BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FAST, SCALABLE AND DOS-RESISTANT PROOF-OF-STAKE CONSENSUS PROTOCOL BASED ON AN ANONYMIZATION LAYER

RYCHLÝ, ŠKÁLOVATELNÝ, A DOS-REZISTENTNÍ PROOF-OF-STAKE KONSENSUÁLNÍ PROTOKOL ZALOŽEN NA ANONYMIZAČNÍ VRSTVĚ

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR **Bc. MAREK TAMAŠKOVIČ**
AUTOR PRÁCE

SUPERVISOR **Ing. IVAN HOMOLIAK, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2021

Department of Intelligent Systems (DITS) Academic year 2020/2021

Master's Thesis Specification

22623

Student:	**Tamaškovič Marek, Bc.**
Programme:	Information Technology
Field of study:	Information Technology Security
Title:	**Fast, Scalable and DoS-Resistant Proof-of-Stake Consensus Protocol Based on an Anonymization Layer**
Category:	Security

Assignment:

1. Get familiar with existing proof-of-stake protocols, in particular, Algorand and Tendermint.
2. Make a theoretical comparison of these protocols in terms of throughput, scalability, security, liveness, safety, finality, etc. In security analysis, consider all existing proof-of-stake vulnerabilities as well as general ones.
3. Acquaint yourself with anonymization techniques used in network traffic, in particular, onion routing.
4. Design a fast proof-of-stake protocol that provides deterministic leader election and is resilient against DoS of the leader. Consider an internal onion routing as an anonymization layer of the protocol.
5. Evaluate the key performance and security measures of the proposed protocol and compare it to the related work.

Recommended literature:

- Biryukov, Alex, and Ivan Pustogarov. "Bitcoin over Tor isn't a Good Idea." 2015 IEEE Symposium on Security and Privacy. IEEE, 2015.
- Fanti, Giulia, et al. "Dandelion++: Lightweight cryptocurrency networking with formal anonymity guarantees." Proceedings of the ACM on Measurement and Analysis of Computing Systems 2.2 (2018): 29.
- Gilad, Y., Hemo, R., Micali, S., Vlachos, G. and Zeldovich, N., 2017, October. Algorand: Scaling byzantine agreements for cryptocurrencies. In Proceedings of the 26th Symposium on Operating Systems Principles (pp. 51-68). ACM.
- Buchman, Ethan. Tendermint: Byzantine fault tolerance in the age of blockchains. Diss. 2016.
- Homoliak, I., Venugopalan, S., Hum, Q., & Szalachowski, P. (2019). A Security Reference Architecture for Blockchains. arXiv preprint arXiv:1904.06898.

Requirements for the semestral defence:

- Items 1 and 2.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Homoliak Ivan, Ing., Ph.D.**
Consultant:	Perešíni Martin, Ing., UITS FIT VUT
Head of Department:	Hanáček Petr, doc. Dr. Ing.
Beginning of work:	November 1, 2020
Submission deadline:	May 19, 2021
Approval date:	November 11, 2020

Abstract

In this work, we summarized research in the state-of-the-art Proof-of-Stake protocols like Algorand, Tendermint, and LaKSA. We analyzed and summarized their features and issues. Based on the included research we implement a new PoS protocol that mitigates issues with throughput, scalability, and security.

Abstrakt

V tejto práci sumarizujeme aktuálny výskum protokolov z rodiny Proof-of-Stake ako napr. Algorand, Tendermint a LaKSA. Analyzovali sme ich funkcionalitu a tiež ich problémy. V rámci výskumu sme implementovali a novy protokol z rodiny Dôkaz-Podielom, ktorý rieši nájdené problémy ako priepustnosť, škálovatelnosť a bezpečnosť.

Keywords

Blockchain, Proof-of-Stake, Anonymization, Onion Routing, DoS Resistance

Klíčová slova

Blockchain, Proof-of-Stake, Anonymita, Onion smerovanie, DoS odolnosť

Reference

TAMAŠKOVIČ, Marek. *Fast, Scalable and DoS-Resistant Proof-of-Stake Consensus Protocol Based on an Anonymization Layer*. Brno, 2021. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Ivan Homoliak, Ph.D.

Rozšířený abstrakt

Táto práca sa zaoberá analýzou aktuálnych *„Proof-of-Stake"* (PoS) protokolov a implementáciou novo navrhnutého PoS protokolu skupinou security@FIT. Motivácia vytvoriť tento protokol vznikla analýzou aktuálnych PoS protokolov a zistení ich slabín (priepustnosť, škálovateľnosť a náchylnosť na útok typu odmietnutie služby, ang. denial of service). Novo navrhnutý protokol adresuje nájdené chyby a jeho súčasťou je tiež natívna anonymizačná vrstva.

Aktuálne jedny z najlepších PoS protokolov sú *Tendermint* alebo *Algorand*. Ich analýza sa nachádza v tejto práci. Problémy, ktoré boli odhalené pre jednotlivé protokoly sú nasledovné:

Tendermint, tento protokol je takmer čistou implementáciou byzantského systému odolnému proti poruchám. Tento protokol využíva pevne stanovenú komisiu na schvaľovanie blokov a funguje v troch krokoch: návrh blokov jednotlivými členmi komisie, hlasovanie za jeden blok a následné rozposlanie schváleného bloku. Každý tento krok posiela N správ. Problémy, ktoré z tohto dizajnu plynú sú:

- nemožnosť škálovať komisiu
- pevne stanovená komisia je náchylná na DoS útok
- prílišná „výrečnosť" protokolu, kde v jednom kole sa pošle 3N správ

Algorand je označovaný ako „pravý" PoS protokol, tiež označovaný názvom byzantská dohoda. Jeho architektonické výhody v porovnaní s *Tendermint* sú zjednodušenie návrhu bloku a obmeny komisie v každom kole. Návrh bloku je vyriešený tak, že v predchádzajúcom bloku sa nachádza tzv. maják (*beacon*), podľa ktorého si každý účastník vie zistiť či je, alebo nie je lídrom, resp., alternatívnym lídrom. Tento líder potom vytvorí blok a odošle ho do siete na schválenie. Schvaľovací proces rieši komisia, ktorá je volená obdobne ako líder pomocou „majáku". Týmto spôsobom je komisia dynamická a v konečnom dôsledku sa pošle len 2N+1 správ. Vďaka majákom vopred nevieme kto je líder, vieme ho len overiť ak už prijmeme blok, preto je veľmi náročné cieliť DoS útok na možných lídrov.

Navrhnutý protokol, ktorý táto práca implementuje, ide v optimalizácii ešte ďalej a odpadá potreba komisie. Ak niekto príjme blok, a tento blok je vygenerovaný lídrom ako v prípade *Algorand*-u, tak tento blok príjme. Aby nenastal problém s vytvorením paralelnej reťaze v *blockchain*-e, tak bol použitý *Casper*, ktorý uviedol kontrolný bod v histórii blokov, kde sa nová reťaz smie vytvoriť iba v prípade, ak rozdvojenie nastane na bloku, ktorý je vyšší (mladší) ako kontrolný bod. Vďaka tomuto riešeniu je pre útočníka ťažké vytvoriť paralelnú reťaz dosť dlhú na to, aby predbehla hlavnú reťaz tvorenú pravými lídrami.

Ďalšia časť navrhovaného protokolu je anonymizačná vrstva. Pri použití aplikácií tretích strán, ktoré anonymizujú účastníka v sieti vznikajú problémy, ktoré môžu odhaliť účastníkov v *blockchain*-e. Riziko odhalenia účastníka v protokole vieme znížiť tým, že natívnu implementáciu anonymizačného protokolu zahrnieme do nášho konsenzus protokolu. Pre tento prípad sme sa rozhodli použiť *Onion* (cibuľové) smerovanie správ v sieti. Tento prístup bol zvolený aj pre vysokú motiváciu účastníkov mať najlepšie pripojenie, čo v prípade použitia aplikácií tretích strán nie je zaručené.

Výsledky implementácie sa dajú považovať za úspech, pretože celá aplikácia je braná ako prototyp, ktorý bez výrazných optimalizácií dosiahol priepustnosť 760 tx/s (transakcií za sekundu). Pre porovnanie, *Algorand* dosahuje rýchlosť cca. 3000 tx/s. Výrazné spomalenie sme videli v prípade, ak pri prijímaní bloku hneď aj validujeme všetky transakcie v danom bloku. Toto spomalenie bolo pre slabú implementáciu knižnice, ktorá overovala podpisy.

Fast, Scalable and DoS-Resistant Proof-of-Stake Consensus Protocol Based on an Anonymization Layer

Declaration

Acknowledgements

I would like to thank my supervisor Ivan Homoliak for his supervision of this work and my consultant Martin Perešíni as well. Further, we acknowledge that this work is a part of the ongoing research in the security@FIT group, and it will be fully published with all the contributors: Lukas Helebrant, Ivan Homoliak, Kamil Malinka, and Peter Hanacek. Computational resources were supplied by the project "e-Infrastruktura CZ" (e-INFRA LM2018140) provided within the program Projects of Large Research, Development and Innovations Infrastructures. Lastly, I would like to thank my family and closest friends for the emotional and physical support they created in my academic path. They deserve the biggest 'Thank you!'.

Contents

1 Background and Preliminaries **3**
1.1 Preliminaries 3
1.2 Blockchains and their Principles 3
1.2.1 Participants 3
1.2.2 Distributed System Properties 4
1.3 Stacked Architecture of Blockchain 5
1.3.1 Network Layer 5
1.3.2 Consensus Layer 5
1.3.3 RSM Layer 6
1.3.4 Application Layer. 6

2 Proof-of-Stake Protocols **9**
2.1 Principles 9
2.2 Tendermint 9
2.3 Algorand 11
2.4 LaKSA 14
2.5 Casper 15
2.6 Summary 15

3 Attacks on PoS Protocols **17**
3.1 General Attacks 17
3.1.1 Blockchain attacks 17
3.1.2 Network Layer Attacks 19
3.2 Proof-of-Stake Attacks 21
3.3 Summary 23

4 Anonymization Techniques in Network Traffic **25**
4.1 Tor project 25
4.1.1 Onion Routing 25
4.1.2 Hidden Services 27
4.2 I2P Anonymous Network 28
4.2.1 Garlic Routing 28

5 Proposed Proof-of-Stake Protocol **30**
5.1 Consensus Protocol 30
5.1.1 Design Considerations 31
5.1.2 Initialization 32
5.1.3 Normal Operation 32

5.1.4 Incentives and Rewarding Scheme 34
5.1.5 Joining the Protocol 35
5.1.6 Churn of the Nodes 35
5.1.7 Forks 35
5.2 Anonymization Layer 36
5.2.1 Joining the network 36
5.2.2 Sending the messages 36
5.2.3 Relaying the messages 37

6 Proof-of-Concept Implementation **38**
6.1 Requirements 38
6.2 Use Cases 38
6.3 Structure 39
6.4 Class Hierarchy 40
6.5 Toolset 43
6.6 Used Libraries 44

7 Analysis of proposed Proof-of-Stake Protocol **46**
7.1 Testing Plan 46
7.2 Test Cases 47
7.3 Results 47

8 Conclusion **51**

Bibliography **52**

A Contents of the included storage media **57**

B Excel@FIT Material **58**

Chapter 1

Background and Preliminaries

This chapter will describe the ideas and basic building blocks of blockchain technology. We will explain the layers blockchain utilizes, how the distribution across the network works, and both the permission and permissionless modes. Then we will discuss different blockchain types, with a focus on their consensus layer. Finally, we will examine a few applications that utilize blockchain for their functions.

1.1 Preliminaries

The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'MAY', and 'OPTIONAL' in this document are to be interpreted as described in RFC 2119 [12].

1.2 Blockchains and their Principles

Blockchain is, in the most basic sense, a data structure that represents an append-only distributed ledger. The ledger consists of records called blocks that are immutably cryptographically connected. The individual blocks consist of a header and body. The cryptographic connection is created by a cryptographic signature in the block header. The signature is created by signing a hash of the previous and current blocks. The body of the block is a collection of records that are called transactions. A transaction is a unit of work performed within the blockchain. Because the blocks are immutably connected, they create a chain of blocks, blockchain as shown in Figure 1.1.

1.2.1 Participants

There are usually three types of participants or nodes present in the blockchain: *consensus*, *validating*, and *lightweight nodes* [38]. The consensus nodes are the essential part of the blockchain. They can read and write into the blockchain. The consensus nodes actively participate in the creation and verification of the blocks. They can also prevent malicious behavior by not appending invalid transactions or blacklisting malicious nodes.

The validation nodes can only do read operations on the blockchain, but they can validate the blockchain as the name suggests. They are not capable of mitigating the adversaries, but they can detect them.

The lightweight nodes depend on the validation and consensus nodes. They read only a small part of the blockchain, typically just the block headers that concert them.

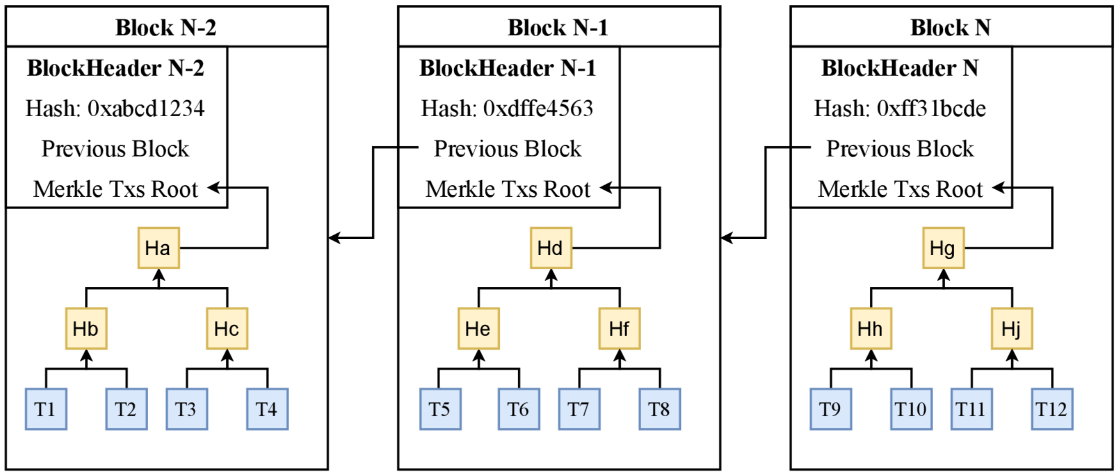


Figure 1.1: The figure shows an example of a blockchain data structure. The header contains a Merkle root hash of the Merkle transaction tree. The yellow blocks represent a hash value of a subtree. The blue blocks represent transactions numbered from 1 to 12. The number of transactions in a block depends on specific blockchain implementation. The first block in the blockchain is called the genesis block. The genesis block is typically hard-wired into implementation.

