus protocol that achieves true decentralization by employing a different, random set of validators every time a new block is proposed. Additionally, TrueBFT addresses the problem of validators' selection regarding 'how many to select' by selecting a *number of validators* proportional to the environment's hostility. TrueBFT 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, TrueBFT replaces the set of validators each time a new block is proposed. Thus, TrueBFT improves 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. With respect to efficiency, TrueBFT decides the number of validators based on the risk likelihood (i.e., *risk*) of the proposing nodes. To show how secure is TrueBFT even with a relatively small number of validators, we set *risk* so that the number of validators is equal to $\log n$. This choice saves substantial computational and communication costs and achieves high security.

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. The calculation of R is presented in Section 4.4.2.4. There are four types of nodes in TrueBFT:

1. **Proposing (proposer):** This is the node which executes the *local authentication* algorithm in our Block-Supply Chain (Section 3.4.2.1). After successful authentication, it creates, proposes, and broadcasts to the network the new block for the product.

2. **Validation-leader:** This is the node responsible for determining the number of validators for the proposing node and randomly selecting them.
3. **Validator:** This node is responsible for validating the newly proposed block by executing the *global authentication* algorithm (Section 3.4.2.2). Moreover, validators communicate their votes on the block to reach consensus.
4. **Idle:** This node does nothing except waiting 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.

It is worth noting that the validation-leader and validator nodes are rewarded some fees associated with the proposed block for carrying out the consensus process.

TrueBFT 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 order, where the nodes have a pre-determined schedule specifying when each node becomes a proposer. 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. The next two subsections present an in-depth description of how these two phases are executed.

4.4.1 Initialization Phase

The main task of this phase is *mapping proposers to validation-leaders*. At the genesis state (i.e., when the genesis block is proposed), 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 sin-

gle Byzantine fault [9]. As TrueBFT is based on Tendermint, it is assumed that a Tendermint network has two-thirds non-Byzantine nodes. A simple approach is to employ only one validation-leader per 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 TrueBFT. Our approach works with any number of validation-leaders per proposer other than four, but we utilize the minimum for efficiency. Additionally, this number can be random to further increase robustness.

The mapping is executed randomly according to the nodes' weights/reputations (R). As shown in Algorithm 3, we use the Weighted Random Sampling (WRS) algorithm [20] due to its simplicity, flexibility, and its ability to select a weighted random sample in one-pass [20]. 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 validation-leaders, and no validation-leader knows its proposer. This way, we prevent a malicious proposer from corrupting or bribing its validation-leaders and vice versa.

After selecting the validation-leaders for a proposer and to accomplish the anonymous mapping, the blockchain initiator **first** includes a proposer's secret (S_1) in every proposer's genesis block, so this proposer uses this secret when it is ready to propose its block. S_1 is a hash that includes the proposer's public key ($pr.pk$), all four of the selected validation-leaders' public keys [$vl_1.pk - vl_4.pk$], the blockchain ID ($blockchainID$), and a random number ($Rand_1$) as follows:

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

We introduce the random number ($Rand_1$) to prevent brute force attacks on S_1 in which an attacker tries all the possibilities until he finds a hash that matches S_1 ,

and hence discovers the identities of the leaders. This attack is feasible because the attacker knows $pr.pk$, $blockchainID$, and all the public keys of the nodes in the network. As a result, without $Rand_1$, when a proposer broadcasts S_1 , the attacker can find the identities of the leaders by hashing every possible combination of four nodes' public keys with $pr.pk$ and $blockchainID$. A powerful attacker can accomplish that quickly, especially in a network with a small number of nodes. Utilizing $Rand_1$, where each proposer in the network has a unique, random $Rand_1$ prevents such an attack because $Rand_1$ is not known to the attacker (note that $Rand_1$ is not published to the system). It is worth noting that the larger the size of $Rand_1$, the securer the system.

There is only one proposer's secret (S_1) per node in the network to use it when the node becomes a proposer. Each proposer in the system has its own S_1 . When a proposer broadcasts its S_1 , this S_1 is checked by each node in the network to determine if it is a leader for the proposer, as will be discussed shortly.

Second, the blockchain initiator generates a validation-leader's secret (S_2) for each of the selected four leaders. S_2 is a hash that includes the corresponding 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, unique $Rand_2$ for each selected validation-leader to make S_2 different for each one of them. Note that S_2 and $Rand_2$ are private and are only known to its particular validation-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 secret (S_1), the validation-leader's public key ($vl.pk$), and the blockchain ID ($blockchainID$) as below:

Algorithm 3: Proposers to leaders mapping

Input : A population A of n nodes having reputation values (R)

```

1 foreach  $pr \in A$  do
2   for  $k \leftarrow 1$  to 4 do
3     Try:
4      $p_i(k) \leftarrow \frac{R_i}{\sum_{s_j \in A-B} R_j}$ 
5     Randomly select  $vl_i$  with probability  $p_i(k)$  from  $A - B$ 
6     if  $C_i.size > 4$  then
7       | Go to Try
8     else
9       |  $B.add(vl_i.pk)$ 
10    end
11  end
12  Randomly generate  $Rand_1$ 
13   $S_1 \leftarrow hash(pr.pk || vl_1.pk || vl_2.pk || vl_3.pk || vl_4.pk || blockcahinID || Rand_1)$ 
14  foreach  $vl_i \in B$  do
15    Randomly generate  $Rand_2$ 
16     $S_2 \leftarrow hash(S_1 || Rand_2)$ 
17     $\pi \leftarrow Sign_{in.sk}(S_1 || vl_i.pk || blockcahinID)$ 
18     $g \leftarrow \lceil ((i-1) \cdot \frac{n}{4}) + 1, i \cdot \frac{n}{4} \rceil$ 
19     $C_i.add(S_2 || Rand_2 || \pi || g)$ 
20  end
21  Flush  $B$ 
22 end

```

$$\pi \leftarrow Sign_{in.sk}(S_1 || vl_i.pk || blockcahinID)$$

The validation-leader must submit this proof to its selected validators, so that each can verify π using the initiator's public key ($in.pk$) prior to becoming involved 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. TrueBFT 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 that we assume that all the nodes in the network have the same set of nodes in the same order. As shown in Algorithm 3, g is predetermined by the blockchain initiator and is defined as below:

$$g \leftarrow \lceil ((i-1) \cdot \frac{n}{4}) + 1, i \cdot \frac{n}{4} \rceil$$

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

In Algorithm 3, 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). After executing Algorithm 3, 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.

4.4.2 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 is to decide the validity of the newly proposed block and to reach a consensus on this decision.

Algorithm 4: 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

```

1 decision  $\leftarrow$  false
2 foreach  $tuple_i \in C$  do
3   | if  $S_2^i = \text{hash}(S_1 || Rand_2^i)$  then
4   |   | decision  $\leftarrow$  True
5   | end
6 end
7 Return 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 validation-leader for this proposer as shown in Algorithm 4.

4.4.2.1 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. TrueBFT utilizes a risk likelihood threshold ($risk$). The number of validators (m) is proportional to the risk likelihood ($risk$). $risk$, as mentioned in Section 4.1, is set manually for all proposers in the network and is static. However, in Chapter 5, we enhance TrueBFT to make $risk$ dynamic and proportional to each proposer's likelihood.

For each validation-leader, the number of validators (m) will be a number bound by the minimum number of validators (i.e., four) and a fraction of $\frac{n}{4}$ proportional to $risk$ (we choose $\frac{n}{4}$ because we have four validation-leaders). The reasoning behind this choice (i.e., minimum number of validators) is that, as mentioned earlier, four is the minimum number to provide tolerance to a single Byzantine fault [9]. So, a validation-leader selects its m to be: (a) $\frac{risk \cdot (n-2)}{4}$ (note $n - 2$ because we exclude the

Algorithm 5: Deciding the number of validators per a leader (m)

Input : The number of nodes in the network n , the risk likelihood $risk$
Output: The number of how many validators to select m

```

1 if  $\frac{risk \cdot (n-2)}{4} > 4$  then
2   |  $m \leftarrow \frac{risk \cdot (n-2)}{4}$ 
3 else
4   |  $m \leftarrow 4$ 
5 end
6 Return  $m$ 

```

proposing and the validation-leader nodes), or (b) four if $\lceil \frac{risk \cdot (n-2)}{4} \rceil$ is less than four as shown in Algorithm 5.

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 m s from the validation-leaders, it calculates the overall number of the validators involved in the protocol (M) as follows:

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

This way, the overall number of validators is proportional to $risk$. Note that TrueBFT inherits the Byzantine tolerance provided by Tendermint. In other words, the system can work with less than one-third of Byzantine leaders (one faulty leader), of which M is the aggregation of only three m s. However, if more than one-third of leaders are faulty, TrueBFT, like in Tendermint, stalls momentarily, and then switches to a new round of proposing; thus preserving liveness.

4.4.2.2 Selecting Validators

Each validation-leader selects its set of m validators from the set of $\frac{n-2}{4}$ nodes (note each leader has its own pool to select from defined by g excluding itself and the proposer). The four sets of selected validators will be responsible for validating the

Algorithm 6: Validators' selection

Input : A population V of $\frac{n-2}{4}$ nodes having reputation values (R)
Output: A set of validators/pre-voters PV and a set of pre-committers PC of size m