1.2.2 Distributed System Properties

Several important properties describe the consensus layer. These are *finality*, *liveness*, and *safety*.

Liveness - If a peer sends a transaction into the blockchain through a honest node, it will eventually spread across the network. After some time, it will occur in a published block, i.e., the transaction will process. And the whole process from receiving to processing the transaction represents liveness. If the transaction waits to be processed and every other transaction is processed instead, the transaction could starve. The starvation is an antonym to the liveness.

Safety is when an honest node accepts (or rejects) a transaction and all other honest nodes do the same. Usually, the consensus protocol is responsible to guarantee safety and liveness.

Finality or time to finality represents the sequence of blocks from genesis block up to block B that is unlikely to overturn. After several successive published blocks, the chance of overturn or a change to another chain quickly drops, which means that we can say that these blocks are final. There was only one incident with the rollback in Ethereum caused by [2].

Scalability represents a property that describes how hard is to join the consensus nodes. There is an implementation that does (Algorand) and does not scale (Tendermint).

Throughput describes how many transactions can be processed in a given time (typically transactions per second or blocks per second).

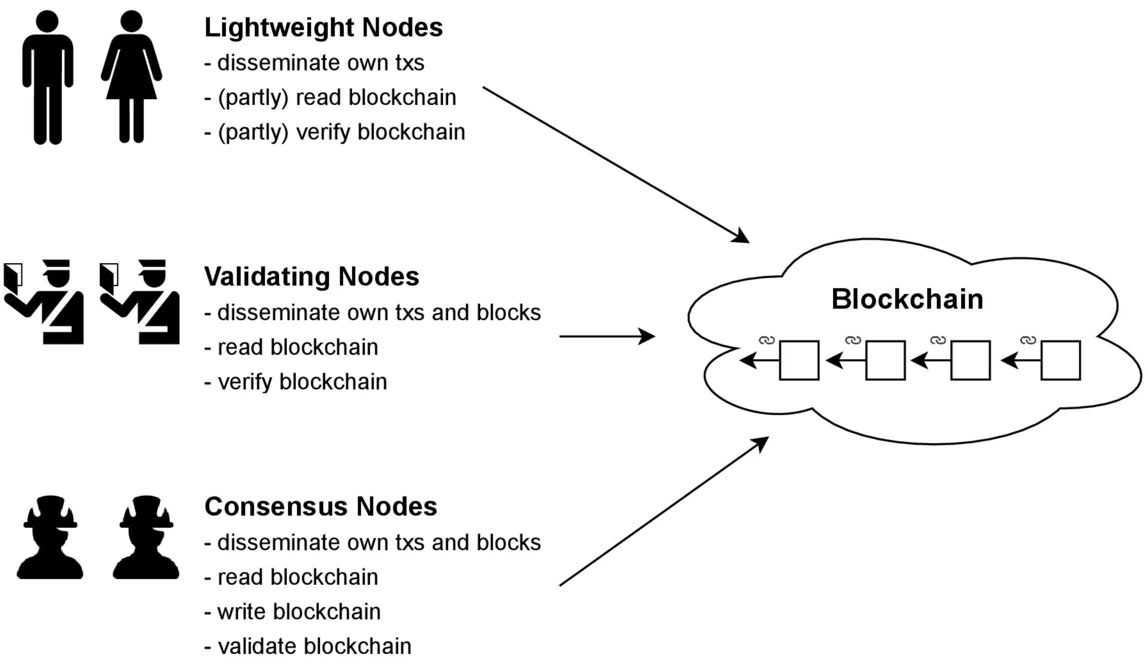


Figure 1.2: The figure shows an interaction of involved parties with the blockchain [38].

1.3 Stacked Architecture of Blockchain

The blockchain architecture that has been proposed in [66] can be imagined as a stacked model with four layers, the network, consensus, RSM, and application layer. The first layer is the *Network Layer*, which covers the network connectivity, i.e., communication protocols, data encoding, representation, and transfer across the network. The next layer is the *Consensus Layer*. This layer deals with ordering transactions and agreeing on a leader.

1.3.1 Network Layer

The blockchain can run on public or private networks. Both of those variants have some advantages and pitfalls. Blockchains on public networks have high availability, geographical decentralization, and low entry barrier. However, the network layer has centralized elements, i.e., IP addresses and ASes, managed by centralized entities like IANA (Internet Assigned Numbers Authority). Another problem is that external adversaries can compromise the chain. Private networks are usually routed through VPNs (Virtual Private Network), which means that the connections are centralized. Private networks have many advantages, such as low latency, high throughput, privacy, and resilience to external attacks. However, the disadvantages are that it is suitable only for permission blockchain, it is not resilient to insider threats, and requires a VPN for geographically spread participants.

1.3.2 Consensus Layer

The consensus layer deals with ordering transactions and agreeing on a leader. The layer has many variations in how nodes communicate and achieve consensus, such as *Proof-of-Resource*, *Proof-of-Stake*, *Byzantine Fault Tolerant* protocols. Nodes in *Proof-of-Resource* protocols must prove the spending of scarce resources in a lottery-based fashion [40].

There are many solutions om how to cover the crypto-tokens (can be named as *spending resources* as well):

1. Proof-of-Work spending computation power - every node is trying to find a hash of the current block with specific properties, e.g., Bitcoin, Ethereum
2. Proof-of-Space spending storage - where a service requestor must dedicate a significant amount of disk space as opposed to computation [26], e.g., SpaceCoin [54], SpaceMint [53]
3. Proof-of-Burn spent crypto-tokens - after sending money node must prove that it cannot use that money [42],
4. Proof-of-Retrievability combination and modification of previous types, e.g., Permacoin [48], PeerCoin [44]

Many of these protocols are categorized as the first generation of consensus protocols, and they are mostly based on Nakamoto Consensus [51]. Proof-of-Stake protocols are lottery-based and it must prove investment (stake) to participate in the consensus protocol. Nodes that want to be a leader must have a stake to be able elected as a leader. The stake can be obtained just from nodes that already have some stake. Real word implementations, e.g., Algorand [31], Tendermint [13], Peercoin [44], LaKSA [56].

Based on the way how new consensus nodes, we recognize permissioned, permissionless, or semi-permissionless blockchains. When the node first needs to obtain permission to join the consensus protocol, we say it is a permissioned one. The permission is obtained from centralized authority or authorities, while nodes usually have equal consensus power. When the node can connect without affecting the consensus, a blockchain is called permissionless. These are designed to run over the Internet. A node can connect to the permissionless blockchains as an anonymous node. The semi-permissionless blockchains are a combination of both. The permission to connect is based on the stake obtained from other consensus nodes in the network. The difference between permissionless and semi-permissionless is that the consensus power is based on how much stake the node has, and the consensus is based on the stake rather than on resources spent.

1.3.3 RSM Layer

The replicated state machine layer (RSM) is responsible for interpreting transactions into the blockchain ledger.

A replicated State Machine is a deterministic automaton that is replicated amongst processes such that it can function as a single automaton despite the failure of some processes [57]. Transactions and Smart Contracts are used as input for the RSM [32, 8]. Every change can be done if the consensus layer has issued a new block. After that, the RSM can interpret transactions from newly issued blocks. The transactions that are not yet processed are in a queue named mempool that is sorted by priority (height of a fee).

1.3.4 Application Layer.

The last one is the application layer representing user services, e.g., e-voting, wallets, and exchanges. The layer contains the most common end-user services. This layer can be divided into two groups with common functionalities designed to provide building blocks for higher-level applications [38].

Crypto-Tokens & Wallets is part of the basic building blocks of applications. This block provides crypto-currencies with a native crypto-tokens [50]. There are two types of secure wallets, Self-Sovereign Wallets (SSW) and Hosted Wallets (HW), to secure tokens [10, 27]. SSW users create local wallets where they can store private keys and directly interact with crypto-currencies using these private keys. The private keys can be stored in software wallets (My Ether Wallet[1]) in hardware wallets (e.g. Trezor[2]). HWs are managed by a centralized party that provides an interface to interact with the wallet and with the blockchain. By the way, how HW handles private keys we refer to them as a server-side or client-side wallet. In server-side the private keys, the centralized party has full control over private keys. The client-side wallet has private keys stored in the client's web browser.

Exchanges are places where users can exchange their crypto-tokens. Exchanges can be centralized or decentralized (DEX). The centralized design of centralized exchanges can cause security threats. The only countermeasure is to use DEX. Exchanges can be divided into multiple types based on the type of exchange (cross-chain or intra-chain).

A direct cross-chain atomic swap is between two parties in two different blockchains, where the two parties want to exchange their tokens using atomic swap [38]. To do so, they use the atomic swap or they can use contra-party DEX. Users can make the atomic swap with DEX, using it as a middleman. Intra-chain DEX requires parties to put offers into a blockchain, and the execution of the trade is done with smart contracts.

Filesystems are used as distributed data storage infrastructure. Used filesystems derived from peer-to-peer storage systems that persevere data with tokens. Data stored in the blockchain can be fully replicated with the ledger because the full content is stored in the blockchain, which means that we have full data replication, and extremely high durability. We can replicate data partially in the ledger, i.e., the data block is encoded with two numbers (k, n) where n represents total erasure shares and k represents a minimum number of shares to recover data [48]. Another way to partially replicate data is without a ledger. In this case, are used distributed hash tables are used as a decentralized data lookup service.

Identity Management, (IDM) provides binding from a public key to the identity of an entity. It can be named as Public Key Infrastructure (PKI) as well, and it has a few security goals, [29], i.e., accurate registration, identity retention, censorship resistance. The problem with PKI is that it is hard to design decentralized, secure, and human-readable infrastructure [67].

E-Voting tries to implement many of the security features of paper voting [29]. Decentralized e-voting protocol has multiple phases and requires multiparty computation executed by the voters [19, 21]. The main advantages of e-voting are immutability, higher availability, and public verifiability. An example of (first) voting protocol is the Open Voting Network, (OVN) [45], implemented using Ethereum smart contracts.

Notaries are responsible for providing proof of the existence of a document, also to vet and certify notarized documents [52].

An example can be notary based on Ethereum named EthNot [35]. The notary is based on smart contracts as well as IDM. Using the IDM the notary can verify the parties and using the smart contracts it can create the contracts that are made by vetting parties.

[1] `https://www.myetherwallet.com/`

[2] `https://trezor.io/`

Honorable mentions of other blockchain applications: **Oracles**, *Secure Timestamping*, *Reputation Systems*, *Data Provenance*, *Direct Trading*, *Escrows*, *Auctions.* We refer the reader to work [38] for a more detailed application overview.

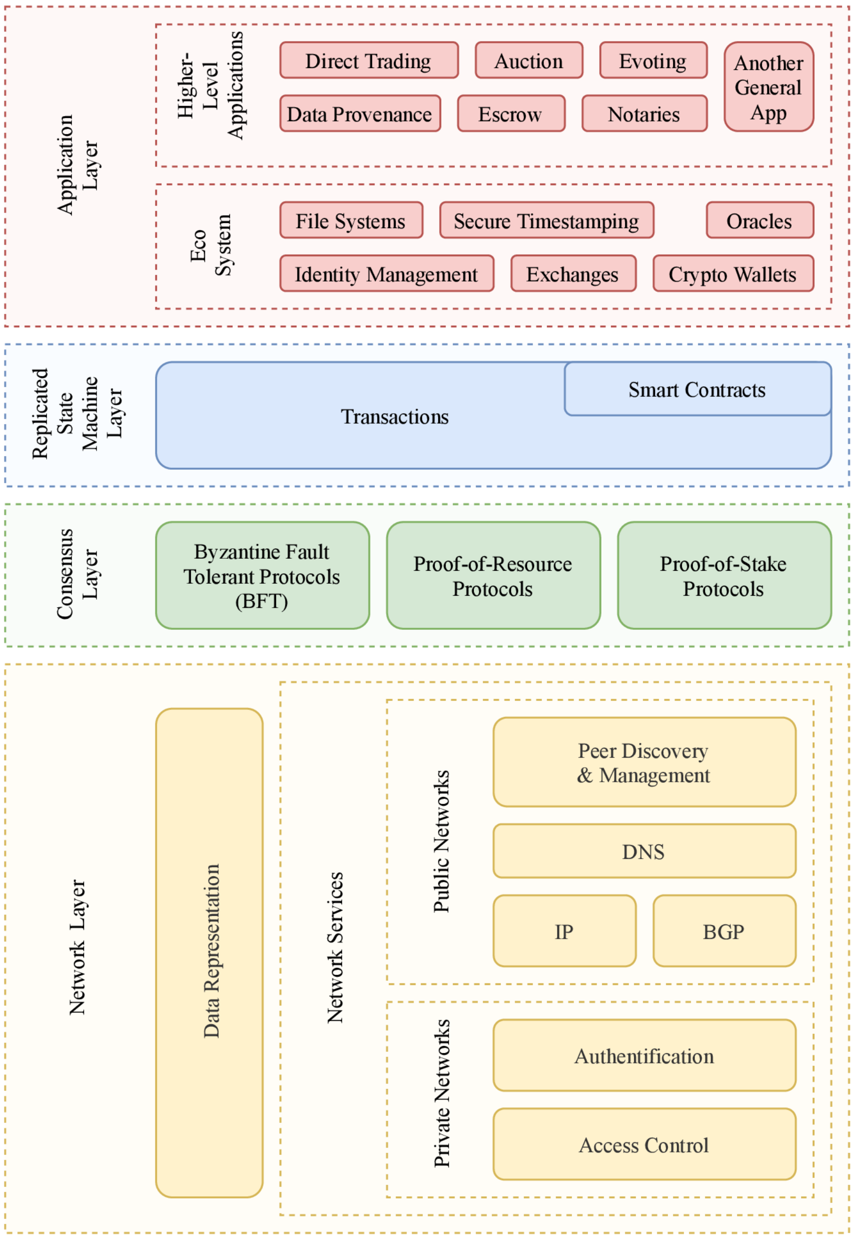


Figure 1.3: This figure shows the security stacked model of blockchain introduced by Homoliak et. al. [38]. We can see the network, consensus, RSM, and application layers.

Chapter 2

Proof-of-Stake Protocols

There are several well-known PoS protocols, Algorand [31], Tendermint [13]. In this chapter, we will have a closer look at their consensus layer and we will discuss their properties, i.e., liveness, fairness, safety, throughput, and scalability.

2.1 Principles

We already mentioned that there are several implementations of the consensus layer. The first implementation was a proof of concept called Bitcoin [51]. Other implementations were then derived or inspired by it, e.g., Proof-of-Stake (PoS) protocols [55]. The idea behind the PoS protocols is that to sign blocks, you have to be a leader of the current round. The node could be elected as a leader only if the node proved ownership of a certain amount of cryptotokens, called a stake. The stake is deposited in the blockchain as a security deposit. The security deposit was introduced by a paper called Slasher [15] due to the 'nothing-at-stake' attack. The node can obtain a stake just from nodes that already have some stake.

The election process is probabilistic and the chance for the node to be elected is proportional to the amount of stake the node has. After a consensus has been reached in the network, the newly accepted block is distributed across the network. Some protocols use to reward such nodes that were participating in reaching the consensus. Usually, these newly obtained cryptotokens are moved directly to the leader's stake [60].

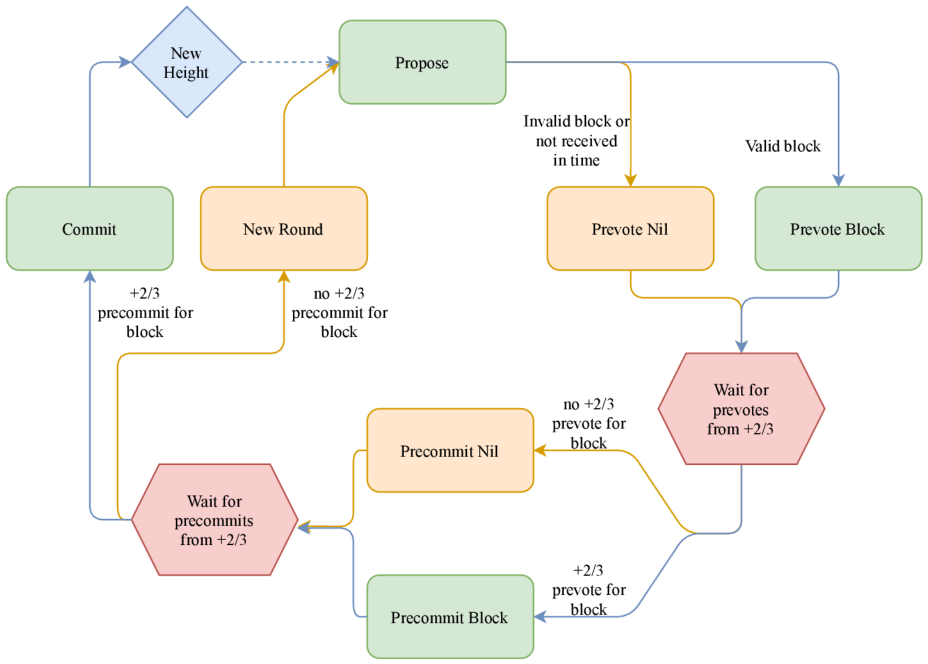
In the following we will elaborate on a few well-known PoS protocols and we will present their comparison.

2.2 Tendermint

Tendermint is a Byzantine Fault Tolerant (further BFT) protocol with elements of proof of stake [13]. The main goal of the protocol is to have a balance among throughput, security, and scalability. The whole platform is Open Source, written in Go, and the repository is hosted on GitHub[1]. Tendermint is commercially used[2] for cryptocurrency SDK named Cosmos Cash.

[1] https://github.com/tendermint/tendermint

[2] https://tendermint.com/



New Height
Propose
Invalid block or not received in time
Valid block
Commit
New Round
Prevote Nil
Prevote Block
+2/3 precommit for block
no +2/3 precommit for block
Wait for prevotes from +2/3
no +2/3 prevote for block
Precommit Nil
Wait for precommits from +2/3
+2/3 prevote for block
Precommit Block

Figure 2.1: An overview of Tendermint. After the proposal phase, validators can only continue if two thirds or more (+2/3) send their answers for the proposition or precommit. The dotted arrow extends the consensus into the atomic broadcast by moving to the next height[13].

Overview In the beginning, there is a set of validators that are identified by their public keys. Their responsibilities are to maintain the full copy of the replicated state, proposing new blocks, and voting on them. Every block has a parameter called height, which is an incrementing index. There is only one block of a certain height in the valid index. In every round, there can be only one valid block proposer. Block proposition is handled in turns, during which the validators vote for a block that will be committed. The new block's commitment can take multiple rounds due to network asynchrony or due to the halting of the network. Halting can happen in the case when a one-third or more of the validators are offline. Voting for the commitment is divided into two phases, where validators engage with each other. The process follows a simple locking mechanism that ensures the safety of the round and can be compromised only if a malicious coalition is formed with at least one-third of the validators [13].

Consensus In Tendermint, the consensus algorithm can be divided into three phases *Proposals*, *Votes*, *Locks*.

Each round starts with a *Proposal*. The selection of a proposer is deterministic, and the process uses round-robin to select the next proposer. There can only be one proposer in each round. When the proposer is selected, he will create a proposed block with a transaction from the local cache (mempool), and he broadcasts the block into the network. For Byzantine tolerance, a cycle proposer is necessary. Due to cycling proposers, we can ensure liveness because if one proposer does not pick some transaction, the others might [13].

The *voting* phase has two sub-phases. The first is called pre-vote and is initialized after validators receive the proposed block. If the validator does not receive the block in *ProposalTimeout* or the block is invalid, it votes for nil. If the validator pre-voted nil, it means that it is prepared to move to the next round of proposals. However, if he voted for the proposed block, it is ready to commit the proposed block. If the validator receives a polka[3] it is a justification of the network for the validator to broadcast a pre-commit vote for that block. On the other hand, when the validator receives nil-polka[4] it means validators did not receive a block or it was invalid, and thus pre-commit vote will be nil, to move to the next round. Sometimes there is a situation where the validator does not receive polka or nil-polka. This scenario means that the validator is not justified to commit the block and must pre-commit nil (move to the next round). If the result of pre-commit for a single block is $\frac{2}{3}$ votes supporting the block, the consensus nodes commit the block, set the round to 0, and they increase the height of the blockchain [13].

Locks are used to ensure safety across rounds. The locks prevent from committing two different blocks in two different rounds at the same height. The locks are built around polka resulting in two locking rules *Prevote-the-Lock* and *Unlock-on-Polka*. The lock mechanism ensures that the validator must pre-vote for the block he locked on. If the validator is a proposer then the validator proposes the block. Behavior like this is used to prevent validators to committing to multiple blocks in different rounds at the same height. Unlocking means to release the lock only if a locked validator sees a polka at round greater than that at which it locked. This way validators ensure liveness and do not compromise the safety of the network [13].

Throughput In the original thesis [13], there were three experiments focused on how many transactions per second can Tendermint achieve. Parameters were set accordingly: ProposalTimeout was set to 10 seconds, and all other timeout parameters were set to 1 millisecond. Additionally, all mempool activity was disabled [13]. We use as reference the second experiment which was run on multiple machines in one data center (latency is close to 0). The results can be seen in Figure 2.2

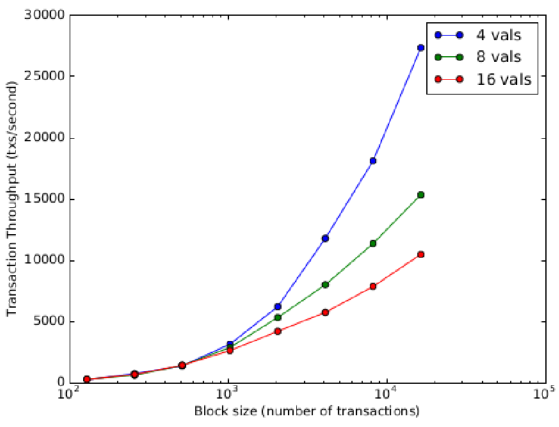
Safety Tendermint suffers from BFT architecture. As mentioned in Section 2.2 it has three parts where the second and the third phase is delaying the whole round. If two phases could be substituted for something more efficient, it would significantly increase the throughput. As mentioned in the original paper, the committee that verifies and commits block is fixed, which means that Tendermint does not scale very well. Another problem with Tendermint is that round-robin is used to elect a new leader in the round. The adversary can use that to commit a *Denial-of-Service* (DoS) attack by attacking specific nodes because the adversary knows which node will be a leader in the next round.

The adversary can misuse *DNS attacks* or *Routing attacks* because Tendermint relies on the proper network layer. If this assumption is broken, Tendermint halts its consensus layer, and it is waiting for at least $\frac{2}{3}$ of consensus nodes to go back online. *Eclipse attack* types of attacks are not possible in Tendermint because every consensus node is exchanging messages with every consensus node. Due to BFT like consensus layer, the *Nothing at stake* and *Grinding* attacks are relevant for Tendermint.

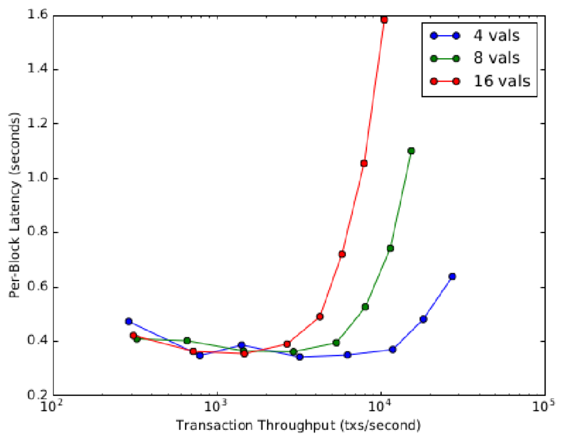
2.3 Algorand

Algorand is a pure Proof-of-Stake protocol that implements a Byzantine Agreement (BA). As the author says it is *„Algorand is a truly democratic and efficient way to implement*

[3] A set of more than $\frac{2}{3}$ pre-votes for a single block at a given round
[4] Like polka but pre-votes for nil



(a) The figure shows how [IH: the number je pocet; a number je niekolko, takze the] number of transactions per second depends on block size.



(b) Block latency according to number of txs.

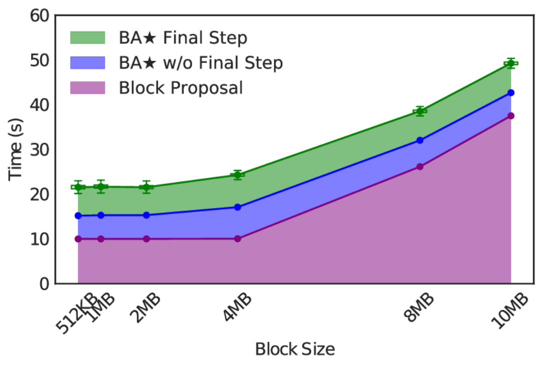
Figure 2.2: Colored lines represent how many validators are in the network. Looking at the graphs, we can see that Tendermint can handle tens of thousands of transactions per second [13].

public ledger.“ [31]. Similarly, to Tendermint, it is commercially used as cryptocurrency[5], but it can be extended for other usages.

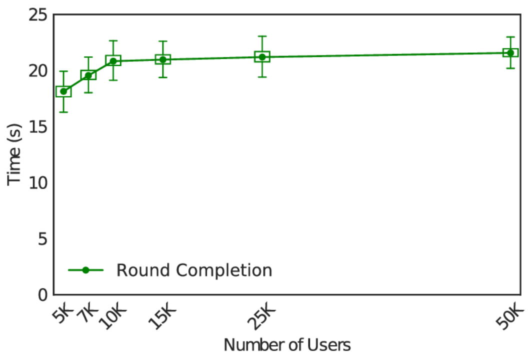
Overview Algorand's properties are very advantageous. It can withstand even many arbitrary users in a permissionless environment where users can join the network without any issues. In the permissioned environment, Algorand can run even better. A very advantageous property is that Algorand can work even in very adversarial environments, e.g., where the adversary can instantaneously corrupt any user any time he wants. This situation can be overcome in a permissionless environment, where $\frac{2}{3}$ of the money belongs to honest users or $\frac{2}{3}$ of honest users in a permissioned environment. Algorand can work even if the adversaries can control all corrupted users at once, and it can work even if the adversary can schedule messages, provided that each message from an honest user can be delivered to 95% of the honest users within a time λ_m [31]. Another super beneficial property is that Algorand uses minimal computational power, no matter how many users are connected to the network. Generation of a new block does not take longer than 10 minutes. The only limiting factor is network speed. The blockchain forking can happen only with a very small probability (i.e., less than one in a trillion), and thus users can rely on transactions as they appear in the ledger (small finality) [31].

Consensus To select a new leader of the round uses a public-key version of a keyed cryptographic hash called verifiable random function (VRF). Only the private key holder can compute the hash, but anyone with the public key can verify the correctness of the hash [46]. In Algorand is used VRF (cryptographic sortition) to select members that will acknowledge the leader (selected verifiers) and the leader itself. Every committee member candidate will locally compute VRF, and if the value from VRF is smaller than a threshold set by the previous block, the user becomes committee member. To enhance safety in the block, there

[5]https://www.algorand.com/



(a) Latency for one round of Algorand as a function of the block size [31].



(b) Latency for one round of Algorand, with 5,000 to50,000 users [31].

Figure 2.3: Colored lines represent how many validators are in the network. Looking at the graphs, we can see that Tendermint can handle tens of thousands of transactions per second [31]

is a carefully defined nounce that is used as input for VRF as well. With this value, we can overcome issues where adversaries can craft specific transactions into the last block and thus influence new leaders and newly selected verifiers. When the user realizes that he is the new leader for the next block, first he secretly assembles the new proposed block and then disseminates it together with his credentials [31]. After broadcasting, the block leader can peacefully die and the next problem lies on verifiers. They must run the Byzantine Agreement upon a newly received block. After their peer-2-peer agreement, they broadcast the block into the network.

Throughput By the article [7], Algorand can handle up to 500k peers in a network with a throughput of around 1000 transactions per second. The throughput is limited only by the speed that can messages travel around the network. The protocol scales independently of the number of total users participating [7]. Nevertheless, the finality of the block is relatively small.