```

1 for  $k \leftarrow 1$  to  $m$  do
2    $p_i(k) \leftarrow \frac{R_i}{\sum_{v_j \in V - PV} R_j}$ 
3   Randomly select  $v_i$  with probability  $p_i(k)$  from  $V - PV$ 
4    $PV.add(v_i)$ 
5 end
6 for  $l \leftarrow 1$  to  $m_i$  do
7    $p_i(l) \leftarrow \frac{R_i}{\sum_{c_j \in V - PC} R_j}$ 
8   Randomly select  $c_i$  with probability  $p_i(l)$  from  $V - PC$ 
9    $PC.add(c_i)$ 
10 end
11 Return  $PV$  AND  $PC$ 

```

proposed block. The validators are selected randomly, and each set of selected validators is only known to its selecting validation-leader. TrueBFT is based on Tendermint, which involves two steps of voting (pre-vote and pre-commit). 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, TrueBFT 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 not useful knowledge. Algorithm 6 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:

$$\tau \leftarrow \text{Sign}_{vl.sk}(vl.pk || pv.pk || \pi)$$

A node that 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 ($in.pk$).

4.4.2.3 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. TrueBFT is based on Tendermint and adapts its consensus approach. The *pre-voters* in TrueBFT *pre-vote* on the proposed block, and when the *pre-committers* hear from more than two-thirds of M 's other nodes, they *pre-commit* the block. The block is committed when more than two-thirds of M *pre-commits* are received. The consensus algorithm is summarized as follows:

1. When a node in the Block-Supply Chain becomes a proposing node, this node creates the new block and *proposes* it to the network along with its secret S_1 .
2. Upon receiving the proposal, each node in the network checks if it is a validation-leader, as mentioned earlier. If a node is a validation-leader, then it executes the "validators' selection" algorithm to randomly select m *validators/pre-voters* and

m *pre-committers*. Then, it sends a *validate* command to the selected validators and a *pre-commit* command to the selected pre-committers. Moreover, each leader broadcasts its m to all nodes.

3. After that, if a node receives a *validate* command, then it acts as a validator and carries on the validation process. However, 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*.
4. After receiving the *validate* command, 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 proposing's "*round-number*" to zero (*round-number* is a number that represents how many times a proposer has been given a chance to re-propose). Proposers are given multiple proposing rounds to allow them to re-propose in case if they fail to propose on time. 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 within the *proposer-time-out*, it validates the block, and *pre-votes* 'valid' if the block is valid or 'invalid' otherwise. However, if the *proposer-time-out* terminates, then the validator *pre-votes* 'timed-out', as illustrated in Figure 4.3.
5. When a pre-committer receives more than $\frac{2.M}{3}$ *pre-votes*, it *pre-commits* 'valid', 'invalid', or 'timed-out' according to the received *pre-votes* type.
6. When a node receives more than $\frac{2.M}{3}$ *pre-votes* followed by more than $\frac{2.M}{3}$ *pre-commits*, it *commits* 'valid', 'invalid', or 'timed-out' according to the received *pre-commits* type.

Figure 4.3: Our consensus mechanism. The first step is "Propose." After this step, validators only advance after hearing from more than two-thirds of M other nodes. By the end of the consensus, either the block is committed and the blockchain height increases, or the proposed block is rejected and the protocol aborts.

7. There is a final subsequent step following the *commit* step, and it is of three types, as shown in Figure 4.3. First, if a node *commits* 'valid,' then it adds the proposed block to the blockchain and extends it to a new height. Second, if the node *commits* 'invalid,' then it aborts the protocol (in applications other than the Block-Supply Chain, the protocol allows the next proposer inline to propose, as will be discussed in Chapter 6). Third, if the *commit* is of type 'timed-out,' then the validators check the *round-number* against a rounds' counter that

limits how many times a proposer can re-propose, we call it *rounds-limit*. If the *round-number* is less than or equal to the *rounds-limit*, the validators: a) increase the *round-number* by one, b) increase the *proposer-time-out* based on the network conditions, and c) start a new proposing 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., *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.

4.4.2.4 Calculating the Reputation Values (R) for the Consensus Nodes

Each node in the network has a global reputation/trust value (R) associated with its public key, which affects the node's chance to be selected as a validator or leader over time. Each node in the network has a list of other nodes' public keys and reputation values. Hence, the reputation of a node in the network represents its value [24]. This encourages the network's nodes to build good reputations by behaving well. At the genesis state, all nodes have the same initial R . However, as more blocks are proposed and validated, a participating node in the proposing and validation process will have a different value of R based on its behavior.

The reputation of a node participating in a ledger height H is updated in the next height (i.e., $H + 1$). The ledger is extended to a new height when a new, valid block is added to it; that is, a proposer proposes the block, and its validators and pre-committers reach agreement on its validity. When a proposer tries to propose a new block for height $H + 1$, it calculates the reputation values for all nodes that participated in height H . After that, this new proposer broadcasts these reputation values to all nodes in the network along with the newly proposed block. Accordingly,

the validators of height $H + 1$ validate the correctness of the proposed reputations and vote on them. When the validators and pre-committers of height $H + 1$ agree on the proposed reputation values, all the nodes in the network update R for each node that was involved in height H . This way, the nodes will have consistent global reputation values for other nodes without exchanging a large number of messages or performing much computation.

Note that the participating nodes in height H do not know the identities of the validators in height $H + 1$ due to anonymity and random selection. As a result, TrueBFT prevents bribing or corrupting the validators in height $H + 1$ to agree on false R s provided by a colluding proposer in height $H + 1$. Furthermore, due to validating and voting on R s by anonymous random validators, TrueBFT withstands the *bad-mouthing attack* [19], in which a dishonest proposer in height $H + 1$ provides dishonest R s to defame good nodes in height H .

To calculate R , we use the EigenTrust algorithm [39] due to its simplicity and effectiveness. In this algorithm, the proposer calculates the new R s of the height H 's participants as follows:

1. Rate every node in height H as positive ($tr_{ij} = 1$) or negative ($tr_{ij} = -1$), where node i is the rating node (i.e., the proposer) and node j is the rated node.
2. Calculate the trust value (s_{ij}) for node j by aggregating all the ratings of node j as follows:

$$s_{ij} = s_j + tr_{ij} \quad (4.1)$$

Where s_j is the sum of all the ratings of node j provided by the previous rating nodes except node i defined as:

$$s_j = \sum_z tr_{zj} \quad (4.2)$$

3. Calculate the reputation value (R_j) of node j by normalizing the trust value for node j as follows:

$$R_j = \frac{\max(s_{ij}, 0)}{\sum_j \max(s_{ij}, 0)} \quad (4.3)$$

This ensures that all R s will be between 0 and 1. For more details about the EigenTrust algorithm, we refer the reader to [39]. After calculating all the R s for the participants in height H , the $H + 1$'s proposer broadcasts the rating (tr_{ij}), the trust value (s_{ij}), and the reputation value (R_j) for every rated node j . Each corresponding validator first vitrifies the integrity of each provided tr ; that is, ensuring that the proposer provides an honest, fair rating. Then, the validator ensures the correctness of each calculated R s. The validators then vote on R s. Note that the remaining nodes in the network wait for the decision to be made by the validators and pre-committers. Once the consensus is reached, all nodes update their copies of R s.

4.5 Experiments and Evaluation

In this section, we evaluate the performance and security of our proposed TrueBFT 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, ability to maintain liveness and safety in the presence of Byzantine nodes, and most importantly its similarity to our blockchain use case (i.e., mining-less).

TrueBFT achieves remarkable performance and at the same time maintains a reasonable level of robustness in a fully decentralized and distributed manner. The high performance is accomplished by **decreasing the number of validators**. The security is achieved by the **anonymous leaders-proposers' mapping** and the **random validators' selection**. We set the *risk* value in TrueBFT so the 'number' of the

employed validators would be $\log n$ nodes. This is only to show that even if we select a relatively small number of validators, TrueBFT 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 4.4.2.1, the number of validators in TrueBFT is dynamic and changes based on the proposing node’s risk likelihood, as will be covered in Chapter 5.

4.5.1 Experiments Setup

In our experiments, we used Omnet++ as our simulation platform. OMNeT++ is a C++-based discrete event simulator for modeling communication networks [95]. It has gained wide-spread popularity in peer-to-peer (p2p) protocols simulations [7], 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 [97]. Other advantages of OMNET++ are that it is well structured, highly modular, *not limited to network protocols simulation*, and source code is publicly available [72]. Nevertheless, OMNET++ doesn’t have a blockchain-based platform; however, we had to use it due to the lack of blockchain-enabled simulators and its popularity in modeling p2p networks.

We simulated the Block-Supply Chain as a p2p network. The two protocols (Tendermint and TrueBFT) were exactly simulated as described before (Sections 4.3.3.2 and 4.4). The only thing that we had to introduce was 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. 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. Put differently, when a node in our simulated network receives a *control* message, it becomes a proposing node. 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*.

4.5.2 Performance

We have conducted several experiments to examine TrueBFT performance with different numbers of nodes. The question 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 a different consensus protocol (i.e., one uses TrueBFT, and another uses Tendermint), having the same number of nodes, under the same network conditions, which protocol will outperform the another?"

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 true decentralization, and optimal security is by involving $n - 1$ nodes in the validation process. Besides, our goal was to compare TrueBFT's security to an optimally secured protocol.

We chose networks of sizes 100, 125, 150, 175, 200 nodes respectively to investigate

Figure 4.4: Consensus latency for committing one block.

the scalability of the protocols when the number of nodes increases. We evaluated two performance metrics:

1. **Latency:** Measured as the time taken to commit one proposed block. Figure 4.4 shows that TrueBFT 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 the scalability of each protocol, 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 4.5 illustrates the comparison between

Figure 4.5: Scalability: TrueBFT’s consensus latency with $\log n$ validators increases gradually, while Tendermint’s time with $n - 1$ validators increases significantly on networks of sizes between 100 - 200 nodes.

our $\log n$ validators protocol and the $n - 1$ validators Tendermint.

Despite the overhead that we introduced by the random validators’ selection, TrueBFT outperforms Tendermint and shows great scalability. After analyzing the simulation results, we found that two factors influence a protocol’s performance:

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

4.5.3 Security

As mentioned in Section 4.4.1, 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 selection approach protects against the following attacks:

4.5.3.1 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 [38]. Leaders' and validators' replaceability and randomizing their selection can significantly mitigate this attack. This is because 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 attacking all the nodes in the network to undermine the system. Similarly, attacking the validation-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-committers). Note that we aim only to protect the validation and consensus process from DDoS attacks.

4.5.3.2 Eclipse Attacks

This attack is presented by Heilman et al. [33] 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 [33]. As in the DDoS attack, an adversary mounting this attack needs to know the identity of the node participating in the validation and consensus process in advance. Introducing variable, anonymous leaders and validators on each block's proposal makes the adversary's job more difficult.

4.5.3.3 Bribery Attacks

Bonneau et al. [8] introduced a new attack on blockchain consensus, the so-called Bribery attack. In this attack, a malicious node deliberately pays miners/validators to work on specific blocks and forks. The goal of such an attack is to generate an arbitrary fork that benefits this malicious node [8]. The attack generally works by the adversary offering the miners/validators 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. TrueBFT anonymizes the interaction between the consensus and validation parties, which significantly mitigates such an attack.

4.5.3.4 Hijack and Wait

One possible attack is for an adversary to hijack a number of nodes (sufficient to undermine the consensus process), and wait for them to be selected to exploit. For example, an adversary can hijack $\frac{\log n}{3}$ nodes and wait for them to be selected in one of the validation rounds (we assume here that the adversary is very powerful to hijack such a number). In this case, $\frac{\log n}{3}$ is sufficient to undermine the consensus process, since the protocol only tolerates $< \frac{\log n}{3}$ faulty nodes. This attack is rare-occurrence, but feasible. TrueBFT does not provide a means to prevent such an attack, but it

Figure 4.6: Event of Interest, where A is the set of n elements, B is the subset of $\log n$ elements, and C is the subset of $\frac{\log n}{3}$ elements.

greatly mitigates it. Below we give a probabilistic analysis to illustrate how such an event is rare to occur.

Let A be the set of all n nodes in the blockchain network. Let B be the subset of the random, selected $\log n$ validators responsible for driving the consensus process. Assume that an adversary was able to hijack the smallest number of nodes enough to undermine the consensus. That is, a subset C of $\frac{\log n}{3}$ nodes. Our goal is to find the probability that C will lie in B .

To calculate this probability, first consider the Event of Interest in the Venn diagram in Figure 4.6, where all elements of C (i.e., $\frac{\log n}{3}$) lie in B . In other words, $\frac{\log n}{3}$ elements are common for the B and C . Intuitively, this common region is also obtained from the n elements set A .

For simplicity and for illustration purpose, we will assume a uniform distribution, and the randomly selected nodes are selected and allocated to each subset without repeating elements. Having that said, let:

$$\frac{\log n}{3} = j, \text{ and } \log n = k$$

The probability p that all elements of C will lie in B (i.e., successful attack) is:

$$p = \frac{\binom{n-j}{k-j}}{\binom{n}{k}} \quad (4.4)$$

In Equation 4.4, the set A of n elements is fixed. There are $\binom{n-j}{k-j}$ ways of selecting the subset C , and $\binom{n}{k}$ ways of selecting the subset B . The probability p shows that this event is rare to occur. For example, with a network of size 150 nodes (i.e., set A), the probability that the hijacked $\frac{\log n}{3}$ nodes (i.e., subset C) will be in the chosen set of $\log n$ validators (i.e., subset B) is:

$$p \approx 0.0018$$

Which can translate to if we run the experiment (i.e., proposing new blocks and validating them) for 555 times this attack might occur once.