Safety Algorand depends on the correct networking layer and thus adversary can attack on DNS and on routing as well. The beauty of Algorand is that if the block pushed to be the leader is delayed more than blocks of alternative leaders, it will be still backward accepted and the blockchain will rollback. This can be done only to a specific point (checkpoint in hones chain). Another attack that can adversary uses is the *Eclipse attack* or the *long-range attacks*, however, it will not be efficient against the consensus layer of Algorand. Hypothetically if the adversary can cooperate with the actions of multiple peers, then it is rather a pessimistic hypothesis because that type of coordination is challenging to achieve. It is much simpler to coordinate smaller groups of malicious users; however, coordinating these groups suffocates at the same problem when one adversary controls them all. Assuming that adversary can corrupt peers secretly and immediately is pessimistic [31]. By the Algorand characteristic VRF function, it is resistant against *grinding attack* and DoS on the leader as well. The *nothing at stake attack* in Algorand is not properly handled because as it says in Algorand paper: „... one possible way to avoid this trade-off, which we do not explore in Algorand, is to take the minimum of a user's current balance and the user's balance from the look-back block as the user's weight.“ [31].

2.4 LaKSA

LaKSA is derived from Algorand and it adopted ideas from DFINITY and Randhound [33, 62]. It is a proper Proof-of-Stake protocol with some BFT ideas [56]. It is not yet commercially used as the protocols above. It was developed to reduce drawbacks as high reward variance and long confirmation times. It enhances Algorand properties such as lightweight committee voting, it should be more robust and easily scalable than other protocols. In LaKSA, committee members are randomly and periodically sampled to vote for their preferred main chain views [56].

Consensus Each round of consensus consists of two steps. First, everyone obtains a random beacon from the previous block (in the first round from genesis block), used to elect leaders and voters (selected verifiers as in Algorand). In the first step, the node obtains a number of the stake it can use in that round. If the node has some stake to use in voting, it is called a voter. The other then assembles the vote and broadcast it to the network. If the node does not have the stake, it can verify the votes (if it is a legitimate voter if the vote is formatted correctly, authentic, and not from the future). After successful verification, a vote is added to the pending list of votes directly supporting the last block [56]. This list of votes creates a so-called virtual block. After some time, the second part of the consensus starts. In this part node checks if it is elected as a leader (from broadcasted random beacon). If the node is the leader, it will assemble a new block-based on votes from the first part. After block assembly, the block is broadcasted into the network. A node receiving a new block verifies multiple attributes - if the new block is authentic, the leader is legitimate, formatted correctly, points to an already existing block, etc. The electing method is called cryptographic sampling that was firstly presented in the LaKSA paper [56]. Random beacon is not specified in the original paper and it is proposed to investigate the suitability of beacons from DFINITY and RandHound algorithms [33, 62]. The block commitment is decided by every peer individually. A peer computes the risk of the block commitment and probability that the target block can be reverted. If the risk and the probability are below a threshold, it will be committed.

Analysis As a result of the block commitment, there is a long finality due to the compute probability of chain reverting, etc. Fairness in LaKSA is a bit better than in Algorand. Every node that voted in one round gets a reward proportional to their stake. Safety is similar to Algorand with honest money majority and honest users majority. Another weak spot in this protocol is the beacon implementation. If it is poorly implemented, adversary can affect the beacon towards the advantage of malicious nodes using *grinding attack*. *Breaking network assumptions* can only affect the blockchain if the committee to accept published blocks is not fully operational. This means that the whole committee must sign the block. In other case it will be waiting till the committee will be fully functional (in next round). The long-range attack will suffer on the same thing as in Algorand. It is almost impossible to cooperate with a large group of malicious nodes. Due to the committee's acceptance of a new block, it is very hard to create separate chains and do the *nothing at stake attack*.

2.5 Casper

Casper is just a gadget, from the BFT school of PoS, that developed some ideas on how to sooner finalize blocks and choosing the honest chain. It was introduced by Ethereum Foundation [16]. It is just an overlay and it does not implement the proposed mechanism. It introduced new properties to blockchain:

Accountability that will penalize any validator who has violated some network rules. In this case, the penalty is the violator's whole deposit. This is property is useful in PoS protocols rather than in PoW because in PoS this way violator cannot be a leader without any stake involved.

Dynamic Validators are a useful property because validators can join and leave the verification process. When the node wants to be the validator it will send a message to the blockchain. The validator can start validation after some checkpoints passed in the blockchain. To leave, he will send the message once more with information that he is leaving. However, he must verify the block for some checkpoints afterward. The specific numbers differ from implementation to implementation, but in the Casper paper, it is shown as 2 checkpoints to join, 2 checkpoints to leave [16]. If the verifiers left without a proper goodbye, it will be permanently banned from verifying the chain. To be able to verify, you need to deposit some cryptotokens like in PoS to be able to become the leader. However, here everyone who deposits the tokens can be a verifier.

Defenses against long-range revision attacks as well as attacks where $\frac{1}{3}$ of the network are unreachable were introduced as well. However long-range attacks and their mitigation are described in Chapter 3.

Modular overlay design approach enables to implement Casper into existing protocols such as any PoW or PoS.

Casper's identification of the honest chain is based on finding the highest checkpoint in the block tree. In this behavior are some vulnerabilities, that are described in Chapter 3.

2.6 Summary

in Table 2.1 is side by side comparison of PoS protocols with their properties. Casper is the only framework for blockchains that decreases time to finality, so other properties are not evaluated. The best PoS protocol with the biggest throughput is Algorand. Tendermint's disadvantage is when there is not enough validation nodes online, the whole network halts itself and can start when there is enough users to make consensus.

	Liveness	**Throughput**	**Finality**	**Scalability**
Tendermint	Eventualy every tx will be processed.	Relatively low throughput due to 2 phase consensus of 2/3 nodes	Blocks are almost instantly finalized	Very hard, there is still the same set of verifying nodes
Algorand	Eventualy every tx will be processed.	Relatively large throughput due to fast block broadcasting	Finalized blocks are only those which are located before checkpoint	Simple scalability based on stake transfer
LaKSA	Similar as Algorand	Similar as Algorand	Due to fixed committee, finality is lower than in Algorand	Simple scalability based on stake transfer
Casper	Depends on the chosen proposal mechanism.	Could not evaluate	Finalized blocks are until checkpoint	Could not evaluate

Table 2.1: Side by side comparison of PoS protocols and their properties. Details are described in sections above.

Chapter 3

Attacks on PoS Protocols

In this chapter, we will describe known attacks on blockchain as a whole ecosystem and proof-of-stake specific attacks as well. The security of the protocols mentioned in this chapter will be concluded in Table 3.2.

3.1 General Attacks

General attacks describe the most common attacks that are not dependent on any specific blockchain protocol but the general behavior of these protocols and on technologies that blockchain depends on.

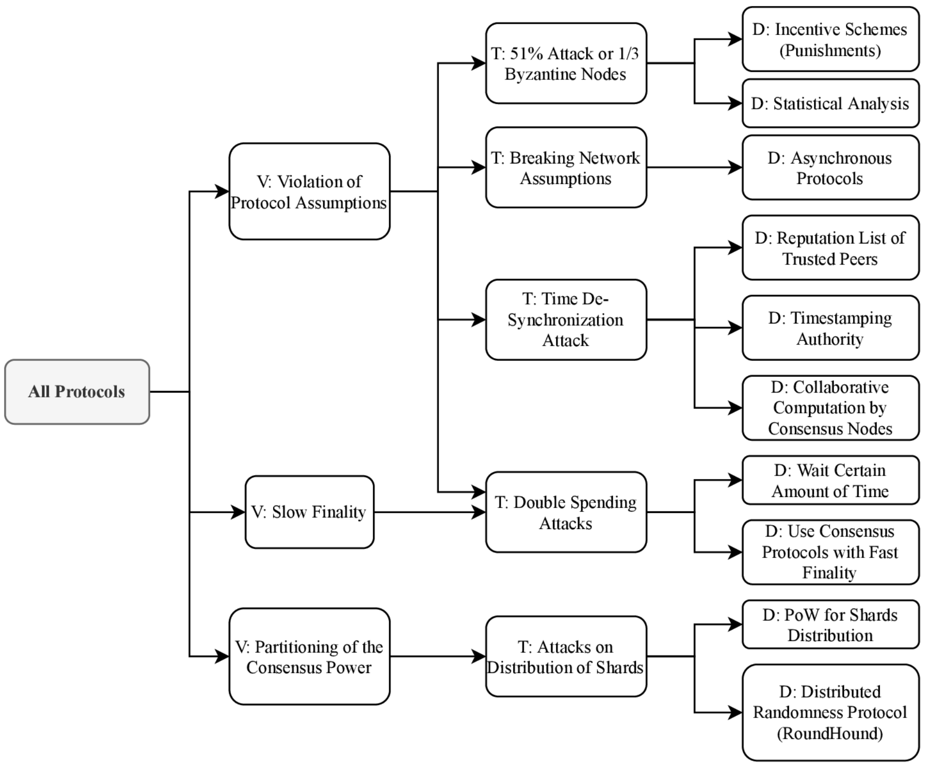
3.1.1 Blockchain attacks

These attacks are threats for all blockchain protocols not just any type like PoS or PoR.

Double-Spending Attack is possible due to the creation of two or more conflicting blocks with the same height. This behavior results in forking, and due to forking some cryptotokens can be spent in both blocks. However, only one block can be used in an honest chain. To mitigate this attack the transaction can be counted when the block is settled. The time to settle can vary and it is called finality. Finality can be shortened with protocols like BFT where finality is close to zero [20].

Attacks on Shards. To describe this attack we need to say what the sharding means in this context. Sharding is when the whole blockchain network is partitioned into subsets of consensus nodes that are cooperating. The cooperation is established as an agreement on which transactions will be accepted or rejected. Shards are working in parallel and it can increase throughput and scalability of the whole blockchain network. However, it is a threat because the shard can have only a few nodes, and it can be easier for an adversary to attack these nodes. If the attacker compromises many shards, it can compromise the whole network [24, 47]. Mitigation to this attack is to randomly distributes nodes among shards. Sharding attacks can lead to replay attacks [61].

Time De-Synchronization Attacks are used to inject invalid timestamps that are used as median or mean values computed from peers. This median is usually appended into the block header, and the receiving node can validate the block if it is not too old. The attacker can exploit this approach by connecting a significant number of peers into the network that will change the mean (median) time. After injection network can discard even valid blocks



All Protocols
V: Violation of Protocol Assumptions
V: Slow Finality
V: Partitioning of the Consensus Power
T: 51% Attack or 1/3 Byzantine Nodes
T: Breaking Network Assumptions
T: Time De-Synchronization Attack
T: Double Spending Attacks
T: Attacks on Distribution of Shards
D: Incentive Schemes (Punishments)
D: Statistical Analysis
D: Asynchronous Protocols
D: Reputation List of Trusted Peers
D: Timestamping Authority
D: Collaborative Computation by Consensus Nodes
D: Wait Certain Amount of Time
D: Use Consensus Protocols with Fast Finality
D: PoW for Shards Distribution
D: Distributed Randomness Protocol (RoundHound)

Figure 3.1: Generic threats and defenses of the consensus layer [38].

due to their invalid time (timejacking) [11]. To mitigate time-jacking you can create a list of trusted peers that will be used authority for time-stamping [64].

Breaking Network Assumptions can be broken in protocols like BFT and PBFT where nodes depend on synchronous or partially synchronous message delivery. If it fails, protocol is unusable and can not be achieved consensus. This vulnerability leads to developing threshold-based BFT that can operate when at least X nodes are online and can communicate [49, 17].

Adversarial Centralization of Consensus Power is an attack that is trying to violate the assumption about the decentralized distribution of consensus power in the network. The typical example of the attacks from this category is *51% attacks* for PoR and *1/3 of Byzantine nodes* for the PoS protocols family. When a majority of the network (at least 51%) is controlled by an adversary, it is called a 51% attack. In the PoS family there is equivalent and it can even halt the whole network. For this disruption, it is only needed 1/3 of the nodes to be controlled by the adversary. To mitigate these issues there are plentiful solutions such as:

- schemes to reward honest nodes
- schemes to discourage/punish protocol violation
- statistical analysis of sudden anomalies in the history of consensus power distribution, that can be used in fork-choice rules in the consensus protocol

3.1.2 Network Layer Attacks

Network layer attacks are focused on the infrastructure on which blockchain depends. Those are typically attacks on network address translation, routing, or denial of service attacks.

DNS Attacks are targeted at blockchains that use hard-coded DNS seeders. Mitigation of this attack can be using DNSSEC. Describing attacks on DNS and their mitigation are not in the field of this thesis and thus we leave it for the reader. Examples can be found [5].

Routing Attacks can be described as the manipulation of routing the packets among nodes. Within manipulation, we can imagine eavesdropping, modification, or dropping packets. This can lead to network partitioning or in 51% attacks, selfish mining, and many more. Proof of this attack can be found here [4] and it shows that Blockchain is vulnerable to BGP attacks. To mitigate these attacks, there is work from the same author, Apostolaki et. al. [3] that creates a secure relay network that runs alongside the bitcoin network. More general mitigation is to use multi-homed nodes for route diversity (choosing nodes from another ASes). This diversified routing can be useful to broadcast and retrieve recent blocks.Another mitigation can be using BGPsec [39]. Another way to secure routing such as secure routing table, creating a trusted authority for routing nodes (probably not wanted in decentralized style of this thesis), or using self-certifying data for routing was introduced in [18].

Eclipse Attacks evolved from DNS and routing attacks [34, 68]. This attack is focused on hijacking all node's traffic to peers and the blockchain itself. The eclipsed node can unknowingly vote for the attacker's chain that implies that the adversary is partitioning the network. Additionally, the eclipse attack induces selfish mining and double-spending. Erebus [65] is another network partitioning attack that is part of the eclipse attacks. It has a small network footprint. Erebus attack influences the node's peer selection because Erebus floods the peer connection with the attacker's shadowed IP addresses. There is at least a handful of mitigation. One of them is proposed in Erebus's paper that there should be two lists of peers and those are already connected peers and new peers. This way, the attacker can fill only one table and the node is still operating normally. Another mitigation is that nodes can communicate with lightweight nodes over gossip protocol (out-of-band) [1]. Next is randomly choosing peers, which is the mitigation proposed in [34].

DoS Attack is the budget option of listed attacks. These attacks can be divided into two groups and those are *Resource DoS* and *Connectivity DoS*. Resource DoS are attacks that are targeted on node hardware, e.g., memory. An example of memory flooding can be a penny-flooding attack which will flood the memory pool with low-fee-transactions. This may lead to a system crash. To prevent this attack, we can set a minimum fee for the transaction which means that it will be more expensive for the adversary [58]. Another mitigation can be to receive only a predefined number of transactions in a fixed period from one peer. DoS attacks on the connectivity of consensus nodes may lead to loss of consensus power. This means that the node will not be rewarded because of not participating in consensus [37]. If the DoS is focused on verifying nodes, it may result in a disruption in services that are dependent on blockchain such as [59].

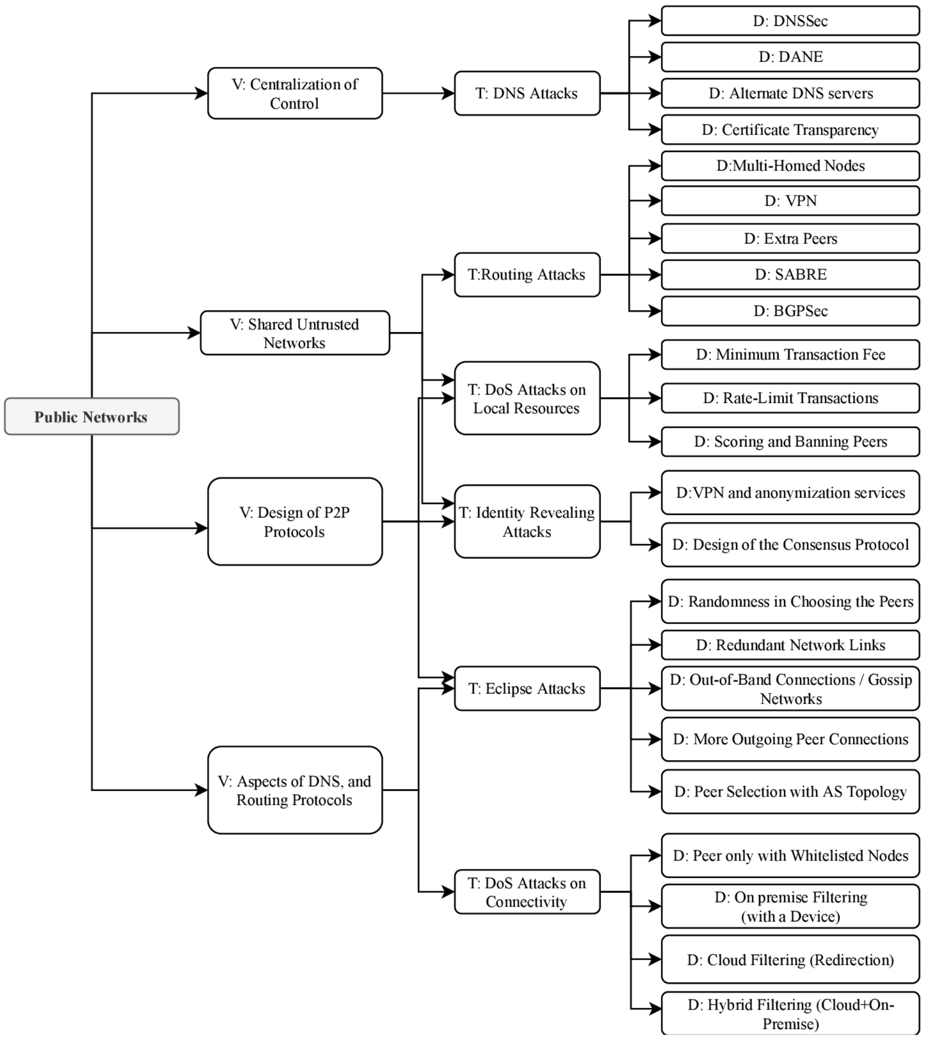

Public Networks
V: Centralization of Control
V: Shared Untrusted Networks
V: Design of P2P Protocols
V: Aspects of DNS, and Routing Protocols
T: DNS Attacks
T:Routing Attacks
T: DoS Attacks on Local Resources
T: Identity Revealing Attacks
T: Eclipse Attacks
T: DoS Attacks on Connectivity
D: DNSSec
D: DANE
D: Alternate DNS servers
D: Certificate Transparency
D:Multi-Homed Nodes
D: VPN
D: Extra Peers
D: SABRE
D: BGPSec
D: Minimum Transaction Fee
D: Rate-Limit Transactions
D: Scoring and Banning Peers
D:VPN and anonymization services
D: Design of the Consensus Protocol
D: Randomness in Choosing the Peers
D: Redundant Network Links
D: Out-of-Band Connections / Gossip Networks
D: More Outgoing Peer Connections
D: Peer Selection with AS Topology
D: Peer only with Whitelisted Nodes
D: On premise Filtering (with a Device)
D: Cloud Filtering (Redirection)
D: Hybrid Filtering (Cloud+On-Premise)

Figure 3.2: Vulnerabilities, Threats and Defenses in the Public Networks [38].

3.2 Proof-of-Stake Attacks

These attacks arise from PoS protocol architecture and they are specific and targeted just to them.

Nothing-at-Stake attack can be described as publishing two or more conflicting blocks at the same round and height. With this approach, a node can increase the probability to be rewarded. This problem is much worse in PoS protocols rather than in PoR because PoS protocols do not require any resources to spend. This unwanted behavior is increasing forking and extending time to finality. There are at least two approaches to fix this behavior. The first is *Checkpoint based solutions* such as [16, 41, 22, 44]. Casper deals with forking in the highest checkpoint manner. It means that it will choose a fork in which is the checkpoint higher than in any other fork. PPcoin deals with this issue that it will ignore any duplicate transaction until a successor block is received as an orphan block [44]. Casper can be used as an example of *deposit-based* solutions. This solution requires a deposit during some fixed time/round.

Grinding Attack is based on knowing a leader before the round starts. Based on this information, the attacker can manipulate the selection process to his advantage to be a leader in the next round, e.g., PoS protocols take hash from the previous round for the election, thus the attacker can create many blocks with different hashes to select the leader. The solution can be that the election of the leader can be done with the interaction of consensus nodes within some committee[43]. Another solution brought by Algorand is the usage of the VRF functions to determine the leader. The VRF function uses a beacon from a previous round to compute a value that is used as a threshold for determining the leader. If the internal value of a consensus node (e.g. public key) is smaller than the newly computed threshold from VRF, then the node is considered as a leader. Input is the user public key and the randomness bound to the previous block [31].

Denial of Service on a Leader/Committee is easy to perform on protocols where a committee or a leader is publicly determined before the round starts [43]. If it is the case, then the adversary can conduct DoS or DDoS to the leader or committee. This attack leads to a restart of the round and the adversary can repeat the attack until the adversary's desired nodes are elected. The solution to this attack can be found in Algorand, Ouroboros Praos, or Dfinity [31, 23, 33]. In Algorand, nodes privately determine if it is the leader of that round and if so it will broadcast the block candidate.

Long-Range Attack was described as in [14] or in [22]. In this attack, the adversary bribes a consensus node or steals a private key from the consensus node. The problem arises when the attacker has enough keys, then he can change the chain (rewrite the blockchain history). Another issue is that he can exchange the consensus node balance in fiat money. A variant of the long-range attack is stake-bleeding [30]. Mitigation of this attack is to lock the deposit for a longer time than the time participated in consensus [6]. Another mitigation is to do regular checkpoints in the blockchain that results in the irreversibility of the blockchain concerning the last checkpoint. Next mitigation is a context-sensitive transaction, i.e., the transaction adds a hash of a recent valid block to the transaction [30]. To mitigate this attack, we can use the key evolving technique that was introduced in [28].

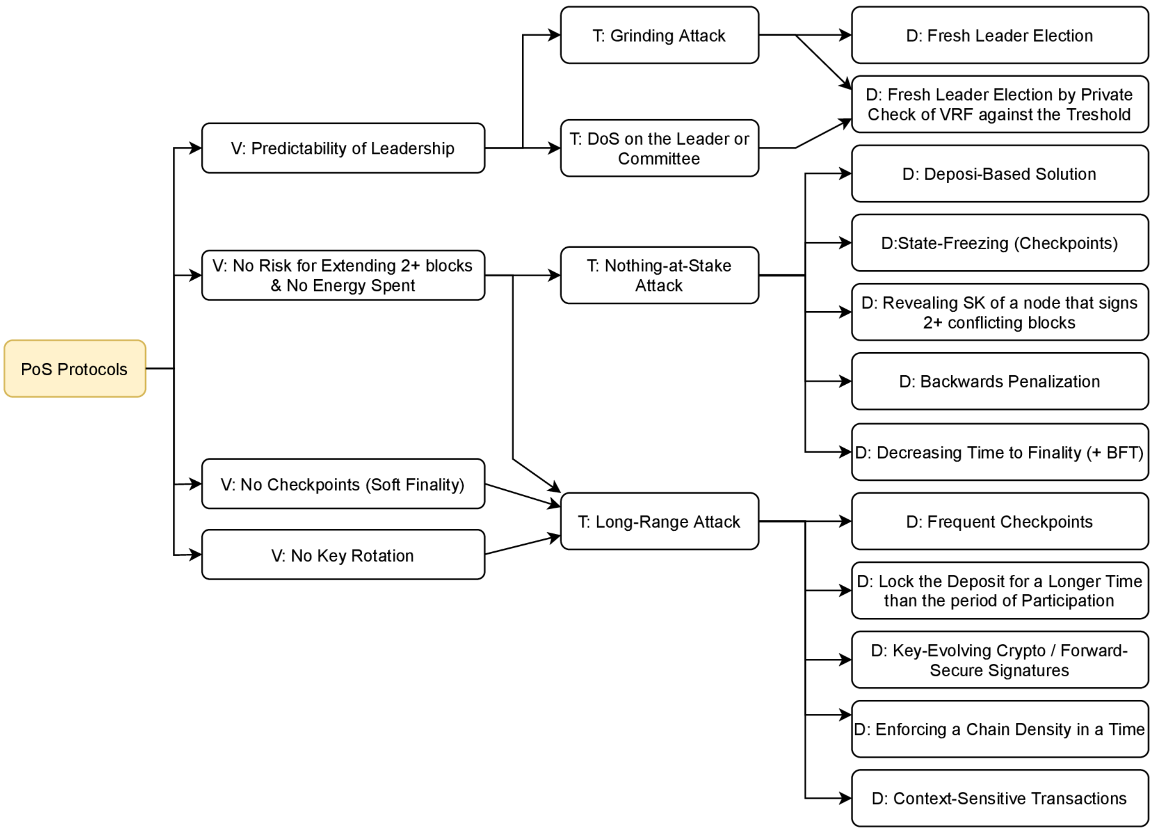

PoS Protocols
V: Predictability of Leadership
V: No Risk for Extending 2+ blocks & No Energy Spent
V: No Checkpoints (Soft Finality)
V: No Key Rotation
T: Grinding Attack
T: DoS on the Leader or Committee
T: Nothing-at-Stake Attack
T: Long-Range Attack
D: Fresh Leader Election
D: Fresh Leader Election by Private Check of VRF against the Treshold
D: Deposi-Based Solution
D:State-Freezing (Checkpoints)
D: Revealing SK of a node that signs 2+ conflicting blocks
D: Backwards Penalization
D: Decreasing Time to Finality (+ BFT)
D: Frequent Checkpoints
D: Lock the Deposit for a Longer Time than the period of Participation
D: Key-Evolving Crypto / Forward-Secure Signatures
D: Enforcing a Chain Density in a Time
D: Context-Sensitive Transactions

Figure 3.3: Vulnerabilities, threats, and defenses of PoS protocols (consensus layer) [38].

	Network Attacks			
	DNS	Routing	Eclipse	DoS
Tendermint	Not affected	Can delay messages	Relatively easy to eclipse consensus nodes due to small number of them, but Tendermint will halt if more then $\frac{1}{3}$ of nodes are not operational	Consensus nodes are known
Algorand	Not affected	Can delay messages	Very hard to implement	Not affected
LaKSA	Not affected	Can delay messages	Very hard to implement	Committee is known but hardened using Dandelion
Casper	Not affected	Can delay messages	Very hard to implement	X

Table 3.1: Network attacks on some Proof of Stake protocols.

3.3 Summary

We compare mitigations across various implementation in Table 3.1 and Table 3.2. From the properties, the most resistant implementation against DoS is Algorand and LaKSA if Dandelion [9] is part of the implementation.

	Proof of Stake attacks			
	Nothing at Stake	Grinding	DoS on Leader or Committee	Long-Range
Tendermint	Not affected, there is no other chain then honest chain	Not affected	Consensus nodes are known	Can affect protocol due relatively small number of consensus nodes
Algorand	Vulnerable, not implemented any mitigation	Not affected	Not affected	Not affected
LaKSA	Not affected	If beacon implementation is vulnerable then Adversary can attack on committee.	Committee is known but hardened using Dandelion	Not affected
Casper	Not affected, implemented penalization for nodes that violate nothing-at-stake	Not affected	X	As long as a client gains complete knowledge of the justified chain at a regular interval, it will not be susceptible to a long range attack [16].

Table 3.2: Proof of Stake specific attacks on some PoS protocols.

Chapter 4

Anonymization Techniques in Network Traffic

Demand for anonymity on the Internet or in the P2P[1] networks arose in the last years due to users' realization that their information is valuable for commercial companies and open for potential abuse. This problem may not sound terrible for average Internet users. However, when we start speaking about dissidents and whistle-blowers, it starts to make a lot of sense [69]. Based on this demand for anonymity, new technologies were created to anonymize users in the P2P networks such as Tor, I2P, GNU Net, etc. The guaranty is based on mathematical models that were proven correct and developed for many years. In this work, we will describe the two most common anonymization networks, Tor and I2P.

4.1 Tor project

Tor is the biggest anonymizing network to this date. This network was deployed in 2003 and design papers were introduced at the USENIX conference in 2004 [63]. In the last ten years, the whole network quadruplet[2], and the userbase is in time of writing this thesis around 2.5 million active users[3]. Tor is based on, so-called, „Onion Routing" that transfers all network data through the „Routing nodes" to the users.

Another anonymization technique introduced to the Tor network was so-called 'hidden services'. These services were designed to hide a service provider's location, i.e his IP address, and resist denial-of-service attacks. This technique is not bounded to the Tor and can be used with other anonymization technologies as well.