4.5.3.5 Experimental Evaluation

Proving a consensus protocol security experimentally requires examining all the possible strategies for an adversary, which is infeasible [26]. However, for illustration purposes, we chose 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.

The question that we were aiming to answer here is: "if we apply the same attack on two blockchain networks, each one of them uses a different consensus protocol (i.e., one uses TrueBFT, and another uses the optimally secured Tendermint), having the same number of malicious nodes under the same network conditions, which protocol

will achieve the higher security?” We modeled the attacks and evaluated the protocols as follows:

- We randomly selected a 0.33% fraction of the nodes to be malicious from each network. The reason for this choice is that Tendermint and TrueBFT tolerate up to one-third of Byzantine nodes, which is in this case malicious. To evaluate a critical aspect such as 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.
- We assumed without loss of generality that these malicious nodes are bribable (corruptible). On the other hand, the remaining nodes are honest and would not accept a malicious bribe.
- Our security metric was 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 selected the 0.33% random malicious nodes for each 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 (optimal security).
 - TrueBFT without validation-leaders (i.e., fixed-validators).
 - TrueBFT with validation-leaders, but *without* anonymous proposer-leaders’ mapping.
 - TrueBFT with validation-leaders and anonymous mapping.

The reason that we selected these four types of protocols is that we were aiming to illustrate how our true decentralized TrueBFT protocol evolved and to show the importance of applying anonymity and the validators' replaceability to TrueBFT.

- 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 [74].
 - 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. Note that if the goal of the attack is to stall or undermine the liveness of the protocol, one-third of validators is sufficient to bribe.
 - TrueBFT *without* anonymous proposer-leaders' mapping: the malicious proposer would try to bribe a *chosen set* of $\log n$ nodes over time. Then, the proposer bribes its corresponding *known* leaders to convince them to select these nodes as validators (note that we are not applying the anonymous proposer-leaders' mapping).
 - TrueBFT *with* anonymous proposer-leaders' mapping: the malicious proposer bribes a *chosen set* of $\log n$ nodes over time. However, in TrueBFT, the proposer does not know its corresponding leaders in advance, as they are anonymous. Hence, it can only try to blindly select a set of nodes hoping they are its leaders and bribe them. We call this set a 'lottery pick.'

Figure 4.7: Detection Rate (DR) with 95% Confidence Intervals.

Figure 4.7 shows the Detection Rate (DR) with 95% Confidence Intervals (CI). This figure nearly offers a clear vision of how the true decentralization (anonymous proposers-leaders' mapping and random validators' selection) contributes to the consensus protocol security. TrueBFT with the anonymous proposers-leaders' mapping achieves high attack detection rates, ranging from 99.5% when the network size is 100 nodes to 99.8% with 200 nodes on the network. Note we only examined one attack (Bribery), but we argue that true decentralization withstands many other attacks that require knowing the identities of the attacked nodes such as DDoS and Eclipse.

4.6 Limitations and Concluding Remarks

We have proposed a new truly decentralized consensus protocol utilizing anonymous mapping and randomness. TrueBFT randomly employs a different set of anonymous validators each time a new block is proposed to protect against several real attacks mounted by powerful adversaries. Our simulations show that TrueBFT is very scalable for large networks when utilizing a relatively small number of validators. At the same time, it maintains a satisfactory level of security.

Despite the security that true decentralization offers, attacks that resulted from form dishonest behavior such as *block withholding* and *denial of service* cannot be mitigated, as we will discuss in Chapter 5. To overcome this, we in Chapter 5, integrate a game theoretical model into TrueBFT to incentivize honest behavior and penalize the dishonest one. Besides, as we have seen in Section 4.5.3.5, knowing the number of validators (the size of the validators' committee) might provide useful knowledge for powerful adversaries. As a result, in our game theoretical-based TrueBFT (Chapter 5), the number of validators is not static and is changeable based on the risk likelihood of every proposer. In addition, making the number of validators proportional to the likelihood of proposers contributes towards efficiency.

One possible threat that we are not providing a solution for in this dissertation is *the reputation boosting and dumping attack*, in which some nodes may behave well for a long time to gain good reputation values, but begin to be malicious in a time point. That may be a serious problem for the blockchain then.

5 Game Theoretical Consensus Protocol

This chapter discusses the problem of the dishonest behavior of the consensus parties (i.e., validation-leaders and validators) and the attacks resulting from this behavior. Additionally, this chapter addresses the issue of the number of selected validators. This chapter also presents in detail our fourth contribution (a game theoretical model integrated into our TrueBFT protocol) to overcome these issues.

5.1 Introduction

Blockchain technology was introduced to be *secure by design* and resistant to modification [9]. Despite its potential to elevate security, as with all new technologies, security risks can be found beneath the hype [85]. Moreover, blockchain technology has introduced new kinds of attacks such as *block withholding* and *selfish mining*. Such attacks occur for various incentives, mostly financial. To defend against such attacks and to strengthen blockchain security, *game theory* stands out as a potentially powerful means.

The consensus protocol that we proposed in Chapter 4 enjoys true decentralization, which makes it immune to attacks that require knowing the victims' identities before attacking them (e.g., DDoS, Eclipse, and Bribery attacks). Nevertheless, it does not consider the following two issues:

First, true decentralization does not guarantee that the validators are always honest and do not deviate from the protocol. For example, a dishonest validator might

perform a *block withholding* attack [78] in which the validator does not participate in the validation process, or does not reveal the results of the verification in favor of a malicious proposing node. Such an attack can result in undermining the consensus process. Another example is the *denial of service* attack [18], where an omission fault occurs due to: (a) a leader avoiding selecting validators as instructed by the protocol, or (b) a validator avoiding the validation process or broadcasting a vote. To overcome such vulnerability, we integrate a game theoretical model into our consensus protocol that rewards honest consensus participants and punishes dishonest or lazy ones that do not adhere to the protocol.

The second problem with most of the current consensus protocols (including TrueBFT) is that the number of validators is fixed, despite the variation in hostilities in blockchain environments. As we observed from our simulation results (Section 4.5.2), the number of validators in a blockchain network influences its security and efficiency substantially. In response to this, our game theoretical model enables our consensus protocol to select a different size of the validators' committee for each block proposal. This size (i.e., the number of validators) is proportional to the *risk likelihood* of the proposing node. The number of validators is based on the outcome of a game played between the proposing node and its validation-leaders.

In addition to overcoming the above issues, TrueBFT utilizes the proposed game theoretical model to enable validators to validate with some probability proportional to the proposers' risk likelihoods instead of the always-validation (i.e., validators always validate, even if the risk likelihood of a block's proposer is low). The always-validation might be unnecessary computational work particularly found in blockchains with low hostility.

In this chapter, we study the incentives of malicious nodes to deviate from the consensus protocols, and we apply a game theoretical model to reinforce honest be-

havior. Also, we address the problem of validators' selection in terms of *how many validators to select* to achieve a satisfactory trade-off between security and efficiency.

Our proposed game theoretical model is a *two-stage attacker-defender Bayesian game* integrated into TrueBFT. The first-stage game takes place between the proposing node as a potential attacker and its corresponding validation-leaders. The outcomes of this game determine the proposer's risk and, hence, the number of validators to select. The proposer and the selected validators then play the second-stage game in which the validators decide whether or not to validate. In both games, our proposed model reinforces honest behavior by rewarding honest parties and penalizing dishonest ones. Our proposed two-stage game theoretical model has the following advantages:

1. It reinforces the honest behavior of the consensus participants by rewarding honest ones and penalizing dishonest ones.
2. The *number of the selected validators* is dynamic and variable. Hence, instead of selecting a fixed number of validators, TrueBFT utilizes the outcomes of the theoretical games to select a different number of validators on every block proposal proportional to the *risk likelihood* of the proposing node.
3. It eliminates the *always-validation* mode. Instead, the validators validate with probability proportional to the the proposing node *risk*.

5.2 Background

In economics, game theory is the study of mathematical models of strategic interaction among players (decision-makers) who are known to be rational [77]. Game theory has been widely used for modeling conflict situations and predicting the participants' behavior [57]. It has been applied to many applications in social science,

military, logic and computer science [77]. In computer systems, Intrusion Detection Systems (IDSs) that use game theory proved to outperform the always-on traditional IDSs in terms of throughput [54], especially in resource-constrained networks. In blockchain applications, the use of game theory can elevate security. For example, we have seen in Bribery attacks that the fees associated with a proposed block are not always sufficient to behave honestly. Game theory in such cases can be used as a rewarding and punishing mechanism to further reinforce playing honestly and as expected. Moreover, game theory can be utilized to analyze how risky a player is, which is a useful piece of information in blockchain-based applications.

5.2.1 Definition

A theoretical game G consists of the three following elements [52, 89]:

- **Players (P):** The entities participate in the game. They could be people, institutions, or anything that can interact.
- **Strategies (S):** Each player has a set of strategies, in which he/she can choose from. Each strategy represents an action (move). Strategies can be pure (i.e., single move) or mixed.
- **Payoff functions (U):** When each player in the game has played his/her strategy, the player gets a negative or a positive payoff determined by his/her payoff function $u \in U$.

Formally, G is defined as a triplet (P, S, U) [6]. The payoff $u_i(s)$ is the benefit b of player i minus the cost c for plying the strategy s , and it is expressed as: $u = b - c$.

Based on the assumption that every player in the game is rational, the players will select the strategies that maximize their payoff and minimize their cost (i.e.,

the negative payoff) [52]. This leads to the concept of **Equilibrium** for the game. Equilibrium is considered to be the solution to the game, as we will discuss shortly.

5.2.2 Classification of Game Theories

Theoretical games have many aspects [89]. The classifications of games based on their aspects are as follows [89]:

1. **Number of stages:** A game can be played in one stage or multiple stages as follows:
 - *Static/Strategic Game:* This is a one stage game in which players take actions (moves) simultaneously at the same time without knowing the strategies chosen by other players.
 - *Dynamic/Extensive Game:* This game consists of multiple stages. In such games, the players can observe their own or others' previous moves (i.e., perfect information). Hence, they can benefit from them at different stages of the game. The number of stages can be finite or infinite [25].
2. **Perfect or imperfect information:** Based on this category, a game can be classified into:
 - *Perfect Information Game:* In this type, a player knows all the previous actions/moves of the other players. Hence, the player selects his/her current move based on this knowledge.
 - *Imperfect Information Game:* Here, one or more players do not know all of the previous moves of the other players.
3. **Complete or incomplete information:** We can classify a game based on the information a player has as:

- *Complete Information Game*: In this type of game, each player knows the payoff function and current strategy profile of every other player in the game.
- *Incomplete Information Game*: One or more players do not know all players' payoff functions and their current strategy profiles.

5.2.3 Bayesian Game

A Bayesian game is a game with incomplete information, and it introduces a new component to the game G [89]. In such games, we use the term *type* to capture the incomplete information. That is, in addition to the set of players P , the set of strategies S , and the set of payoff functions U , the Bayesian game adds a set of players types Φ . For each player i , this player can be of type $\theta_i \in \Phi_i$. The Bayesian game is very suitable for the attacker-defender scenario. This is because the defender player has incomplete information about the type of the other player, which could be of two types: a) *malicious*, or b) *regular*.

5.2.4 Nash Equilibrium (NE)

The combination of the players' strategies in which each player's strategy is the best response with respect to other players' strategies is called an **Equilibrium** for the game G [52]. The *best response* is the selected strategy that results in the maximum payoff of a player, given other players' strategies.

Formally, assuming there are n players, the strategy profile $s = \{s_i\}_{i=1}^n$ is the n -tuple of the players' strategies [57]. As mentioned above, player i has a set of strategies which he/she can execute (denoted by S_i). $br_i(s_{-i})$ is the best response function for player i to the remaining players' strategies s_{-i} . $br_i(s_{-i})$ is the function that maximizes $u_i(s_i, s_{-i})$, that is player i 's payoff over his/her set of strategies S_i .

with regard to other players' strategies s_{-i} formulated as below:

$$br_i(s_{-i}) = \underset{s_i}{argmax} \quad u_i(s_i, s_{-i}) \quad (5.1)$$

If a strategy profile s^* satisfies $s_i = br_i(s_{-i})$ for every player i , then no player has the incentive to deviate from s^* . This is because each player has his/her best response function satisfied in terms of the payoff. We can say that the strategy profile s^* is in **NE** [64], which can be expressed as following, for each player i :

$$u_i(s_i^*, s_{-i}^*) \geq u_i(s_i, s_{-i}^*), \forall s_i \in S_i \quad (5.2)$$

5.2.5 Mixed Strategies

The previous explained strategy profile s^* is called *pure strategies*. However, a player can randomize (or mix) with some probability between his/her pure strategies, especially when there is no *pure strategies NE* for the game. This is called *mixed strategy*. In other words, for a player i , the mixed strategy x_i is a probability distribution over S_i (i.e., his/her pure strategies set) [57]. A mixed strategy profile $x^* = \{x_i^*\}_{i=1}^n$ is considered a *mixed-strategy NE* solution for the game if,

$$\bar{u}_i(x_i^*, x_{-i}^*) \geq \bar{u}_i(x_i, x_{-i}^*), \forall x_i \in X_i \quad (5.3)$$

Where $\bar{u}_i$ is the expected payoff function for player i , X_i is the set of distributions over S_i (i.e., the set of pure strategies), and x_{-i} is a set of mixed strategies of the other players (i.e., excluding player i).