4.1.1 Onion Routing

Tor is just an implementation of Onion routing introduced in 1998. To show how this routing works imagine a situation where a person named Alice wants to send a message to a person named Bob. Firstly, Alice needs to obtain a list of all onion routers (OR). After that, A picks up three ORs from the obtained list, let's call them $R_1, R_2, ...R_3$ (typically they are three). The third step would be to create TLS tunnel between Alice and R_1. After the creation of the first tunnel, Alice will try to negotiate the second tunnel to R_2 through the already created tunnel. In the network, it will look like Alice is communicating with the

[1]Peer2Peer

[2]https://metrics.torproject.org/networksize.html

[3]https://metrics.torproject.org/userstats-relay-country.html

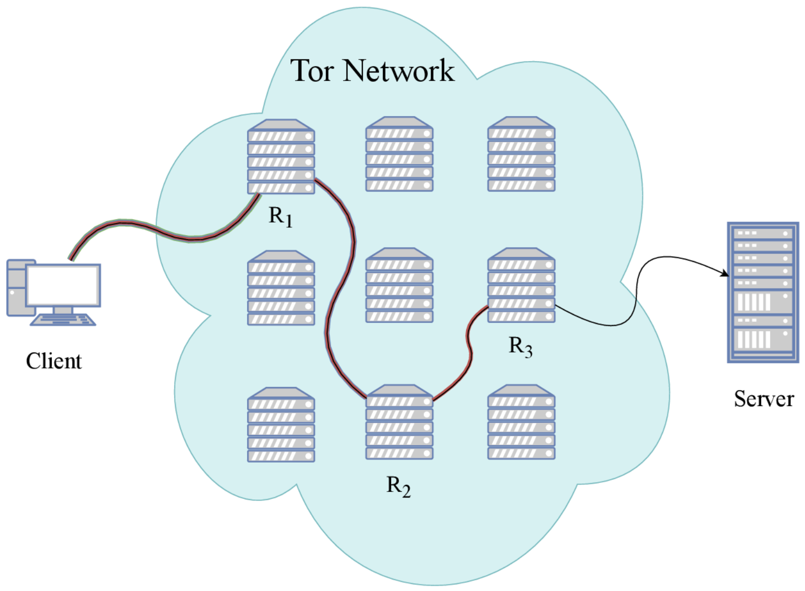


Figure 4.1: Example of a route in Tor network. Location of the Tor relay nodes can be anywhere around the globe, e.g., guard-node can be in Germany, next hop in the USA and the exit-node can be in Japan.

router R1 and not with router R_2 as shown in Figure 4.1. This procedure is then applied to R_3 and Bob as well. This way there is created a tunnel from Alice to Bob. The shown layering of tunnels looks like an onion and thus it is called onion routing. From Bob's perspective, he knows only the last router R3 called 'exit node'. Similarly, the R_1 is called an entry-node.

Let's say that k_i will represent a key shared with R_i and $E_k(m, T)$ is tuple (m, T) encrypted with key k. In our example, m is message that we want to send to target T. The message m' from Alice to Bob is encrypted accordingly:

$$m' = E_{k_1}(E_{k_2}(...(E_{k_n}(m, B), R_n)..., R_3), R_2)$$

The message m' is sent to OR R_1. When R_1 receives the message it will decipher it with the key $k1$. From the deciphered message OR R_1 knows where it should send the deciphered message. The deciphered message looks like this:

$$m'' = E_{k_2}(...(E_{k_n}(m, B), R_n)..., R_3)$$

Tor implements the onion routing with small improvements. The main difference is how Tor picks ORs. ORs are labeled by their network properties (fast and stable) and these labels are stored with their address in the list.

- An OR is labeled fast if its bandwidth is higher than the median of all routers in the network.
- An OR is labeled stable if its availability is higher than the median of all routers in the network.

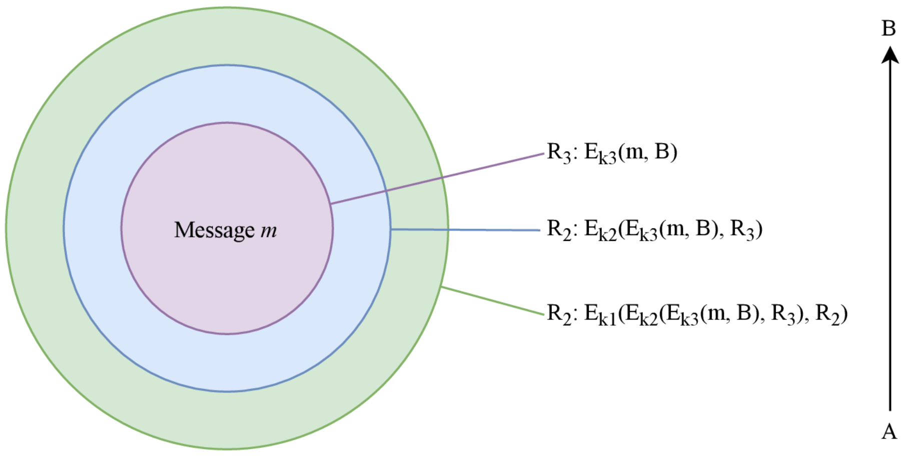

Message m
R_3: $E_{k3}(m, B)$
R_2: $E_{k2}(E_{k3}(m, B), R_3)$
R_2: $E_{k1}(E_{k2}(E_{k3}(m, B), R_3), R_2)$
B
A

Figure 4.2: Message encapsulation in Tor protocol

When Alice wants to create a route to Bob, she must pick an R_1 that is stable and fast. Other ORs will be picked up to target the optimal network speed and latency. Another significant difference is during the tunnel creation. In Tor, the second TLS connection is not created directly but through the R_1 and vice versa for R_3 and Bob. Together with the route creation, there is created a unique session key for every session created. The message encapsulation is illustrated in Figure 4.2.

4.1.2 Hidden Services

A hidden service is any regular service that can be only accessed through the Tor network and the connection is end-to-end encrypted. The important thing that hidden services introduced is the idea that the client does not know the address of the server and vice versa. The main difference between a hidden service and a classic service is that we know the address of the classic service. However, when we are connecting to the hidden service we only know the index to the DHT where are located introduction points.

The architecture of hidden service consists of:

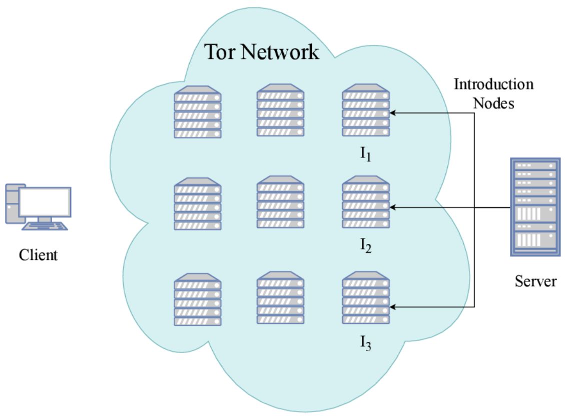
1. The hidden service (server) chooses three onion routers as 'introduction points' (server will create an onion tunnel to the introduction point).
2. The client will use the onion address to ask DHT for introduction points.
3. The client will select one onion router as a rendezvous point, create an onion route to the router and create the first part of a handshake. Afterward, he sends the address of the rendezvous point and the partial handshake to the introduction point.
4. The introduction point will send this information to the server.
5. The server will decide if he wants to connect to the rendezvous point. If yes, he will create an onion route to the rendezvous point and finish the handshake.
6. The rendezvous point will send the handshake to the client to verify it. if everything is in check the rendezvous point will connect these two onion routes. Thus the client and the server can exchange messages in a completely anonymous way.

4.2 I2P Anonymous Network

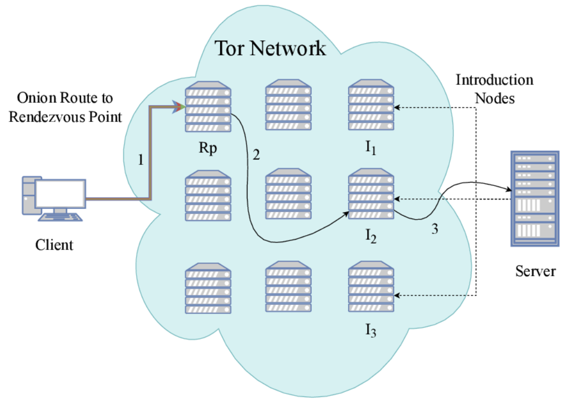
Invisible Internet Project is sometimes called by acronym I2P. It was introduced to the world in the Master thesis by Roger Dingledine [25]. I2P is based on the garlic routing and that is derived from the Onion routing.

4.2.1 Garlic Routing

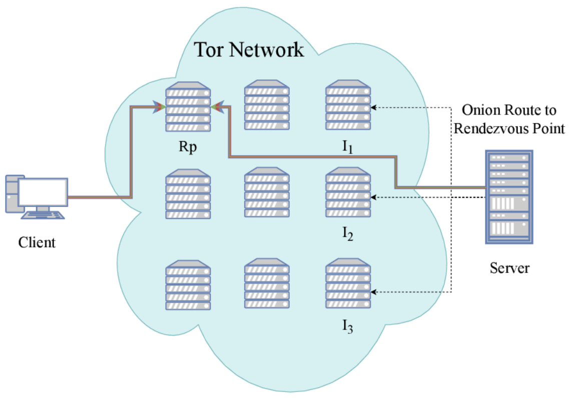
The extension to the onion routing is that the garlic routing is bundling multiple messages together. Every bundle is called a bulb. Every clove in a bulb has its delivery information. I2P implements delaying messages, which makes timing attacks from Tor not relevant anymore. Another extension is that path or tunnel is unidirectional instead of bidirectional in Tor. Unidirectional tunnels imply that every subject in the network must create two tunnels (inbound and outbound tunnel). This way of sending messages using different tunnels will diversify traffic in the network more.



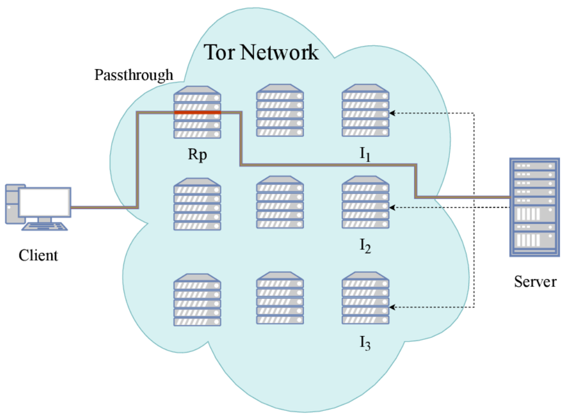
(a) Service will pick some nodes (in our case three) and marks them as introduction nodes.



(b) A client will create an onion route to a node (that he selected), and he marks it as a rendezvous node. Next, the client will send the address of the rendezvous point to one of the introduction points. The introduction point will transport that information to the service.



(c) If the service accepts the connection request, then the service will create an onion route to the rendezvous point. The rendezvous point will transfer all needed information to the client.



(d) When everything is in check, the rendezvous point is used as a proxy to transfer messages between the client and the service.

Figure 4.3: This figure shows a schema of connection establishing between a client and a hidden service.

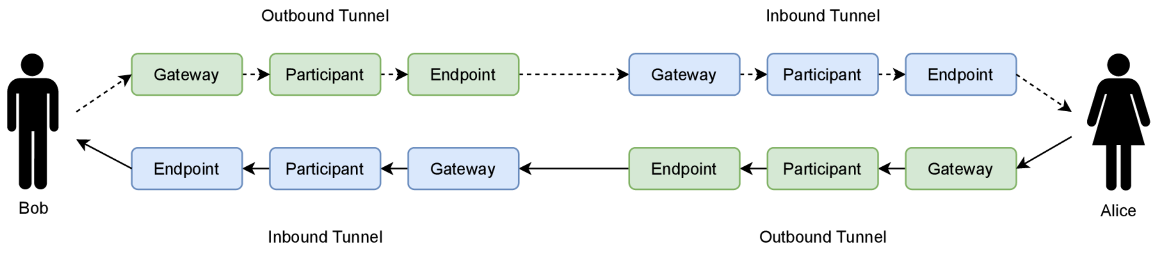


Figure 4.4: I2P illustrated tunnels

Chapter 5

Proposed Proof-of-Stake Protocol

This chapter will focus on laying a high-level overview on the proposed PoS protocol. The new PoS protocol will have just one leader in a round and he will unambiguously determine the leader of the next round before the round starts. Another emphasis in this protocol is to be DoS resistant. To achieve DoS resistance, the anonymization layer should be implemented right in the network layer of this protocol.

At the beginning of the protocol, we assume a genesis list of all initial participants with the stake distributed by the Initial Coin Offering (ICO). The position of the node in the ICO will determine her ID, which will never change. Every node needs to establish a connection to N randomly selected nodes, using the anonymization layer (see Section 5.2).

All participants produce a random seed rand of the first round using the RoundHound protocol [62]. The random seed unambiguously determines the leader of the first round. The leader is selected from the list of all participants in the blockchain. The unambiguity is guaranteed by the combination of the VRF and stake-weighted probabilistic leader election (SWEPLE). This principle is iteratively used in each round and thus it generates a sequence of block leaders who are known only one round ahead. Thanks to the network anonymization layer these block leaders cannot be DoS-ed. In our case, our protocol could be used for a smart contract platform, however, in this work we consider the use case of cryptocurrency, such as Bitcoin. In the following section, we will describe the consensus protocol and the anonymization technique presented at the consensus and network layer, respectively.