5.3 Related Work

Although blockchain technology has gained considerable attention from the computer science and economics communities, the use of game theory methods in this

technology is limited [27, 70]. In this section, we present the most relevant and recent works that utilize game theory in blockchain technology.

Xu et al. [98] proposed a game theoretical approach to suppress the attack motivation on a blockchain that consists of mobile devices and edge servers. The game is formulated between a mobile device and an edge server, where the mobile device can send a request to the server to acquire a real-time service or launch an attack. On the other hand, the server chooses to either provide the service or to attack the mobile device. The authors introduced a punishment mechanism according to the action record to mitigate the attacks on the blockchain. They have concluded that both players tend to behave well when the punishment weight is significant. The proposed approach was designed to deal with attacks like zero-day, DDoS, and password-based attacks.

Kiayias et al. [40] exploited game theory to study the miners' behavior in the Bitcoin protocol. They studied the incentive of selfish miners, in which a miner decides not only which block to mine, but also when to release the mined block to other miners. In this work, two games were introduced. The first one is the immediate-release game in which the miner's strategy is to select an appropriate block to mine. In this game, every miner releases the mined blocks immediately. The second game is the strategic-release game, where the miners have the choice to withhold releasing the mined blocks in addition to selecting which blocks to mine. In both games, the authors showed that when the computational power of a miner is small, its best response matches the expected behavior. Nevertheless, when the computational power is large, the miner deviates from the expected behavior.

Johnson et al. [38] employed a game theoretical model to analyze the incentives for a mining pool to launch a DDoS attack against another mining pool. The players in the game are two competing mining pools, where each one may utilize additional

computing resources to increase the chance of winning the mining race, or to trigger a DDoS attack to lower the expected success of the other competing mining pool.

Luu et al. [55] studied the block withholding attack on mining pools using a game theoretical approach by formulating the Bitcoin mining as a game. They analyzed the block withholding attack and concluded that the attack is profitable and well-incentivized in the long-term. The authors derived the game equilibrium state, which is a mixed strategy where all clients are incentivized to attack rather than participate honestly to maximize their payoffs. Finally, the authors concluded that the PoW protocol is vulnerable to such an attack.

In a paper entitled 'The Miner's Dilemma,' Eyal [23] studied the scenario when pools attack each other. Open pools (i.e., pools of miners that allow any miner to join the mining work) are vulnerable to block withholding attacks performed by infiltrated miners from competing pools. This paper defined a game where pools recruit some of their participants to infiltrate other pools to diminish their mining capabilities. This game is called the miner's dilemma where players are two pools, and their strategies are whether or not to attack each other. The author observed that attacking is the dominant strategy for each player.

All the above works have introduced game theoretical approaches to the PoW mining protocol. As previously discussed in Section 4.1, PoW is not an attractive approach for blockchains that are efficiency-sensitive due to its massive computation demands. In a more relevant work presented by Kiayias et al. [41], the Ouroboros consensus protocol was proposed. Similar to TrueBFT, Ouroboros eliminates the need for an energy-hungry PoW protocol. Ouroboros is based on the Proof of Stake (PoS) mechanism. It works by dividing the time into rounds called slots in which each slot is assigned to a leader. The leaders are picked based on the stake they have. A chosen leader is responsible for producing a block for its time slot. The authors

utilized game theory to introduce a reward mechanism to incentivize the participants in the blockchain. By means of the game theoretical design, attacks such as selfish-mining and block withholding are mitigated. The rewarding mechanism works by awarding a positive payoff to participants who do not diverge from the protocol.

5.4 The Proposed Two-Stage Bayesian Game Model

In Chapter 4, we proposed a new consensus protocol that exploits randomness and anonymous proposers-leaders mapping to achieve the true decentralization security. The protocol works in two phases: the initialization phase, which is executed once by the blockchain initiator, and the verification phase, which is executed by the blockchain nodes every time a new block is proposed.

In this chapter, we propose a two-stage game theoretical model and integrate it into TrueBFT. The two games (i.e., first-stage and second-stage) are played in the verification phase of the protocol, each time a new block is proposed. The two games are modeled as attacker-defender games. The proposer (player x) is the potential attacker in both games. The defenders (player y) in the first-stage are the validation-leaders. The defenders in the second-stage are the validators (player z) that have been selected by the validation-leaders from the first-stage. The next two subsections present an in-depth description of the proposed model.

5.4.1 First-Stage Game

As mentioned in Section 4.4.2, 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. After a node finds that it is a leader for the proposer, this leader plays the first-stage game with the proposer to decide how many validators (m) to select.

5.4.1.1 Game Model

This game takes place between the proposer (i.e., player x) and each of its validation-leaders (i.e., player y). The validation-leader determines the number of validators based on the outcome of the game. There are two strategies for the validation-leader (i.e., S_y), from which to choose. The first one is to *UseMinimumValidators* where the minimum is four validators. The second strategy is to *AddMoreValidators* where the number of validators varies based on the outcome of the game, which is proportional to the risk likelihood of the proposer. The strategy profile for the proposer (S_x) is a) *Cheat* in which the proposer broadcasts an invalid block, and b) *NotCheat*. A proposer could be of two types: *malicious* or *regular*.

Our game is considered to be a one-to-four game, where one proposer plays with four leaders. Each of the four leaders has no cooperation with the other leaders, so we consider each game between a leader and the proposer as an independent event. Since the validation-leader does not know the type of player x (i.e., regular or malicious), we model our game as a Bayesian game. This is because the leader node (player y) in our model has incomplete information about the game. Player x , however, has this private information about its type known only to it.

5.4.1.2 Strategic Form of First-Stage Bayesian Game

First, we model our game as a strategic form as shown in table 5.2 and 5.3. Table 5.1 shows the notation of the first-stage game. Table 5.2 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 5.3 shows the payoff matrix of the game when player x is of type *regular*. The goal of both players x and y is to maximize their payoffs. We assume that the players are rational.

Table 5.1: The first-stage 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 our Block-Supply Chain, a node that supplies products (and proposes blocks for these products) to 10 other nodes is more critical 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 certain threshold by playing <i>UseMinimumValidators</i> . However, player y can lose γ (i.e., deducted from its gain g_y) if it plays <i>AddMoreValidators</i> 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 <i>Cheat</i> and player y plays <i>UseMinimumValidators</i>).
w_x	Work done by the proposing node (player x) to play <i>Cheat</i> .
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 <i>AddMoreValidators</i> .
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 it 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 5.2: Strategic form of the first-stage Bayesian game (player x is malicious)

Game Matrix		Player y (Validation-leader)	
		<i>AddMoreValidators</i>	<i>UseMinimumValidators</i>
Player x	<i>Cheat</i>	$(\beta \cdot c_x) - w_x, [(\beta \cdot g_y) - \gamma] - w_y$	$(\beta \cdot g_x) - w_x, \beta \cdot c_y$
	<i>NotCheat</i>	$0, -w_y - \gamma$	$0, \gamma$

Table 5.3: Strategic form of the first-stage Bayesian game (player x is regular)

Game Matrix		Player y (Validation-leader)	
		<i>AddMoreValidators</i>	<i>UseMinimumValidators</i>
Player x	<i>NotCheat</i>	$0, -w_y - \gamma$	$0, \gamma$

Figure 5.1: Extensive form of the first-stage Bayesian game.

5.4.1.3 Extensive Form of First-Stage Bayesian game

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 5.1 represents the Bayesian game extensive form. μ can be assigned according to the environment of the network. A higher value of μ is given when the environment is hostile.

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.

5.4.1.4 Bayesian Nash Equilibrium (BNE) Analysis

A. Game Pure-Strategy BNE: In this section, we analyze BNE assuming that player x knows player y 's belief of μ . If player x plays its pure strategy (*Cheat* if

malicious, *NotCheat* if regular), then, based on our game in Figure 5.1, the expected payoff of player y playing its pure strategy *AddMoreValidators* is:

$$E_{\mu_y}(\textit{AddMoreValidators}) = \{\mu \cdot [(\beta \cdot g_y) - \gamma] - w_y\} + \{(1 - \mu) \cdot (-w_y - \gamma)\}$$

Similarly, the expected payoff of player y playing its pure strategy *UseMinimumValidators* is:

$$E_{\mu_y}(\textit{UseMinimumValidators}) = [\mu \cdot (\beta \cdot c_y)] + [(1 - \mu) \cdot \gamma]$$

So, if $E_{\mu_y}(\textit{AddMoreValidators}) > E_{\mu_y}(\textit{UseMinimumValidators})$ Or,

$$\{\mu \cdot [(\beta \cdot g_y) - \gamma] - w_y\} + \{(1 - \mu) \cdot (-w_y - \gamma)\} > [\mu \cdot (\beta \cdot c_y)] + [(1 - \mu) \cdot \gamma]$$

Which can be simplified to:

$$\mu > \frac{w_y + 2\gamma}{\beta(g_y - c_y) + \gamma} \quad (5.4)$$

Then, the best response of player y is to play *AddMoreValidators*. Nevertheless, if player y chooses to play *AddMoreValidators*, *Cheat* will no longer is the best response for player x type *malicious* and, instead, will choose to play *NotCheat*. As a result, ((*Cheat* if malicious, *NotCheat* if regular), *AddMoreValidators*, μ) is not a Bayesian Nash Equilibrium (**BNE**). However, if $E_{\mu_y}(\textit{AddMoreValidators}) < E_{\mu_y}(\textit{UseMinimumValidators})$ or,

$$\mu < \frac{w_y + 2\gamma}{\beta(g_y - c_y) + \gamma} \quad (5.5)$$

Then, the best response for player y is to play *UseMinimumValidators* and thus ((*Cheat* if malicious, *NotCheat* if regular), *UseMinimumValidators*, μ) is a *pure-strategy BNE*.

If player x type *malicious* chooses to play the pure strategy *NotCheat*, player y 's dominant strategy is *UseMinimumValidators*, regardless of μ . Nevertheless, if

player y plays *UseMinimumValidators*, the best response for player x type *malicious* is *Cheat*, which reduces to the above case. Hence, $((\text{NotCheat if Malicious, NotCheat if Regular}), \text{UseMinimumValidators})$ is not a **BNE**.

B. Game Mixed-Strategy BNE: We previously showed that when equation 5.4 is true, there is no *pure-strategy BNE*. So, we have to find a *mixed-strategy BNE*. Let p be the probability that player x plays *Cheat*. Let q be the probability that player y plays *AddMoreValidators*. The expected payoff of player y playing the strategy *AddMoreValidators* is:

$$\begin{aligned} E\mu_y(\text{AddMoreValidators}) &= \{p \cdot \mu \cdot [(\beta \cdot g_y) - \gamma] - w_y\} + \{(1 - p) \cdot \mu \cdot (-w_y - \gamma)\} \\ &\quad + \{(1 - \mu) \cdot (-w_y - \gamma)\} \end{aligned}$$

The expected payoff of y playing *UseMinimumValidators* is:

$$E\mu_y(\text{UseMinimumValidators}) = \{p \cdot \mu \cdot (\beta \cdot c_y)\} + \{(1 - p) \cdot \mu \cdot \gamma\} + \{(1 - \mu) \cdot \gamma\}$$

So, player y plays *AddMoreValidators*, if $E\mu_y(\text{AddMoreValidators}) > E\mu_y(\text{UseMinimumValidators})$. Or,

$$p > \frac{w_y + 2\gamma}{\mu\beta(g_y - c_y) + \mu\gamma} \quad (5.6)$$

Similarly, we calculate the expected payoffs for player x . The expected payoff of x playing *Cheat* is:

$$E\mu_x(\text{Cheat}) = \{q \cdot \mu \cdot [(\beta \cdot c_x) - w_x]\} + \{(1 - q) \cdot \mu \cdot [(\beta \cdot g_x) - w_x]\}$$

The expected payoff of x playing *NotCheat* is:

$$E\mu_x(\text{NotCheat}) = 0$$

As a result, player x plays *Cheat*, if $E\mu_x(\text{Cheat}) > E\mu_x(\text{NotCheat})$, or:

$$q > \frac{w_x - (\beta g_x)}{\mu\beta(g_y - c_y)} \quad (5.7)$$

Now, we derive our game's mixed-strategy **BNE** as: $((q \text{ if malicious, } \textit{NotCheats}$ if regular), $p, \mu)$. Expressed in words, player x mixes/plays with probability q if it is malicious, player x always plays *NotCheats* if it is regular, player y always mixes/plays with probability p , and the nature node (N) decides the type of player x with probability μ .

Thus far, we have obtained the above game's mixed-strategy **BNE**. However, this game is molded for one player x and one player y , and we have four defenders (validation-leaders) and player x knows this fact. Hence, $((q \text{ if malicious, } \textit{NotCheats}$ if regular), $p, \mu)$ is no longer a valid *mixed-strategy* **BNE**. Thus, we will derive a new *mixed-strategy* **BNE**. The events of validations are independent. We have four validation-leaders. Therefore, the likelihood that the four validators plays the strategy *AddMoreValidators* is $\hat{p}$ and is calculated as:

$$\hat{p} = (4.p) - p^4 \quad (5.8)$$

Where p is the probability that one validation-leader plays *AddMoreValidators*. Now, the attacker plays *Cheats* with probability $\hat{q}$ defined as:

$$\hat{q} = q - (\hat{p} - p) \quad (5.9)$$

So, our new *mixed-strategy* **BNE** is: $((\hat{q} \text{ if malicious, } \textit{NotCheats}$ if regular), $p, \mu)$.

5.4.1.5 Deciding the Number of Validators (M)

After executing the first-stage game, each validation-leader decides its number of validators (m), of which $m < n$ where n is the total number of nodes in the network. The m value can be: a) four validators if the validation-leader chooses to play *UseMinimumValidators*, or b) a fraction of $\frac{n}{4}$ proportional to p if it plays *AddMoreValidators* (we choose $\frac{n}{4}$ because we have four validation-leaders).

p is the probability that the proposing node (player x) might attack (plays *Cheat*). Hence, we consider p as the "risk likelihood" of an attack. p is computed with the assumption that the validation-leader is 'risk-neutral'; that is, in a fair game each player aims to maximize its expected payoff. Note that p replaces the *risk* that we introduced in our initial consensus protocol presented in Chapter 4. The difference here is that each proposing node has a different p , where in the formal design, *risk* is set to be the risk likelihood for all proposers in the network.

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 . In other words, a validation-leader selects a random number between 5 (the minimum number of validators plus one) and $\frac{p \cdot (n-2)}{4}$ (excluding the proposing and the validation-leader nodes) as follows:

$$m \leftarrow \text{Random}[5, \frac{p \cdot (n-2)}{4}]$$

After a validation-leader decides its m , the protocol proceeds as pre-described in Section 4.4.2.2. Algorithm 7 is the new version of Algorithm 6 that we introduced in Section 4.4.2.2.

The objectives of the first-stage game are:

1. Incentivize the good behavior for both players by rewarding the honest behavior and penalizing the dishonest one.
2. Determine the number of validators (i.e., m) that a leader has to choose. The aggregated number of validators (i.e., M) selected by the four leaders is proportional to the risk likelihood of the proposer (i.e., p). This, as discussed before, influences both the security and performance of the consensus protocol.

Algorithm 7: Validators' selection

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

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

```

1 if AddMoreValidators then
2   |  $m \leftarrow \text{Random}[5, \frac{p \cdot (n-2)}{4}]$ 
3 else
4   |  $m \leftarrow 4$ 
5 end
6 for  $k \leftarrow 1$  to  $m$  do
7   |  $p_i(k) \leftarrow \frac{R_i}{\sum_{v_j \in V - PV} R_j}$ 
8   | Randomly select  $v_i$  with probability  $p_i(k)$  from  $V - PV$ 
9   |  $PV.add(v_i)$ 
10 end
11 for  $l \leftarrow 1$  to  $m_i$  do
12   |  $p_i(l) \leftarrow \frac{R_i}{\sum_{c_j \in V - PC} R_j}$ 
13   | Randomly select  $c_i$  with probability  $p_i(l)$  from  $V - PC$ 
14   |  $PC.add(c_i)$ 
15 end
16 Return  $PV$  AND  $PC$ 

```

5.4.2 Second-Stage Game

After selecting the validators by their leaders and after proposing and broadcasting the new block by the proposer, the second-stage game takes place between the proposer (player x) and each of the validators (player z). The strategy profile for a validator is a) *Validate*, and b) *NotValidate*. This game is modeled similarly to the first-stage game. Tables 5.4 and 5.5 show the strategic form of the second-stage Bayesian game. Figure 5.2 shows the extensive form of the second-stage game. We use the same notations presented in Table 5.1 with following additional notations:

- w_z is the work done by the validator (player z) to play *Validate*.
- g_z is the gain for player z from capturing a cheater.
- c_z is the cost for player z if it fails to capture a cheater.

Table 5.4: Strategic form of the second-stage Bayesian game (player x is malicious)

Game Matrix		Player z (Validator)	
		<i>Validate</i>	<i>NotValidate</i>
player x	<i>Cheat</i>	$(\beta.c_x) - w_x, (\beta.g_z) - w_z$	$(\beta.g_x) - w_x, \beta.c_z$
	<i>NotCheat</i>	$0, -w_z$	$0, 0$

Table 5.5: Strategic form of the second-stage Bayesian game (Player x is regular)

Game Matrix		Player z (Validator)	
		<i>Validate</i>	<i>NotValidate</i>
Player x	<i>NotCheat</i>	$0, -w_z$	$0, 0$

Figure 5.2: Extensive form of the second-stage Bayesian game.

A. Game Pure-Strategy BNE: We follow a similar analysis that we presented in the first-stage game. If player x plays its pure strategy (*Cheat* if malicious, *NotCheat* if regular), then, the expected payoff of player z playing its pure strategy *Validate* is:

$$E\mu_z(\text{Validate}) = \{\mu.[(\beta.g_z) - w_z]\} + \{(1 - \mu). -w_z\}$$

The expected payoff of player z playing its pure strategy *NotValidate* is:

$$E\mu_z(\text{NotValidate}) = \mu.(\beta.c_z)$$

As a result if, $E\mu_z(Validate) > E\mu_z(NotValidate)$ Or,

$$\mu > \frac{w_z}{\beta(g_z - c_z)} \quad (5.10)$$

Then, the best response of player z is to play *Validate*. Therefore, ((*Cheat* if malicious, *NotCheat* if regular), *Validate*, μ) is not a **BNE** because *Cheat* is not the best response for player x type *malicious*. However, if $E\mu_z(Validate) < E\mu_z(NotValidate)$ or,

$$\mu < \frac{w_z}{\beta(g_z - c_z)} \quad (5.11)$$

Then, the best response for player z is to play *NotValidate* and thus ((*Cheat* if malicious, *NotCheat* if regular), *NotValidate*, μ) is a *pure-strategy* **BNE**. Nevertheless, similar to the first-stage game ((*NotCheat* if *Malicious*, *NotCheat* if *Regular*), *NotValidate*) is not a **BNE**.

B. Game Mixed-Strategy BNE: Let p' be the probability that player x plays *Cheat*. Let q' be the probability that player z plays *Validate*. The expected payoff of z playing *Validate* is:

$$E\mu_z(Validate) = \{p' \cdot \mu \cdot [(\beta \cdot g_z) - w_z]\} + \{(1 - p') \cdot \mu \cdot -w_z\} + \{(1 - \mu) \cdot -w_z\}$$

The expected payoff of z playing *NotValidate* is:

$$E\mu_z(NotValidate) = p' \cdot \mu \cdot (\beta \cdot c_z)$$

So, the defender (player z) plays *Validate* when:

$$p' > \frac{w_z}{\mu\beta(g_z - c_z)} \quad (5.12)$$

Similarly, we acquire the expected payoffs the attacker (player x). The expected payoff of x playing *Cheat* is:

$$E\mu_x(Cheat) = \{q' \cdot \mu \cdot [(\beta \cdot c_x) - w_x]\} + \{(1 - q') \cdot \mu \cdot [(\beta \cdot g_x) - w_x]\}$$

The expected payoff of x playing *NotCheat* is:

$$E\mu_x(NotCheat) = 0$$

As a result, the attacker (player x) plays *Cheat* when:

$$q' > \frac{w_x - (\beta g_x)}{\mu \beta (g_z - c_z)} \quad (5.13)$$

As in the first-stage game, this game is 1-to- M game, where M is the number of validators (player z). Hence, our new *mixed-strategy BNE* is: $((q''$ if malicious, *NotCheat* if regular), $p', \mu)$, where:

$$q'' = q' - (p'' - p') \quad (5.14)$$

And:

$$p'' = (M \cdot p') - p'^M \quad (5.15)$$

The objectives of the second-stage game are:

1. Incentivize the good behavior by rewarding the honest behavior and penalizing the dishonest one.
2. Eliminate the unnecessary work done by the validators in case if the risk likelihood (i.e., p') of a proposer is low. This would contribute to saving computation and communication cost which, as a result, enhance the performance of the consensus protocol.

5.5 Security Analysis

In this section, we present some blockchain consensus attacks and demonstrate how TrueBFT protects against them. Exploiting a game theoretical model that rewards honest behavior and penalizes dishonest behavior, motivates the defenders in

TrueBFT to adhere to the protocol and disincentivizes malicious parties. This way, the defenders (leaders and validators) will work hard to maximize the utility that each can gain as a reward for excellent work and avoid the punishment (cost) that might incur due to misbehaving or not obeying the protocol. Additionally, our approach mitigates the incentive for a malicious proposer to cheat, especially if the punishment cost is high. Our theoretical game model can protect against the following attacks:

5.5.1 Malicious or Lazy Validation-Leading

This attack happens when a validation-leader colludes with its corresponding malicious proposing node. It could result in many problems such as utilizing the minimum number of validators or colluding with other malicious nodes as validators. Another type of this attack is the lazy validation-leading, in which the validation-leader does not execute the protocol or does not obey its requirements. For example, it is possible for a validation-leader node to produce an assignment that is not random. Utilizing the reward and punishment payoffs provided by the proposed game model mitigates the incentive to conduct such an attack.

5.5.2 Block Withholding Attack [78]

In this attack, a dishonest validator does not participate in the validation process or does not reveal the results of the verification in favor of a malicious proposing node. The reward and punishment provided by our game incentivize the validators to avoid this attack.

5.5.3 Denial of Service Attack [18]

In this attack, an omission fault occurs due to: (a) a leader avoiding selecting validators as instructed by the protocol, or (b) a validator avoiding the validation

process or broadcasting a vote. This attack could occur for various reasons, but mostly that the fees associated with a proposed block are not worth working on it [18]. Rewarding honest players can encourage the leaders and validators to deliver their service as required.

5.5.4 Malicious Block Proposing

The main attack that we are defending against is the invalid blocks' proposal by malicious proposers. This attack happens when a proposing node maliciously proposes an invalid new block. This attack occurs for various incentives, including double spending. Integrating random and anonymous validators' selection with a game theoretical model contributes substantially to mitigating this attack. This is because a malicious proposer does not know the nodes that will validate its proposed block, which makes it hard to corrupt or bribe them to agree on its invalid block. Additionally, the punishment enforced by the game model could alleviate the attack motives.

5.6 Conclusion

In this chapter, we have proposed a two-stage game model and adapted it to TrueBFT. Our resulting protocol randomly employs a different set of different size of validators each time a new block is proposed. Additionally, TrueBFT applies a game theoretical rewarding mechanism to reward honest adhered parties and punish malicious ones. Furthermore, the second-stage game releases the validators from the always-validate mode in case of environments with low hostility.

Our proposed TrueBFT protocol described so far, however, suffers from two limitations. First, the proposers-leaders mapping guarantees the anonymous mapping only for one time. In other words, when a proposer proposes a block for the second

time, its leaders' identities are revealed. This protocol was originally designed for our Block-Supply Chain, where each node authenticates a product and proposes a block for it only once. Second, the round-robin scheduling for proposing new blocks is vulnerable to DDoS attacks. This is because an adversary knows when a certain proposer will propose its block. Hence, the adversary can launch an attack on this proposer, preventing it from achieving the proposal. Again, TrueBFT was designed for the Block-Supply Chain, where the products flow in a certain path known to all nodes in the supply chain network.

To overcome the first limitation, we in Chapter 6, propose a dynamic proposers-leaders mapping scheme that preserves anonymity and does not require a static mapping party (e.g., the blockchain initiator). Moreover, we propose anonymous scheduling, along with the dynamic mapping scheme, to counter the second limitation by anonymizing the identities of the proposing nodes.

6 Dynamic Proposers-Leaders Mapping

This chapter presents solutions to the limitations of our proposed TrueBFT protocol presented in Section 5.6 — specifically, the static proposers-leaders mapping and the round-robin scheduling for proposing new blocks. In addition, we discuss in this chapter the safety and liveness of TrueBFT.

6.1 Introduction

Our proposed TrueBFT protocol in Chapter 4 was initially designed for our Block-Supply Chain application (Chapter 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 initiator 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 by every node 'only once.' As mentioned in Chapter 3, one successful authentication event corresponds to one block added to the product's blockchain. When the product reaches the market (i.e., end consumer), none of the previous nodes in the supply chain will re-propose a new block for this product. Thus, for an application that allows its 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 require 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 re-propose again). As a result, to make TrueBFT suitable for any other blockchain applications, we need to make the proposers-leaders mapping *dynamic* in a way that preserves the anonymity.

In this chapter, we first 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. Additionally, the new mapping scheme introduces new scheduling for proposing new blocks that anonymizes the identities of the proposing nodes. In the second part of this chapter, we analyze the safety and liveness of TrueBFT and argue that TrueBFT enjoys the same level of Tendermint's safety and liveness.

6.2 Dynamic Proposers-Leaders Mapping

In this section, we describe in detail the dynamic mapping mechanism, which is executed in two phases: 1) initialization at the genesis state, and 2) during the verification phase of TrueBFT.

6.2.1 Initialization Phase

Similar to the initialization phase described in Section 4.4.1, the blockchain initiator executes this phase at the genesis state. This phase is performed only once. After the genesis state TrueBFT proceeds utilizing the dynamic proposers-leaders mapping in a decentralized manner. The blockchain initiator uses its pair of keys (i.e., public

$in.pk$ and private $in.sk$) to execute this phase.

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

- Creates a publicly known list that holds the second public keys pk_2 s (i.e., hidden identities) for all nodes; we call this list *SList*. Each node in the network has this list.
- Creates another list (*CycleSlots*) that represents the **proposing slots** in the **next cycle**. Each cycle consists of many slots, so for each slot a proposer proposes a new block. For example, a cycle may have slots equal to the number of nodes in the network. For each slot, only one block is proposed. Therefore, each cycle slot (cs) in this list can be selected only once, so that 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 at the genesis state. After

Algorithm 8: Dynamic Mapping Initialization Phase

Input : A population A of n nodes having their pk_1 s and pk_2 s