5.1 Consensus Protocol

The proposed protocol creates and extends its blockchain, an append only structure consisting of linked blocks Figure 5.1. The block consist of aggregated transactions and block header created by the leader of the current round. The block header consists of these parts:

- **ID** – the counter of all blocks,
- **hPrev** – the hash of the previous block's header in the blockchain,
- **txsRoot** – the Merkle Patricia root of all transactions included in the block,
- **coinbase** – the public key (PK) of the node that is the leader of the block (it is used for the signature verification),
- **rand** the randomness of the round, which detemines the leader of the next round and/or alternative leaders if the main one is not available. This value represents

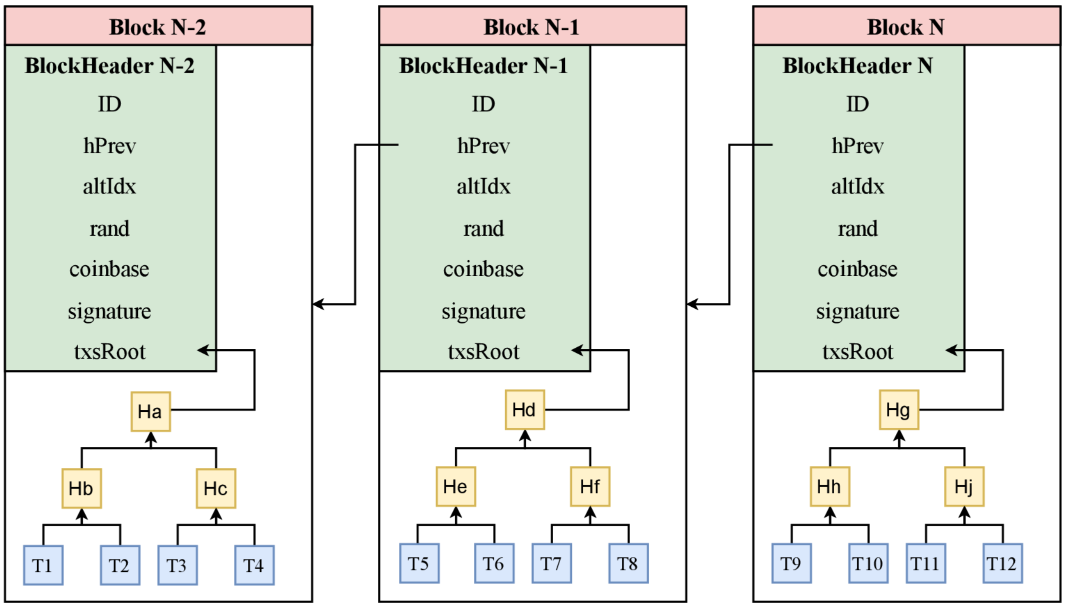


Figure 5.1: Connection between block headers

a signature made by the leader of the current round on the randomness from the previous round at the input

- **altIdx** is the order of the laternative leader who created the current block. Hence, $altIdx = 0$ for the main leader of the round, $altIdx = 1, 2, 3, \ldots$ for the first alternative leader of the round, second one, etc.
- σ the signature of all above fields in the header

Second part of the block are transactions and those consists of:

- **dst** – the address of the recipient of the transaction
- **val** – the value sent from the sender to the recipient
- **fee** – the fee that is payed to the leader who creates the block
- **sig** – the signature of the sender

One may think that the transaction misses the sender. However, the sender's address will be computed from the transaction itself and the signature. To achieve this behavior the signing process must be done with a cryptographic function that can include the signer's public key into the signature.

5.1.1 Design Considerations

In our implementation of the proposed protocol, we do not need to have a history tree data structure. The history tree is used only when we need to guarantee the block order, but that's not our case. For our account-balance model is best suitable Merkle Patricia Tree for a global state (gs). We leverage the *secp256k1* cryptographic algorithm for signing, which is very well established among all blockchain networks.

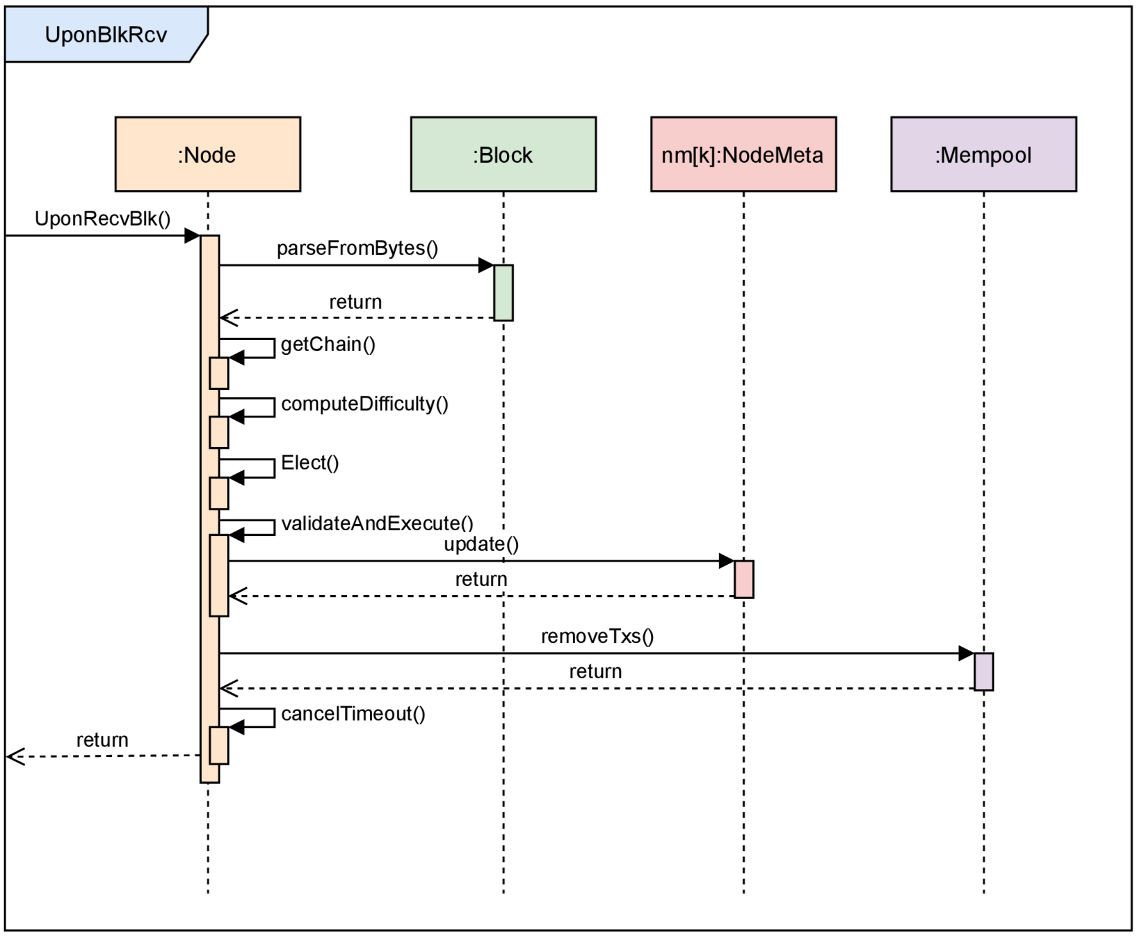


Figure 5.2: This figure shows interaction among objects in a function `UponBlkRcv`.

5.1.2 Initialization

We assume that a code of a full node contains a list of the genesis nodes whose stake is distributed according to ICO. Another assumption is that all of the crypto-tokens are put into stakes of individual full nodes. This assumption is derived from the idea that every node at the beginning wants to participate in the consensus protocol. Next thing is that every node will create an anonymized connection to N randomly selected nodes from the list of all nodes, referred to as transport peers. These peering nodes are used as transport relays for all messages and transactions that are sent/forwarded by the node.

5.1.3 Normal Operation

The normal operation of the consensus protocol is described in the Figure 5.3 and in Figure 5.2. When the round starts, the node resets the counter R of timeout expiration. Then a node checks if it's the leader of the current round based on randomness from the previous round. If the node is a leader of the current round, it will create a new block and broadcasts it afterward (see function *CreateBlock*). The newly created block consists of the set of transactions with the highest fee picked from mempool.

If the node is not a leader in the current round, the node will set synchronous timer τ^B on the expiration of valid block delivery and it waits on the reception of the block (see

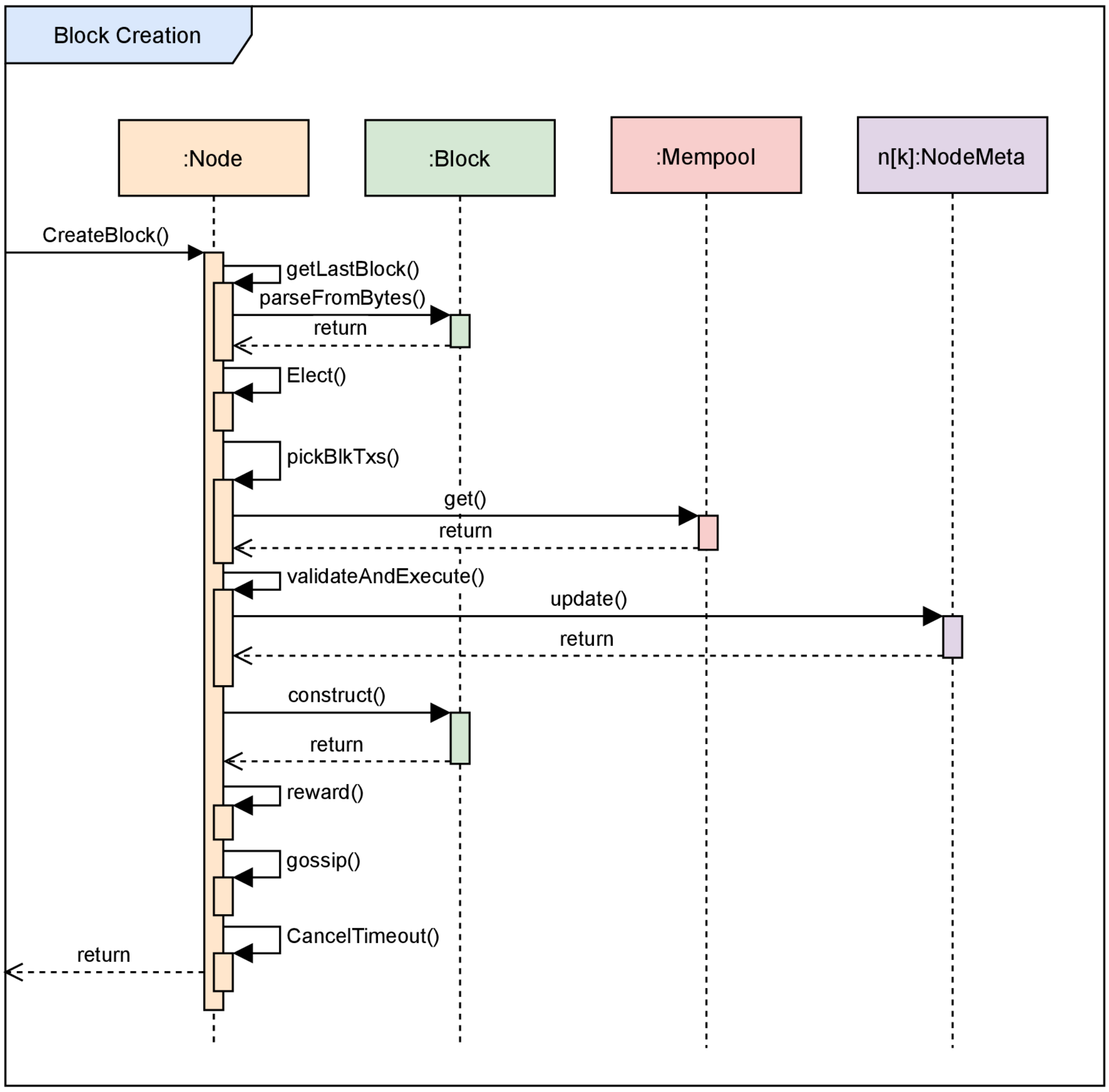


Figure 5.3: This figure shows interaction among objects in a function `CreateBlock`.

function *UponRecvBlk*). When a node receives a block, it will check the binding of the block on its predecessor (i.e., *hPrev*), and checks the signature of the block header using *coinbase* as a public key. Next, the node will validate the transactions, if they are correctly signed (public key is derived from signature and the transaction), and the values are in check with the current global state. Lastly, it will check the correctness of the elected leader. This check is done by comparing the output of the *Elect* function with the coinbase field. If the block is valid, then the node will reward the leader (see Section 5.1.4). Finally, the block is added into the database of all blocks.

If the node does not receive a block within the timeout τ^B, then it means that the main leader is off-line, or the block message was dropped by the network.[1] The node must deal with this situation as described in Section 5.1.6.

[1]Note that the second option is very unlikely due to a high redundancy in transport peers.

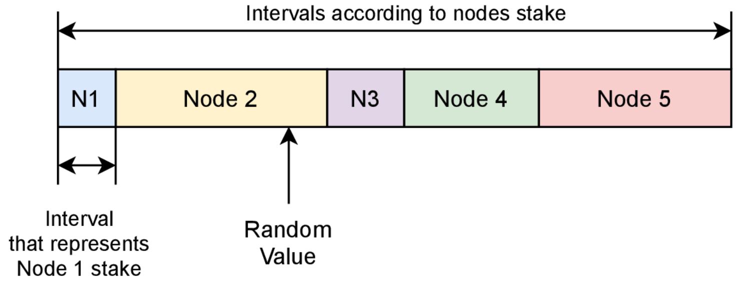

Intervals according to nodes stake
N1
Node 2
N3
Node 4
Node 5
Interval that represents Node 1 stake
Random Value

Figure 5.4: This figure shows how the elect function works. First, we create a list of intervals, which size is proportional to each node's stake. After that, we pick a random value that will be used to determine a first leader. After the election of a first leader, the value is hashed and used one more time to pick up the alternative leader. This process with an alternative leader can be iterated until the round increases.

If the node receives a transaction tx, it verifies the existence of the sender and the transaction's signature. After the initial checks, the node will check the balance of the sender with regards to the transferred amount and the transaction fee. When the transaction is valid, it will add the tx to the *mempool* of unprocessed transactions and gossips the tx to her consensus peers.

5.1.4 Incentives and Rewarding Scheme

The incentive scheme of our protocol is shown in the *Reward* function. This function rewards the leader with transaction fees and full reward R^F, while the alternative leaders are rewarded with partial reward R^P. We decided for this rewarding scheme to decrease reward variance that can occur when just the leader is rewarded [51].

Stake and Balance: The important aspect of the rewarding scheme is that the stake and the balance are separated. The overall value of the node's assets is computed as a sum of these two values. We define different rules for the stake and the balance, each having respective pros and cons. The node's balance has high liquidity and thus can be used by the node to make any transactions according to her will. The stake is quite the opposite, it has low liquidity, and it is meant as an investment that yields an 'interest rate'. We emulate the long term investment and its interest rate by two requirements:

- When a node want's to shift a part of the balance to the stake, the node has to wait for $K^{\#}$ blocks before the stake is taken into account at the *Elect* function, that is shown in Figure 5.4. This shift is made within a dedicated transaction (sender and receiver are the same).
- Similarly, when the node wants to shift part of the stake to the balance, the assets will be frozen for $S^{\#}$ blocks before it is reflected.

The values $S^{\#}$ and $K^{\#}$ must be big enough to penalize the liquidity of the node's crypto assets. Besides penalization, this scheme can mitigate the loss of the market's capital in a time of crisis. Further, there could be implemented different reward lock-out mechanisms. However, this is not implemented in this work.