```

1 foreach  $node \in A$  do
2   |  $SList.add(node.pk_2)$ 
3 end
4 Create  $CycleSlots$ , where  $r = 2$  and  $s = [1, n]$ 
5 foreach  $node \in A$  do
6   | Append  $SList$  to the  $node$ 's genesis block
7   | Append  $CycleSlots$  to the  $node$ 's genesis block
8 end
```

that, each cycle 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 (i.e., executed by the blockchain initiator at the genesis state). After the first cycle, the protocol proceeds independently, utilizing the dynamic approach to assign the anonymous mapping. Algorithm 8 illustrates the process of dynamic mapping's initialization phase. Note that we chose the number of slots in our system to be equal to the number of nodes (n).

6.2.2 Dynamic Mapping During the Verification Phase

This phase takes place during the verification phase of TrueBFT. 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 validate and vote 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 for this *cs* to be mapped to the newly selected proposer.

3. The *current* validators: These nodes assign the mapping for the *new* selected proposer/leaders and vote on it.

The following subsections explain in detail how each of these entities performs their roles.

6.2.2.1 New Proposers Selection

As mentioned in Chapter 4, 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 9 shows the anonymous new proposer selection, in which the current proposer does the following:

1. Randomly selects a new proposer's hidden ID (i.e., $new.pk_2$) from the public list $SList$.
2. Generates a random number ($Rand_0$).
3. Randomly, selects a cycle slot (cs) from the list $CycleSlots$.
4. Forges an anonymous new proposer ($NewPr$) by encrypting $Rand_0$, and cs with $new.pk_2$ as below:

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

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 are

Algorithm 9: Selecting a New Proposer and its Cycle Slot

Input : $SList$ and $CycleSlots$
Output: $NewPr$, cs , and $[gs_1, gs_4]$
1 Randomly select $new.pk_2$ from $SList - current.pk_2$
2 Randomly select cs from $CycleSlots$
3 Generate $Rand_0$
4 $NewPr \leftarrow E_{new.pk_2}(Rand_0 || cs)$
5 for $i \leftarrow 1$ **to** 4 **do**
6 $gs_i \leftarrow [((i - 1) \cdot \frac{SList.size}{4}) + 1, i \cdot \frac{SList.size}{4}]$
7 end
8 Return $NewPr$, cs , and $[gs_1, gs_4]$

not involved in the dynamic proposers-leaders mapping at all. Although the second public keys are publicly published, there is no correlation between a node's second public key (pk_s) and first public key (pk_1). 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., $new.pk_2$). Furthermore, sk_2 is private and only known to the owner node.

5. As discussed in Section 4.4.1, to avoid selecting a *validator* by more than one *leader*, we assign each node a range 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 hidden IDs (i.e., pr_2s) for the nodes from which 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 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:

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

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 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. Signs and broadcasts $NewPr$, 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.

6.2.2.2 New Leaders Selection

After receiving the newly selected proposer and its cycle slot, each current leader 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 new leader twice. The mechanism for selecting new leaders is performed in a way similar to selecting a new proposer. Algorithm 10 illustrates how to select new leaders anonymously, where each current validation-leader i conducts the following:

1. Randomly selects a new validation-leader's hidden Id (i.e., $new.pk_2$) from the current leader's range sg_i in the list $SList$.
2. Generates a random number ($Rand_0$).
3. Forges an anonymous new leader ($NewVl_i$) by encrypting $Rand_0$, and cs with the new selected leader's public key $new.pk_2$ as below:

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

Algorithm 10: Selecting a New Leader for the Selected Proposer

Input : $SList$, $NewPr$, and cs
Output: $NewVl_i$, $S_{i,1}$, and $Rand_{i,2}$
1 Randomly select $new.pk_2$ from $SList - current.pk_2$
2 Generate $Rand_0$
3 $NewVl_i \leftarrow E_{new.pk_2}(Rand_0 || cs)$
4 Generate $Rand_{i,1}$
5 $S_{i,1} \leftarrow hash(NewPr || NewVl_i || Rand_{i,1})$
6 Generate $Rand_{i,2}$
7 Return $NewVl_i$, $S_{i,1}$, and $Rand_{i,2}$

4. Randomly generates a $Rand_{i,1}$. This random number serves similar to the one we utilize in the static mapping (Section 4.4.1).
5. Partially creates 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 4.4.1) consists of all four leaders' public keys.

6. Randomly generates 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_2 s.
7. Signs and broadcasts $NewVl_i$, $S_{i,1}$, and $Rand_{i,2}$.

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 who the new proposer is due to using $Rand_0$ (step 3 in Algorithm 9). Hence, a *current* leader cannot select a *new* leader that can **collude** maliciously with the *new* selected proposer.

Algorithm 11: 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 $(NewVl_{[1-4]}, S_{[1-4].2}, Rand_{[1-4].2})$
1 **if** *Check* cs *is in* $CycleSlots$ **then**
2 $S_1 \leftarrow S_{1.1} || S_{2.1} || S_{3.1} || S_{4.1}$
3 **foreach** $NewVl_i$ **do**
4 $S_{i.2} \leftarrow hash(S_1 || Rand_{i.2})$
5 **end**
6 **Return** $(NewPr, S_1)$ and $(NewVl_{[1-4]}, S_{[1-4].2}, Rand_{[1-4].2})$
7 **else**
8 **Return** Invalid cycle slot
9 **end**

6.2.2.3 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, they need to vote on mapping the same way they vote on the proposed block. Algorithm 11 shows the task for every validator. Each validator does the following:

1. Validates that cs is an existing valid cycle slot in the $CycleSlots$.
2. Constructs S_1 as follows:

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

3. For each new selected validation-leader $NewVl_i$, does:

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

4. Signs and votes on:

- The new selected proposer and its secret ($NewPr$ and S_1).

Algorithm 12: Checking if a Node is a Proposer or a Leader

Input : $NewPr, NewVl_{[1-4]}$
1 if $D_{own.sk_2}(NewPr)$ then
2 $own.cs \leftarrow NewPr.cs$
3 $own.S_1 \leftarrow NewPr.S_1$
4 end
5 foreach $NewVl_i$ do
6 if $D_{own.sk_2}(NewVl_i)$ then
7 $own.cs \leftarrow NewVl_i.cs$
8 $own.S_2 \leftarrow NewVl_i.S_2$
9 $own.Rand_2 \leftarrow NewVl_i.Rand_2$
10 $own.C.add(own.cs, own.S_2, own.Rand_2)$
11 end
12 end

- Each new selected leader $NewVl_i$, its secret, and its random number ($NewVl_i$, $S_{i,2}$, and $Rand_{i,2}$).

6.2.2.4 Checking if a Node in the New Mapping

After reaching a 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 own second private key (i.e., $own.sk_2$) as below:

$$D_{own.sk_2}(NewPr)$$

If successful, then this node is the proposer for the slot s in the next cycle c . Similarly, each node checks if it is a new leader for that cycle slot by decrypting all the $NewVls$. Algorithm 12 illustrates this process.

It is worth noting that 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 it remains the same for every cycle (we choose it to be equal to the number of nodes in the network (n)).

6.2.3 Proposers/Leaders' Legitimacy Checking

So far, we have described how the dynamic mapping is executed. However, it is possible that a malicious node manages to act as the proposer/leader for a particular cycle slot cs , by sending its secret claiming it is the proposer/leader for this cs . Thus, we counter this issue by introducing variable proofs that need to be submitted to the network by the consensus participants (i.e., proposers and leaders) so the other nodes can check their legitimacy.

At the initialization phase of TrueBFT, 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:

$$\pi' \leftarrow \text{Sign}_{in.sk}(pr.pk || cs)$$

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 specific cycle slot.

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

$$\pi \leftarrow \text{Sign}_{in.sk}(vl.pk || cs)$$

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 this, 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) will forge 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 included 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 proof of its legitimacy to propose for cs .

Likewise, for the *new* leaders, each *current* leader is responsible for creating 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 from 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.

6.2.4 Liveness Issues

In TrueBFT, we employ only one proposer per *cs*. However, having one proposer per *cs* could threaten the protocol liveness. This is because the selected proposer for a particular slot may fail to propose due to regular crashes or maliciousness. To overcome such an issue, TrueBFT allows a proposer to re-propose in case of failures (Section 4.4.2.3). Each proposal attempt is called a *round*. A cycle slot has a finite number of rounds (i.e., *rounds-limit*). The number of rounds is application-specific. This way, the proposers are given several chances to propose equal to the number of rounds. For example, if a proposer fails to propose at $round_0$ due to network failures, the validators vote to move to the next round (i.e., $round_1$), and increase the *proposer-time-out*, as discussed in Section 4.4.2.3.

Nevertheless, what happens if the proposer is malicious and refuses to propose, or the proposer is off for a long time? In this case, the liveness of the protocol is violated. In response to this, inspired by Tendermint design, TrueBFT allows the next cycle slot to be triggered. Put differently, the proposer for the next cycle is allowed to broadcast its proposal.

Although such an approach can preserve liveness, it raises a new problem. The dynamic mapping in TrueBFT is executed in a 1:1 fashion. That is, one slot in a cycle produces a mapping for one slot in the *next* cycle. If a proposer fails to accomplish its proposal, then this implies no mapping is created for a slot in the next cycle. To overcome this, in case of the proposal fails in a cycle slot, TrueBFT requires the proposer for the following cycle slot to perform 'two' mappings to offset the missing mapping in the previous slot.

6.2.5 Limitations

The dynamic proposers-leaders mapping achieves 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) which corresponds to the *secret hidden identity* (i.e., second public key (pk_2)). 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)) is private information and unknown to the other nodes. Although the pk_2 s for all nodes are publicly published in the list *SList* (for the sake of new proposers and leaders selection), no node in the network knows which pk_2 relates to which node except the owner node (i.e., the node that has the corresponding sk_2). In addition, no *current* proposer recognizes its leaders nor does a leader know its proposer or its peer leaders. Therefore, colluding between these nodes to compose a malicious *new* mapping is not feasible. However, this mapping suffers the following two limitations:

1. The *current* proposer may observe the public identity (i.e., pk_1) of the *new* proposer when this new proposer proposes its block in *cs*. Hence, it can correlate the hidden identity (i.e., pk_2) to pk_1 . However, this is not useful 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 pk_2), when the malicious *current* proposer has the chance to re-propose in future. Then, the malicious *current* proposer could launch a DDoS attack to prevent the new proposer from propos-

ing. Nevertheless, switching to a new *cs* in case of faulty proposers can maintain the protocol liveness and mitigate the consequences of such a case.

2. It is possible that a *new* selected proposer (*NewPr*) for a cycle slot (*cs*) can be also selected as a *new* leader (*NewVl*) for itself by one of the *current* leaders. This is because there is no means of 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. TrueBFT, however, can tolerate this issue since it is based on Tendermint and can tolerate up to one-third of Byzantine fault.

6.3 Discussion on Safety and Liveness

To a large extent, TrueBFT relies on the consensus mechanism used in Tendermint [9, 47] to reach consensus on blocks in the ledger. As a result, it inherits Tendermint's capability to ensure safety and liveness as long as the super-majority (i.e., more than two-thirds) of the validators set are honest.

In this section, we argue that our proposed TrueBFT protocol can guarantee the same level of safety and liveness of Tendermint. As mentioned, the main advantage of TrueBFT is that it is truly decentralized (i.e., it does not rely on the same set of validators), and it anonymizes the identities of the consensus participants to make it resistant to attacks like DDoS. To achieve that, we have introduced new players and roles to the existing design of Tendermint. In our TrueBFT, the consensus process is carried out by three parties: proposers, leaders, and validators. Next, we show that despite those newly introduced players and roles, TrueBFT can still preserve the safety and liveness in case of faulty consensus participants.

6.3.1 Safety

To recall, an atomic broadcast is a broadcast that either completes correctly at all nodes, or all nodes abort with no side effects. For a message m , atomic broadcast satisfies the following properties [10, 31]:

- **Validity:** If a correct node broadcasts m , it delivers m eventually.
- **Agreement:** If a correct node delivers m , then all correct nodes deliver m eventually.
- **Integrity:** Message m is received (delivered) by each node at most once, and only if it was previously broadcast.
- **Total order:** If the correct nodes p and q deliver m_1 and m_2 , then p delivers m_1 before m_2 if and only if q delivers m_1 before m_2 .

We take m to be a block in the ledger. Satisfying safety implies satisfying atomic broadcast. Nevertheless, TrueBFT, as in Tendermint, does not satisfy the validity property. This is because there is no guarantee that a proposed block will be committed eventually, since validators may move to a new round and commit a different block [9]. Put differently, the safety of a consensus protocol is compromised if two correct nodes decide differently on a proposed block, two blocks are committed at the same height in the ledger, or some correct nodes agree on an invalid block [9]. In this section, we show how TrueBFT preserves safety in the presence of a malicious/faulty consensus participant.

6.3.1.1 Faulty Proposers

Tendermint allows only one proposer per round, and every validator knows the proposer responsible for producing a block for that round. Thus, one block is added to

the ledger at a time if the block is valid. Nevertheless, a faulty proposer may broadcast different blocks to different validators. However, Tendermint preserves safety in such a case via locking mechanisms. As mentioned, TrueBFT strongly relies on this design feature of Tendermint. The only difference between TrueBFT and Tendermint is in the way the proposers are selected and ordered. In Tendermint, the proposers are ordered via a simple, deterministic round-robin mechanism. However, as argued before, because Tendermint selects blocks proposers deterministically, an adversary could find exactly who the next blocks proposers will be and launch attacks on them. In response, TrueBFT anonymizes the identities of proposers, so that the proposers are only known after they broadcast their secrets and blocks. Next, we present several scenarios that might threaten the safety and the countermeasures provided by TrueBFT.

- **Scenario 1:** As in Tendermint, it is possible for two blocks to be proposed by a faulty proposer at the same round [9]. If the two blocks are proposed for the same height, then this leads to compromising the safety of the protocol as two correct nodes might decide differently.

Countermeasure: To overcome this, Tendermint restricts the number of committed blocks per height (i.e., only one block can be committed to the ledger at a certain height). This is achieved via a locking mechanism. In this mechanism, a validator is *locked* on one block if it receives a valid polka for that block (i.e., more than two-thirds of pre-votes). Once the validators are locked on a block, they are not allowed to vote on another block. As a result, only one valid block will be committed to the ledger at the same height. TrueBFT inherits this property of Tendermint and enforces that only one block is committed per cycle.

- **Scenario 2:** Since TrueBFT does not use the round-robin scheduling used by Tendermint for assigning proposers to rounds, it is possible for a proposer to claim the right to propose for a particular cycle slot. As a result, if two proposers propose for the same cycle slot, where each of them has different validators, the nodes in the network may commit two blocks for the same height, or decide differently. This is because the locking mechanism used in Tendermint will not work. In Tendermint, the validators are only locked on a block when they receive more than two third of pre-votes. If the same set of validators are used for all the proposed blocks, this locking mechanism will work, assuming that more than two-thirds of these validators are honest. However, in TrueBFT, each proposer might have a different set of validators. As a result, it is possible for the validators of a set to be locked on a block and the other validators in the second set to be locked on another block. Thus, compromising the *agreement property* of the atomic broadcast.

To illustrate this, consider two proposers X and Y , Each of them claims that it is the legitimate proposer for the cycle slot cs . Proposer X is mapped to four validators/pre-voters and four pre-committers. Proposer X proposes the block $blockX$ at round R in cs . Similarly, proposer Y is mapped to four validators/pre-voters and four pre-committers. Proposer Y proposes the block $blockY$ at the same round (i.e., R). Assume that the pre-committers for proposer X receive a polka for $blockX$ (i.e., more than two-thirds of pre-votes on $blockX$ form the validators/pre-voters for proposer X). As a result, they are locked on $blockX$, and they proceed accordingly. Note that they can only be locked on $blockX$, and they disregard any pre-votes on other blocks (i.e., $blockY$ in this case). Similarly, let us assume that the pre-committers for proposer Y are locked on $blockY$, as a result of receiving a polka for $blockY$. In this case, the two sets

of pre-committers are locked on two different blocks, and they can proceed to the commit step. Suppose a correct node A receives more than two-thirds of pre-commits for $blockX$, and commits $blockX$. For the same round R , another correct node B receives more than two-thirds of pre-commits for $blockY$ and commits $blockY$. This is a violation of safety since two correct nodes commit two different blocks for the same height.

Countermeasure: To counter this, TrueBFT exploits verifiable proofs (i.e., π') assigned to proposers from the previous cycle. When a proposer broadcasts its secret and block, it should broadcast its proof along with them, so each other node in the network can easily verify this proof. This proof (π') is a digital signature signed by the *mapping* proposer's private key from the previous cycle, and it includes the *mapping* proposer's proof, and the unique cycle slot number. In addition, π' is validated and voted on by the validators of the *mapping* proposer to ensure the validity and uniqueness of this proof. As a result, only the rightful proposer has the right to propose for a particular cycle slot. Note that the proposer for a current cycle slot is selected blindly by a mapping proposer from the previous cycle (Section 6.2.2.1). This way of anonymous selection prevents the collusion between malicious nodes.

- **Scenario 3:** Since the leaders in TrueBFT determine the number of validators and select them, it is possible for a malicious proposer to collude with its leaders to select a number of validators where more than two-thirds of them are malicious. This can lead to committing an invalid block to the ledger; thus, undermining safety.

Countermeasure: TrueBFT anonymizes the interaction between the consensus participants via a dynamic mapping, in which each proposer is blindly

mapped to four leaders. These parties communicate with each other via secrets, and their identities are only revealed after completing their tasks. Hence, this way of mapping and communicating prevents the malicious collusion, since the proposers and their leaders do not know each other in advance.

6.3.1.2 Faulty Leaders

The leader nodes in TrueBFT are the newly introduced players in the consensus process. Their job is to decide the number of validators and randomly select them. Although the leaders do not contribute much in committing blocks, malicious leaders might threaten the safety of the protocol. Next, we show a possible threat scenario introduced by leaders, and how TrueBFT addresses it.

- **Scenario:** Since the leaders are not publicly known for a particular proposer, it is possible that a malicious node claims that it is the leader for a malicious proposer so that they collude to inject an invalid block to the ledger.

Countermeasure: TrueBFT, as described earlier, exploits verifiable proofs assigned to leaders as well (i.e., π). Hence, when a leader broadcasts the number of validators and instructs the selected validators, it should broadcast its proof too. As a result, the leader's proof serves as evidence of legitimacy.

6.3.1.3 Faulty Validators

We did not introduce any further roles for the validators in TrueBFT. The validators in TrueBFT function in a similar way to the ones in Tendermint. Hence, we will discuss in this section how Tendermint preserves safety. Tendermint and TrueBFT ensure safety as long as more than two-thirds of validators are honest (non-Byzantine). To achieve that, Tendermint exploits a locking mechanism. The locks are used to

ensure that no two validators will commit a different block at the same height, assuming that less than one-third of the validators are malicious. Put differently, once there exists more than two-thirds of pre-commits on a block in a round, the network is locked on that block [9]. This way, it is impossible to produce a valid polka (i.e., more than two-thirds of pre-votes) for a different block at the same round or a higher round in the same cycle slot. As a result, only one block can be committed to the ledger by the network.

Tendermint guarantees that no validator can vote on two blocks for the same round due to locks. Next, we sketch a brief proof that Tendermint (and hence TrueBFT) guarantees that safety is always preserved (i.e., validators will never commit different blocks at the same height), as long as more than two-thirds of validators are honest [9].

Assume that less than one-third of validators are Byzantine and at least one honest validator decides on a block B . This means that no honest validator will decide on any block other than B . To prove that, consider the current round R in which at least one honest validator commits block B at this round. This implies that this validator received a valid polka (i.e., more than two-thirds of pre-commits for block B at round R). Taking into account that less than one-third of validators are Byzantine, by arithmetic at least one-third of honest validators must have pre-committed block B at round R . We know that to pre-commit a block, a validator should lock on that block. This implies that this one-third of honest validators must have a lock on block B at round R . As a result, no other block can be committed by honest validators unless some of the honest validators unlock from block B , which is impossible because they are honest. So, at most, less than two-thirds of validators are available to vote for any block other than B , which is insufficient voting power.

The only difference between TrueBFT and Tendermint is the way the validators are selected. In Tendermint, the validators are pre-determined and publicly known to

the network. In contrast, the validators' set in TrueBFT is replaced each time a new block is proposed. As a result, it is possible for a node to claim the right to validate and compromise safety if acted maliciously. To encounter this issue, each validator in TrueBFT must submit a verifiable proof (τ) along with its vote indicating that it has been elected by a legitimate leader (Section 4.4.2). Each other node can easily verify this proof before considering the received vote.

6.3.2 Liveness

In consensus, liveness means that all processes (nodes) will eventually agree on a value, and the computation will terminate. TrueBFT's ability to maintain liveness is dependent on Tendermint. It is known that Tendermint prioritizes safety over liveness [94]. As a result, Tendermint might halt momentarily until more than two-thirds of the validators come to a consensus. Next, we discuss how TrueBFT preserves liveness in the presence of faulty consensus participants.

6.3.2.1 Faulty Proposers

Proposers might undermine liveness if they fail to achieve their proposals. Below, we present some possible scenarios of such a case, and show how TrueBFT deals with them.

- **Scenario 1:** One way to violate liveness is when a malicious node rushes in at the start of a proposing cycle slot to become a proposer, and then refuses to perform the proposal.

Countermeasure: To withstand such a situation, TrueBFT uses a proposing verifiable proof (i.e., π') for each proposer to provide as evidence of its legitimacy. In TrueBFT, there exists only one proposer per slot. So, only the legitimate proposer's block will be considered.

- **Scenario 2:** It is possible for a legitimate proposer to be faulty and not accomplish the proposal.

Countermeasure: To encounter this, TrueBFT utilizes the *proposer-time-out* used in Tendermint. If this time-out terminates, a new round of proposing will start allowing the proposer to re-propose. The only difference between TrueBFT and Tendermint in this regard is that we give a single proposer a number of rounds to re-propose in case of failure, while in Tendermint, each round is assigned to a new proposer. The reason behind this change in design is that each cycle slot is assigned to one proposer, and we need to avoid migrating to a new cycle slot in case of regular crashes. However, in case a faulty proposer remains faulty for many tries and exceeds the number of rounds allowed, the nodes in the network vote to move to the next cycle allowing the next proposer in-line to propose; thus, preserving liveness, as discussed in Section 6.2.4.

6.3.2.2 Faulty Leaders

As mentioned, this is the only new player to the consensus with respect to Tendermint. In this section, we show possible ways that leaders may compromise liveness, and we discuss how TrueBFT design preserves it.

- **Scenario 1:** Since the leaders are not the same for all proposers and they are not publicly known, it is possible for a set of malicious nodes to claim that they are the leaders for a particular proposer, and then stop performing their roles. This malicious act will undermine liveness, since no validators will be selected to validate the proposal.

Countermeasure: The verifiable proofs for the leaders discussed in Section 6.2.3 serve as authorization cards, so that only the node that has that proof can

serve as a leader.

- **Scenario 2:** It is possible for some legitimate leaders to be faulty/Byzantine; thus, stalling the protocol.

Countermeasure: TrueBFT uses at least four leaders per proposer to overcome such halts. The reasoning behind this choice is that four is the minimum number to provide tolerance to a single Byzantine fault [9]. 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 TrueBFT. However, if more than one-third of leaders are faulty, TrueBFT, like in Tendermint, stalls momentarily, and then switches to a new round; thus preserving liveness.

6.3.2.3 Faulty Validators

As discussed in Section 6.3.1.3, we use the same consensus mechanism as Tendermint, and we did not introduce any further roles to the validators.

The locking mechanism used by Tendermint contributes to maintaining the liveness as well. As discussed earlier, a validator in Tendermint (a pre-committer in ours) is locked on a block if it witnesses a valid polka (i.e., more than two-thirds of pre-votes for that block). This contributes to maintaining safety since an honest validator will not pre-vote on a block other than the one it has been locked on. Put differently, only one valid block can be committed at a time. Nevertheless, if some nodes remain locked on a block that the rest of the network does not wish to commit (e.g., not enough pre-commits received for this block), then this can compromise liveness. In response, Tendermint requires the node that was locked on a block from the previous round to *unlock* when it sees a valid polka for another block in a higher round. Having

that stated, next we summarize the liveness proof of Tendermint [9].

If there exists less than one-third of Byzantine validators, then the protocol preserves liveness (i.e., does not deadlock). This is because the only way the consensus process will deadlock is if some honest validators from different rounds have been locked on two different blocks. Assume that some honest validators are locked on block B_1 from round R_1 , and some honest validators are locked on block B_2 from round R_2 , where $R_1 < R_2$. Using the *unlock* role, the validators that are locked on B_1 will unlock when they see the valid polka for B_2 allowing the consensus process to continue. Similarly, the locked pre-committers on B_1 from R_1 will immediately unlock when they hear a polka for B_2 . This way, TrueBFT provides the same level of liveness as Tendermint.

6.4 Conclusion

In this chapter, we have proposed a novel dynamic anonymous proposers-leaders mapping mechanism and integrated it into TrueBFT to: (1) avoid relying on the blockchain initiator to anonymously map each proposer to its leaders (i.e., static mapping) at the beginning of each cycle, and (2) anonymize the identities of the proposing nodes to protect them against DDoS attacks. 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 cycle slots (i.e., one slot represents one new proposed block).

In addition, in this chapter, we have analyzed TrueBFT concerning *safety* and *liveness*. TrueBFT is highly dependent on the consensus mechanism utilized by the state-of-the-art Tendermint. Therefore, TrueBFT draws its ability to maintain safety and liveness from Tendermint. The difference between TrueBFT and Tendermint is in the way that the proposers and validators are selected. TrueBFT selects the

proposers anonymously rather than relying on pre-determined scheduling. Moreover, TrueBFT selects a different set of a different size of validators each time a new block is proposed rather than relying on fixed, known validators. This method of selection does not compromise the safety and liveness that Tendermint enjoys.

7 CONCLUSION

To conclude this dissertation, we summarize our contributions and give directions for future work.

7.1 Research Contributions

In this dissertation, we mainly focused on the problem of product counterfeiting attacks; particularly, tag reapplication attacks. Detecting tag reapplication attacks remains a significant challenge as it requires manual inspection of products. In addition, most current anti-counterfeiting supply chains are centralized, relying on a centralized authentication/tracking server to coordinate the products' authentication management. This centralization architecture creates several limitations such as tremendous processing and storage overheads to authenticate products, and single-point of failure. To address these problems, we proposed the following solutions.

- First, the Tag Reapplication Detection (TRD) system to detect reapplication attacks using low-cost NFC tags and public key cryptography. The key idea in our solution is to exploit the read-only NFC tag's counter to track the number of times the tag has been read in the supply chain. To track the number of a tag's reads, TRD utilizes a 'central' database to store this number, and digitally signs this number on a re-writable memory on the tag. Tracking the number of times a tag has been read ensures that only authorized parties authenticate the tagged product.

- The second contribution is the Block-Supply Chain, a permissioned blockchain to decentralize the product authentication and tracking process. The Block-Supply Chain is a transformation of TRD into a decentralized supply chain. In this system, each node maintains a blockchain (public ledger) for each product. This blockchain comprises chained blocks, where each is an authentication event. The Block-Supply Chain can detect tag modification, cloning, and reapplication attacks without the need for a centralized authentication authority.

In this dissertation, we also addressed three problems of current blockchain consensus protocols. First, the lack of true decentralization in existing BFT consensus protocols. Rather, most of them rely on fixed, static validators, which expose them to attacks like DDoS. Second, the choice of the number of selected validators has not received much attention in BFT consensus protocols. This is of importance as the number of validators influences the security and efficacy of consensus protocols. Third, validators' diversion from honest behavior has introduced new attacks to the blockchain technology such as block withholding and selfish mining. Such a deviation is due to the lack of robust rewarding and punishing mechanisms. In response to these problems, we proposed the following solutions.

- First, a new BFT-based true decentralized consensus protocol (TrueBFT) that does not require PoW and randomly employs a different set of a different size of validators each time a new block is proposed. The key idea of our solution is to anonymously map each proposing node to some leader nodes in a way that none of them knows its corresponding peer. However, these nodes communicate with each other via secrets assigned to them at the genesis state by the blockchain initiators. The leaders are responsible for selecting the validators for their corresponding proposer at random. This way, the identity of each node (i.e., leader/validator) is revealed after it accomplishes its job.

- Second, to determine the number of validators in a way that considers the security and performance, and to enforce the honest behavior of validators, we proposed a game theoretical model and integrated it into TrueBFT. This game theoretical driven approach utilizes the game model to analyze the risk likelihood of the proposing nodes. This risk likelihood is used to determine the number of validators involved in the consensus process. As a result, the number of validators depends on the hostility of the network environment. Thus, this method of deciding the number of validators can enhance the consensus performance, especially in a low hostile environment. Additionally, TrueBFT rewards honest validators and punishes dishonest ones.
- Third, although the anonymous proposer-leaders mapping mechanism used in TrueBFT contributes towards true decentralization, the mapping is fixed and anonymous for only one cycle of proposing. That is, 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. To overcome such an issue, we proposed a novel, decentralized, dynamic mapping between the nodes that participate in the consensus process. This mapping mechanism does not rely on any centralized proposers-leaders mapper (i.e., blockchain initiator). Instead, the regular nodes in the network perform the mapping dynamically and anonymously. Each time a new block is proposed, the proposer creates and proposes a new mapping in a way that preserves anonymity.

Table 7.1 summarizes the problems that this dissertation describes, and the contributions to overcome these problems.

Table 7.1: Contribution summary.

Problem	Contribution	Approach
Tag reapplication attack	Tag Reapplication Detection (TRD) system	Public key cryptography and NFC tags
Centralization of current anti-counterfeiting supply chains	Block-Supply Chain	Blockchain Technology
Non-true decentralization of current consensus protocols	TrueBFT: a Truly decentralized consensus protocol	Randomness and anonymous proposers-leaders mapping
- Diversion from honest behavior, - The number of selected validators	Two-stage Bayesian game theoretical model	Game theory
Static proposers-leaders mapping	Anonymous Dynamic mapping	Regular proposers forge the mapping

7.2 Future Directions

This dissertation addressed problems in two main domains; namely, product counterfeiting and blockchain consensus. We believe that there are many aspects to explore in these two domains. Below, we present some future work directions.

Concerning product counterfeiting, as we discussed in Section 2.6, there are some scenarios in which our approach to detect tag reapplication attacks may fail. This occurs when a counterfeiter removes and reapplies a tag without reading it. TRD (and Block-Supply Chain) cannot detect this attack, because the *Counter* value is not increased. One way to counter this issue is to fingerprint products, digitally sign these

fingerprints, and store them on the tags of the products. Any unique product's special features can be used as a fingerprint such as text and color patterns. Then, we can extract these fingerprints and use machine learning algorithms to detect abnormalities or false correlation between the extracted fingerprints and the corresponding, digitally stored ones.

With regard to the dynamic mapping in TrueBFT, we discussed in Section 6.2.5 that a mapping proposer may observe the identity of the mapped proposer over time. This knowledge might be used by the mapping, malicious proposer to launch a DDoS attack on the mapped proposer. This is rare to occur, but it is feasible. To overcome this, one possible solution is to link each hidden identity (i.e., the second public key pk_2) to a random number, so instead of selecting the hidden identity pk_2 directly, the mapping proposer selects this random number. Nevertheless, to prevent the mapping proposer from observing the identity of the mapped proposer, we shuffle the linked random numbers at the beginning of every cycle, so each pk_2 will be linked to a different random number every cycle.

Finally, blockchain technology can be applied to many aspects of supply chain technology including provenance, brand protection, just-in-time forecasting, etc.

In conclusion, we believe that product anti-counterfeiting and the revolution of blockchain technology are exciting research areas.

Bibliography

- [1] Abeyratne, S.A., Monfared, R.P.: Blockchain ready manufacturing supply chain using distributed ledger (2016)
- [2] Alzahrani, N., Bulusu, N.: A new product anti-counterfeiting blockchain using a truly decentralized dynamic consensus protocol. *Concurrency and Computation: Practice and Experience* p. e5232
- [3] Alzahrani, N., Bulusu, N.: Securing pharmaceutical and high-value products against tag reapplication attacks using nfc tags. In: *Smart Computing (SMARTCOMP), 2016 IEEE International Conference on*. IEEE (2016)
- [4] Alzahrani, N., Bulusu, N.: Block-supply chain: A new anti-counterfeiting supply chain using nfc and blockchain. In: *Proceedings of the 1st Workshop on Cryptocurrencies and Blockchains for Distributed Systems*. pp. 30–35. ACM (2018)
- [5] Alzahrani, N., Bulusu, N.: Towards true decentralization: A blockchain consensus protocol based on game theory and randomness. In: *International Conference on Decision and Game Theory for Security*. pp. 465–485. Springer (2018)
- [6] Aumann, R.J., Hart, S.: *Handbook of game theory with economic applications*, vol. 2. Elsevier (1992)
- [7] Bhardwaj, R., Dixit, V., Upadhyay, A.K., ARSD, K.: An overview on tools for peer to peer network simulation. *simulation* 1(26), 32 (2010)
- [8] Bonneau, J., Felten, E.W., Goldfeder, S., Kroll, J.A., Narayanan, A.: Why buy when you can rent? bribery attacks on bitcoin consensus (2016)
- [9] Buchman, E.: Tendermint: Byzantine fault tolerance in the age of blockchains. Ph.D. thesis (2016)
- [10] Cachin, C., Guerraoui, R., Rodrigues, L.: *Introduction to reliable and secure distributed programming*. Springer Science & Business Media (2011)
- [11] Cachin, C., Vukolić, M.: Blockchains consensus protocols in the wild. *arXiv preprint arXiv:1707.01873* (2017)
- [12] Castro, M., Liskov, B., et al.: Practical byzantine fault tolerance. In: *OSDI*. vol. 99 (1999)

- [13] CDC: Counterfeit drugs, <https://wwwnc.cdc.gov/travel/page/counterfeit-medicine>
- [14] Cheung, H., Choi, S.: Implementation issues in rfid-based anti-counterfeiting systems. *Computers in Industry* 62(7), 708–718 (2011)
- [15] Copeland, C., Zhong, H.: Tangaroa: a byzantine fault tolerant raft (2016)
- [16] Cox, T.: Eos.io technical white paper. GitHub repository (2017)
- [17] Daigle, L.: Following pharmaceutical products through the supply chain (2012)
- [18] De Angelis, S.: Assessing security and performances of consensus algorithms for permissioned blockchains. *arXiv preprint arXiv:1805.03490* (2018)
- [19] Dellarocas, C.: Mechanisms for coping with unfair ratings and discriminatory behavior in online reputation reporting systems. In: *Proceedings of the twenty first international conference on Information systems*. pp. 520–525. Association for Information Systems (2000)
- [20] Efrimidis, P.S., Spirakis, P.G.: Weighted random sampling with a reservoir. *Information Processing Letters* 97(5), 181–185 (2006)
- [21] EPCglobal, P.R.S.: 1.0., epcglobal (2007)
- [22] EVANS, K.: Online counterfeit sales will cost businesses \$135 billion this year, <https://www.digitalcommerce360.com/2011/01/05/online-counterfeit-sales-will-cost-businesses-135-billion/>
- [23] Eyal, I.: The miner’s dilemma. In: *Security and Privacy (SP), 2015 IEEE Symposium on*. pp. 89–103. IEEE (2015)
- [24] Gai, F., Wang, B., Deng, W., Peng, W.: Proof of reputation: A reputation-based consensus protocol for peer-to-peer network. In: *International Conference on Database Systems for Advanced Applications*. pp. 666–681. Springer (2018)
- [25] Gibbons, R.: *Game theory for applied economists*. Princeton University Press (1992)
- [26] Gilad, Y., Hemo, R., Micali, S., Vlachos, G., Zeldovich, N.: Algorand: Scaling byzantine agreements for cryptocurrencies. In: *Proceedings of the 26th Symposium on Operating Systems Principles*. pp. 51–68. ACM (2017)
- [27] Golchubian, A.: *Utilizing a Game Theoretical Approach to Prevent Collusion and Incentivize Cooperation in Cybersecurity Contexts*. Ph.D. thesis, Florida Atlantic University (2017)
- [28] Goodrich, M.T., Tamassia, R.: *Introduction to computer security*. Pearson (2011)

- [29] Gorbunov, S.: Pure Proof-of-Stake Blockchains, <https://medium.com/algorand/secure-blockchain-decentralization-via-committees-7602f598a0a9>
- [30] Hackius, N., Petersen, M.: Blockchain in logistics and supply chain: trick or treat? In: Proceedings of the Hamburg International Conference of Logistics (HICL). epubli (2017)
- [31] Hadzilacos, V.: Fault-tolerant broadcasts and related problems. Distributed systems (1993)
- [32] Hearn, M.: Corda: A distributed ledger. Corda Technical White Paper (2016)
- [33] Heilman, E., Kendler, A., Zohar, A., Goldberg, S.: Eclipse attacks on bitcoin's peer-to-peer network. In: USENIX Security Symposium. pp. 129–144 (2015)
- [34] Hyperledger-Community: hyperledger/fabric (2018), <https://github.com/hyperledger/fabric/tree/v0.6>
- [35] IBM: Digitizing the global supply chain <https://www.tradelens.com>
- [36] Institute, P.S.: Counterfeit situation, incident trends, <http://www.psi-inc.org/incidentTrends.cfm>
- [37] Jayachandran, P.: The difference between public and private blockchain, <https://www.ibm.com/blogs/blockchain/2017/05/the-difference-between-public-and-private-blockchain/>
- [38] Johnson, B., Laszka, A., Grossklags, J., Vasek, M., Moore, T.: Game-theoretic analysis of ddos attacks against bitcoin mining pools. In: International Conference on Financial Cryptography and Data Security. pp. 72–86. Springer (2014)
- [39] Kamvar, S.D., Schlosser, M.T., Garcia-Molina, H.: The eigentrust algorithm for reputation management in p2p networks. In: Proceedings of the 12th international conference on World Wide Web. pp. 640–651. ACM (2003)
- [40] Kiayias, A., Koutsoupias, E., Kyropoulou, M., Tselekounis, Y.: Blockchain mining games. In: Proceedings of the 2016 ACM Conference on Economics and Computation. pp. 365–382. ACM (2016)
- [41] Kiayias, A., Russell, A., David, B., Oliynykov, R.: Ouroboros: A provably secure proof-of-stake blockchain protocol. In: Annual International Cryptology Conference. pp. 357–388. Springer (2017)
- [42] King, S., Nadal, S.: Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. self-published paper, August 19 (2012)
- [43] Koh, R., Schuster, E.W., Chackrabarti, I., Bellman, A.: Securing the pharmaceutical supply chain. White Paper, Auto-ID Labs, Massachusetts Institute of Technology pp. 1–19 (2003)

- [44] Korpela, K., Hallikas, J., Dahlberg, T.: Digital supply chain transformation toward blockchain integration. In: proceedings of the 50th Hawaii international conference on system sciences (2017)
- [45] KOŠIČ, K., ČERNEC, R., BARNSLEY, A., THOORENS, F.X.: Building an open-source blockchain ecosystem with ark (2018)
- [46] Koticha, Z.: The need for an incentive scheme in algorand, <https://blockchainatberkeley.blog/the-need-for-an-incentive-scheme-in-algorand-6fe9db45f2a7>
- [47] Kwon, J.: Tendermint: Consensus without mining. Retrieved May 18 (2014)
- [48] Kwon, J., Buchman, E.: Cosmos: A network of distributed ledgers (2016)
- [49] Lamport, L., et al.: Paxos made simple. ACM Sigact News 32(4) (2001)
- [50] Larimer, D.: Delegated proof-of-stake (dpos). Bitshare whitepaper (2014)
- [51] Lehtonen, M., Staake, T., Michahelles, F.: From identification to authentication—a review of rfid product authentication techniques. In: Networked RFID Systems and Lightweight Cryptography, pp. 169–187. Springer (2008)
- [52] Liang, X., Xiao, Y.: Game theory for network security. IEEE Communications Surveys & Tutorials 15(1), 472–486 (2013)
- [53] Lielacher, A.: Welcome to the world of blockchain consensus protocols (2017), <https://btcmanager.com/welcome-to-the-world-of-blockchain-consensus-protocols/>
- [54] Liu, Y., Comaniciu, C., Man, H.: A bayesian game approach for intrusion detection in wireless ad hoc networks. In: Proceeding from the 2006 workshop on Game theory for communications and networks. p. 4. ACM (2006)
- [55] Luu, L., Saha, R., Parameshwaran, I., Saxena, P., Hobor, A.: On power splitting games in distributed computation: The case of bitcoin pooled mining. In: Computer Security Foundations Symposium (CSF), 2015 IEEE 28th. pp. 397–411. IEEE (2015)
- [56] Mackey, T.K., Nayyar, G.: A review of existing and emerging digital technologies to combat the global trade in fake medicines. Expert opinion on drug safety 16(5) (2017)
- [57] Manshaei, M.H., Zhu, Q., Alpcan, T., Başar, T., Hubaux, J.P.: Game theory meets network security and privacy. ACM Computing Surveys (CSUR) 45(3), 25 (2013)
- [58] Martino, W.: Kadena: The first scalable, high performance private blockchain (2016)

- [59] Max Kordek, Oliver Beddows, G.S.K.: lisk: Access the power of blockchain, <https://lisk.io>
- [60] Mazieres, D.: The stellar consensus protocol: A federated model for internet-level consensus. Stellar Development Foundation (2015)
- [61] Menezes, A.J., Van Oorschot, P.C., Vanstone, S.A.: Handbook of applied cryptography. CRC press (1996)
- [62] Moshi, T.: Lecture notes byzantine fault tolerance in a distributed system (November 13 2017)
- [63] Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)
- [64] Nash, J.: Non-cooperative games. *Annals of mathematics* pp. 286–295 (1951)
- [65] Nayyar, G.M., Attaran, A., Clark, J.P., Culzoni, M.J., Fernandez, F.M., Herringington, J.E., Kendall, M., Newton, P.N., Breman, J.G.: Responding to the pandemic of falsified medicines. *The American journal of tropical medicine and hygiene* 92(6_Suppl), 113–118 (2015)
- [66] NFC-Forum: Nfc forum tag type technical specifications, <https://nfc-forum.org/our-work/specifications-and-application-documents/specifications/tag-type-technical-specifications/>
- [67] NFC-Forum: Signature record type definition: Technical specification, <https://nfc-forum.org/>
- [68] NFCTAGIFY: Transparent round clear nfc sticker ntag213 chip Ø 21 mm, <https://nfctagify.com/product/clear-nfc-tags-ntag213-nfc-chip-genuine-nfc-stickers/>
- [69] Nochta, Z., Staake, T., Fleisch, E.: Product specific security features based on rfid technology. In: *Applications and the Internet Workshops, 2006. SAINT Workshops 2006. International Symposium on*. pp. 4–pp. IEEE (2006)
- [70] Nojournian, M., Golchubian, A., Njilla, L., Kwiat, K., Kamhoua, C.: Incentivizing blockchain miners to avoid dishonest mining strategies by a reputation-based paradigm. In: *IEEE Computing Conference (CC)*. London, UK: IEEE (2018)
- [71] NXP: Ntag213f/216f, https://www.nxp.com/products/identification-and-security/nfc/nfc-tags-for-electronics:MC_1429877262080
- [72] OMNET: Omnet++, <http://omnetplusplus.com/omnet/>
- [73] Ongaro, D., Ousterhout, J.K.: In search of an understandable consensus algorithm. In: *USENIX Annual Technical Conference* (2014)
- [74] Pilkington, M.: 11 blockchain technology: principles and applications. *Research handbook on digital transformations* (2016)

- [75] Poelstra, A.: On stake and consensus (2015)
- [76] Potdar, M., Chang, E., Potdar, V.: Applications of rfid in pharmaceutical industry. In: Industrial Technology, 2006. ICIT 2006. IEEE International Conference on. pp. 2860–2865. IEEE (2006)
- [77] Roger, B.M.: Game theory: analysis of conflict. The President and Fellows of Harvard College, USA (1991)
- [78] Rosenfeld, M.: Analysis of bitcoin pooled mining reward systems. arXiv preprint arXiv:1112.4980 (2011)
- [79] Saeed, M.Q., Bilal, Z., Walter, C.D.: An nfc based consumer-level counterfeit detection framework. In: Privacy, Security and Trust (PST), 2013 Eleventh Annual International Conference on. pp. 135–142. IEEE (2013)
- [80] Saeed, M.Q., Walter, C.D.: Off-line nfc tag authentication. In: Internet Technology And Secured Transactions, 2012 International Conference for. pp. 730–735. IEEE (2012)
- [81] Sankar, L.S., Sindhu, M., Sethumadhavan, M.: Survey of consensus protocols on blockchain applications. In: Advanced Computing and Communication Systems (ICACCS), 2017 4th International Conference on. pp. 1–5. IEEE (2017)
- [82] Schuh, F., Larimer, D.: Bitshares 2.0: General overview (2017)
- [83] Schwartz, D., Youngs, N., Britto, A., et al.: The ripple protocol consensus algorithm. Ripple Labs Inc White Paper 5 (2014)
- [84] Scorpecci, D.: The economic impact of counterfeiting and piracy. In: Presentation at the conference ‘Transatlantic IP Collaboration’ on April. vol. 27, p. 2009 (2009)
- [85] Sheridan, K.: Blockchain All the Rage But Comes With Numerous Risks, <https://www.darkreading.com/vulnerabilities—threats/blockchain-all-the-rage-but-comes-with-numerous-risks/d/d-id/1332038>
- [86] Sompolinsky, Y., Zohar, A.: Secure high-rate transaction processing in bitcoin. In: International Conference on Financial Cryptography and Data Security. pp. 507–527. Springer (2015)
- [87] Staake, T., Michahelles, F., Fleisch, E., Williams, J.R., Min, H., Cole, P.H., Lee, S.G., McFarlane, D., Murai, J.: Anti-counterfeiting and supply chain security. In: Networked RFID systems and lightweight cryptography, pp. 33–43. Springer (2008)
- [88] Staake, T., Thiesse, F., Fleisch, E.: Extending the epc network: the potential of rfid in anti-counterfeiting. In: Proceedings of the 2005 ACM symposium on Applied computing. pp. 1607–1612. ACM (2005)
- [89] Taheri, Y., Garakani, H.G., Mohammadzadeh, N.: A game theory approach for malicious node detection in manets. J. Inf. Sci. Eng. 32(3), 559–573 (2016)

- [90] Tait, D.: Nfc-enabled cellphone shipments to soar fourfold in next five years, <https://technology.ihc.com/490062/nfc-enabled-cellphone-shipments-to-soar-fourfold-in-next-five-years>
- [91] Tian, F.: 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)
- [92] Toyoda, K., Mathiopoulos, P.T., Sasase, I., Ohtsuki, T.: A novel blockchain-based product ownership management system (poms) for anti-counterfeits in the post supply chain. IEEE Access 5, 17465–17477 (2017)
- [93] Tseng, J.H., Liao, Y.C., Chong, B., Liao, S.w.: Governance on the drug supply chain via gcoin blockchain. International journal of environmental research and public health 15(6), 1055 (2018)
- [94] Unchained, C.: Consensus compare: Casper vs. tendermint, <https://blog.cosmos.network/consensus-compare-casper-vs-tendermint-6df154ad56ae>
- [95] Varga, A., Hornig, R.: 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. p. 60. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering) (2008)
- [96] Wood, G.: Ethereum: A secure decentralised generalised transaction ledger. Ethereum project yellow paper 151, 1–32 (2014)
- [97] Xian, X., Shi, W., Huang, H.: Comparison of omnet++ and other simulator for wsn simulation. In: Industrial Electronics and Applications, 2008. ICIEA 2008. 3rd IEEE Conference on. pp. 1439–1443. IEEE (2008)
- [98] Xu, D., Xiao, L., Sun, L., Lei, M.: Game theoretic study on blockchain based secure edge networks
- [99] Yang, L., Peng, P., Dang, F., Wang, C., Li, X.Y., Liu, Y.: Anti-counterfeiting via federated rfid tags' fingerprints and geometric relationships. In: Computer Communications (INFOCOM), 2015 IEEE Conference on. pp. 1966–1974. IEEE (2015)
- [100] Zamfir, V.: Introducing casper “the friendly ghost”. Ethereum Blog URL: <https://blog.ethereum.org/2015/08/01/introducing-casper-friendly-ghost> (2015)