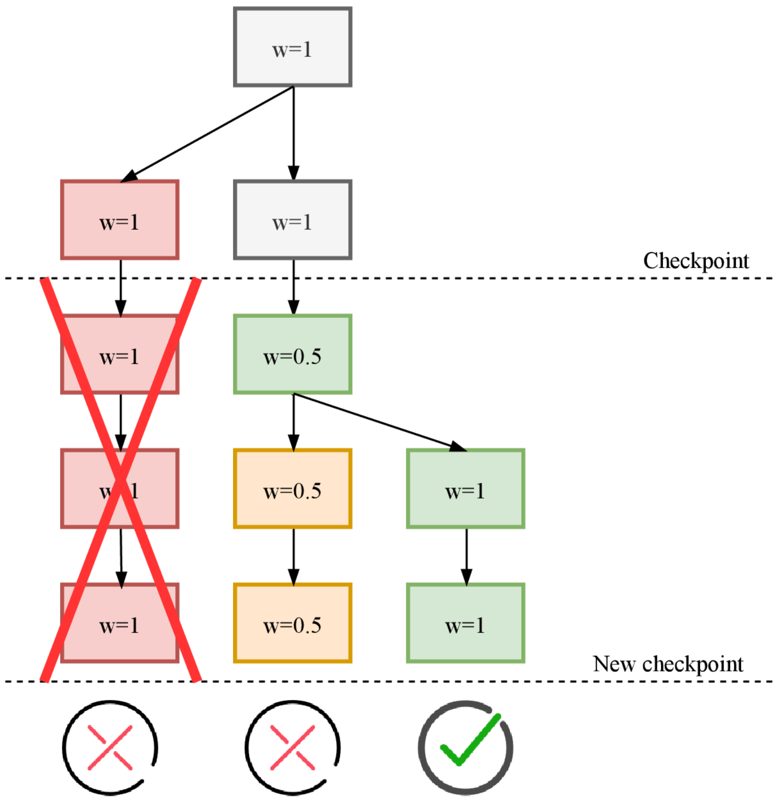


Figure 5.5: Check-pointing mechanisms

5.1.5 Joining the Protocol

To join the protocol, the node has to buy the balance from any of the existing nodes and then convert it into the stake.

5.1.6 Churn of the Nodes

Sometimes may occur a situation where the node will go offline (i.e., churn) and thus not produce a block when it is elected as a leader. If a node does not receive a block within the timeout τ^B, then it increases the number of timeout restarts R. After the incrementation, the node checks if it is an alternative leader and if so it will call the $CreateBlock$ function with R as the parameter. In the positive case, the node creates and gossips the block with adjusted $altIdx$ field to R (i.e., 1). Otherwise, it readjusts the timeout τ^B and waits until a valid block will be received from the next alternative leader.

5.1.7 Forks

Forks are inconsistencies formed from multiple parallel chains in the whole blockchain. When forks occur there come in handy a function $ChainDifficulty$. This function computes the chain difficulty from block's weight as shown in Figure 5.5. Every block is increasing the chain difficulty. The value depends on which leader produced that block. The leader contributes with a higher difficulty than the alternative leaders. Alternative leaders contribute with exponentially decayed difficulty, depending on the index of the alternative leader, i.e, higher the index lower the difficulty.

Although there might exist only a single strongest chain, which is created by main leaders only, sometimes a main leader of the round might not be available (see churn), thus a block of the round is created by an alternative leader. The offline node during the 'leadership' might return online and try to send created block retrospectively. That block can overturn the current strongest chain if it would be extended by the sequence of strong enough leaders. This will break the finality of the whole blockchain because the blocks would not acquire finalized state. To overcome this issue we introduce check-pointing to decrease the finalization of blocks. The checkpoint would be created after each C blocks, after which no node will accept the overturning of the strongest chain. This situation is illustrated in Figure 5.5 for $C = 3$, where w presents a difficulty of each block computed within $ChainDifficulty$.

5.2 Anonymization Layer

The anonymization is realized at the network layer and works as described in this chapter. Details of joining, sending, and relying messages are shown in Algorithm 1. This communication on network level won't be anonymized. It is not an issue, because it won't contain any peer identifier, which makes it impossible to tie peer identity (private key) to node identity (IP address).

5.2.1 Joining the network

When node N want's to join a network, she must do following steps:

1. New node N gets a list of IP addresses of all nodes from a directory (its trustworthiness may be ensured by multiple ways, e.g., a decentralized solution utilizing blockchain [36])

2. N selects n sets of m peers

3. For each set: Build a circuit consisting of m selected nodes (in chosen order $n_1, n_2, ...n_m$ where h_1 is closest to N in the circuit). To build a circuit, perform a key exchange with each of the selected nodes, for example by the approach proposed in [63].

4. The circuit has been established. N now shares secret key K_i with n_i for each i. Every further communication will be anonymized.

5.2.2 Sending the messages

Any message P wants to send (broadcast) is sent in the onion routing manner, i.e.:

1. The message is encoded so it can be received by the intended receivers

2. The message M is encrypted with K_i: $K_i(M)$

3. The result of the previous step is encrypted with $K(m - x)$ for $x = m - 1$ *downto* 1 and appended with IP of the $(m - x + 1)$th peer in the circuit, e.g., $(m = 3)$: $K1(p2, K2(p3, K3(M)))$

Algorithm 1: Anonymization layer interface

```
▷ Declaration of types and variables:
    route { $node_{n-1}$, $node_{n-2}$, ..., $node_0$},
    node { addr, key },
    addr { IP, port },
    this: the current node,
    routes: list of all routes that will be used in anonymization layer,
    Message: constructor of selected messages,
function joinNetwork(n_routes, m_nodes)
    allnodes ← getNodes();
    routes ← pickRoutes(n_routes, m_nodes);
    for route: routes do
        for node: route do
            exchangeKey(node);
    for route: routes do
        verifyRouteInitialization(route);
function SendMessage(dst, msg)
    for route : routes do
        relay_msg ← Message.Relay(dst, msg);
        for node : route do
            ct ← $\Sigma_{node.key}$.encrypt(msg);
            em ← Message.Encrypted(this.addr, ct);
            relay_msg ← Message.Relay(node.addr, em);
        gossip(route[−1], relay_msg);
function RelayMessage(src, relay_msg)
    msg_key ← findKey(nodes, src);
    msg ← $\Sigma_{msg\_key}$.decrypt(relay_msg);
    transport_key ← findKey(nodes, msg.dst);
    ct ← $\Sigma_{transport\_key}$.encrypt(msg);
    em ← Message.Encrypted(this.addr, ct);
    send(msg.dst, em);
```

5.2.3 Relaying the messages

- When a peer $p(n)$ in a circuit receives a message, it decrypts it using the key shared with P. It discovers the identity of $p(n+1)$ and sends the message (that is still encrypted by P using $K(n+1)$) to it.
- If there is no $p(n+1)$, the peer is an exit peer. It decrypts the message and gossips it [63].

Chapter 6

Proof-of-Concept Implementation

The requirements, use cases, particular parts, and whole proof-of-concept implementation architecture will be described in this chapter.

6.1 Requirements

We will require from the application to fulfill the following requirements:

1. The application will be designed with consideration of good OOP practices.
2. The application will consists of multiple layers:
   (a) Network layer
   - based on Transmission Control Protocol (TCP).
   - support of sending files larger than network Maximum Transmission Unit (MTU).

   (b) Anonymization layer
   - Based on onion routing.
   - The parameters of the onion routing could be adjusted, e.g., number of transport nodes, etc.

   (c) Consensus layer
   - Consensus will be implemented according to ongoing research and proposed paper by *Security@FIT* group

   (d) Replicated State Machine Layer

   (e) Presentation Layer
   - Command Line Interface (CLI).
   - Terminal User Interface (TUI).

   (f) Event logger

6.2 Use Cases

Before the creation of architectural design, we identified use cases that are a mandatory part of the creation of the application. First, we identified users of the system. Those are regular user, the node itself, and time. Each of these entities can call some procedures that are shown in Figure 6.1. The main part of the use cases for the user are use cases that show some information of the current local state of the blockchain. He can determine who is the next leader and start the statistics. He extends the node's use cases that are defined by the consensus protocol (create a block, process a block, gossip a block, gossip a transaction, increase the height). The time entity can only increase the round, create a new block or determine who is the next leader by running the elect method.

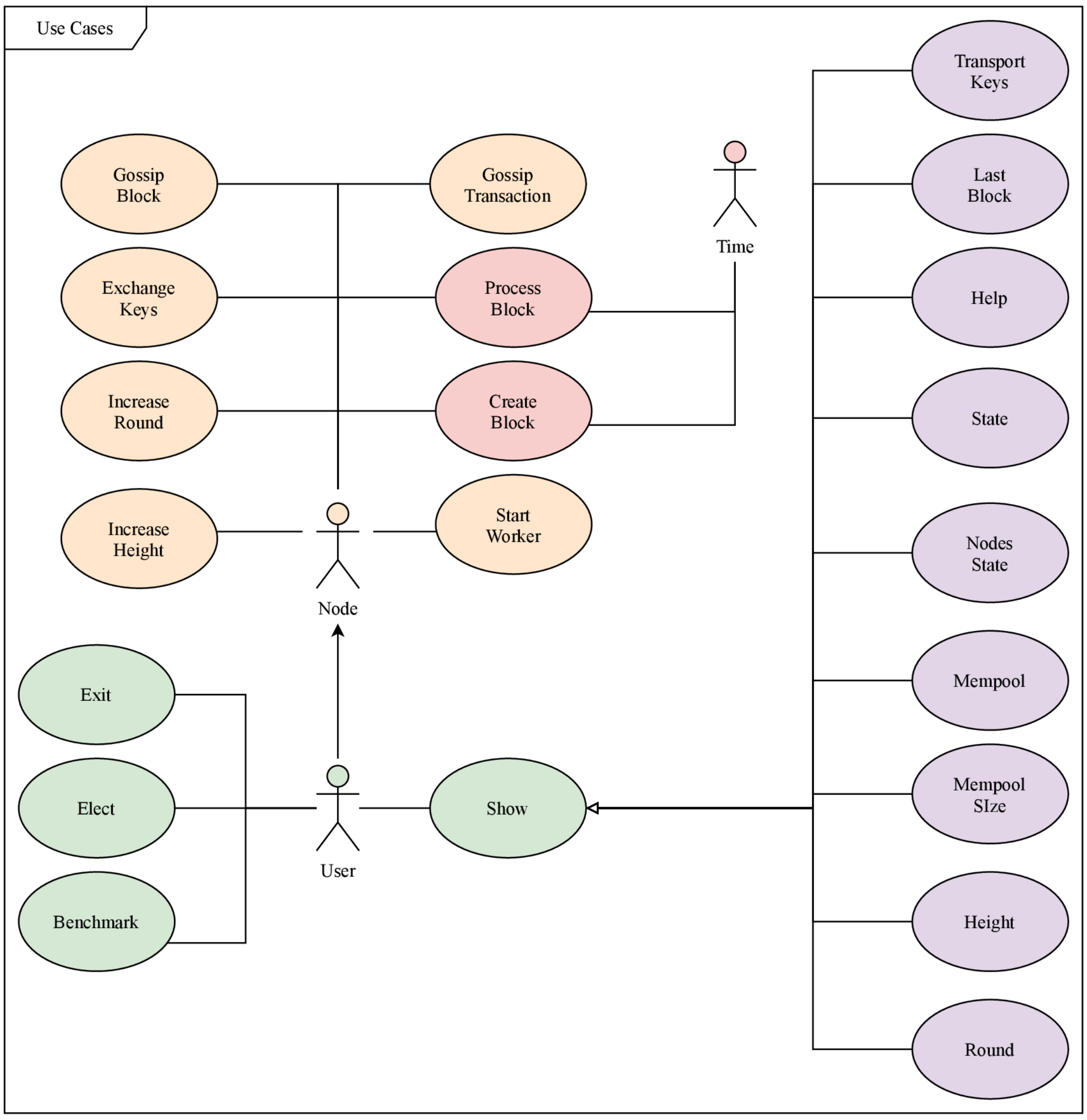


Figure 6.1: Identified use cases that are mandatory to implement.

6.3 Structure

The application structure is divided into multiple layers as introduced in paper [38]. The layers that are implemented in our solution are network, anonymization, RSM. To view the data from mentioned layers, we added a presentation layer. The dependencies among the layers is shown in Figure 6.2.

Network Layer. The Network layer handles the network communication. That means it creates the connection between nodes and creates a secure connection using the Diffie Hellman algorithm. The layer that depends on the network layer is the anonymization layer. This layer is responsible for creating routes inside the network, encryption, decryption, encapsulation of the messages according to onion routing. The next responsibility is to route the messages in the network.

Consensus Layer. In the consensus layer we implemented the proposed protocol by the set of authors mentioned in the introduction. This layer is responsible for ordering the transactions in the blockchain and for the acceptance of published blocks. The accepted transactions inside the blocks are interpreted in the RSM layer. Block creation and processing are handled in this layer as well.

Presentation Layer. Apart from these crucial layers, we have a Presentation layer that includes a shell module. The shell module is presenting information from the core layers mentioned earlier. It is used to execute commands that control the core layers. The shell is designed to be an interactive Text User Interface (TUI).

Others. Other modules that are not part of any layer are Transaction generator, Statistics, Logging, and Key Management. The transaction generator is only used when we are performing performance testing of the whole network. In that case, the generator is creating transactions that are inserted into the mempool. This generator is one of the two ways how to insert transactions inside the node's mempool.

The statistics are part of the performance testing, and it is collecting information about the blockchain itself, such as the number of processed transactions, elapsed time, time to create a block, etc. After the capture of all information, it is responsible for the processing of that information.

The logging module is used every time, and its responsibility is to log every event and save it to an external file. There is the capability of logging multiple events in a multi-threaded application. The order of those events is set by the time they were created.

The last module is the Key Management module, which is responsible for storing the keys that are used across the network. It is not part of anonymization nor the network layer because both of them are using it differently. The network layer is storing Diffie Hellman keys in it, and the Anonymization Layer is storing the public keys of other blockchain participants.

6.4 Class Hierarchy

This section provides information about class hierarchy in this project. The color of classes in figures does not mean anything. It is there only for better readability. The main class is the *Node* class which is composed of *Mempool*. The *mempool* is a class that encapsulates the collection of *transactions* and simplifies the work with this collection. The *node* consists of a dictionary that stores the mapping of processed blocks and their hashes. The *block* is composed of a *block header* and a collection of *transactions*. The *block header* is a data

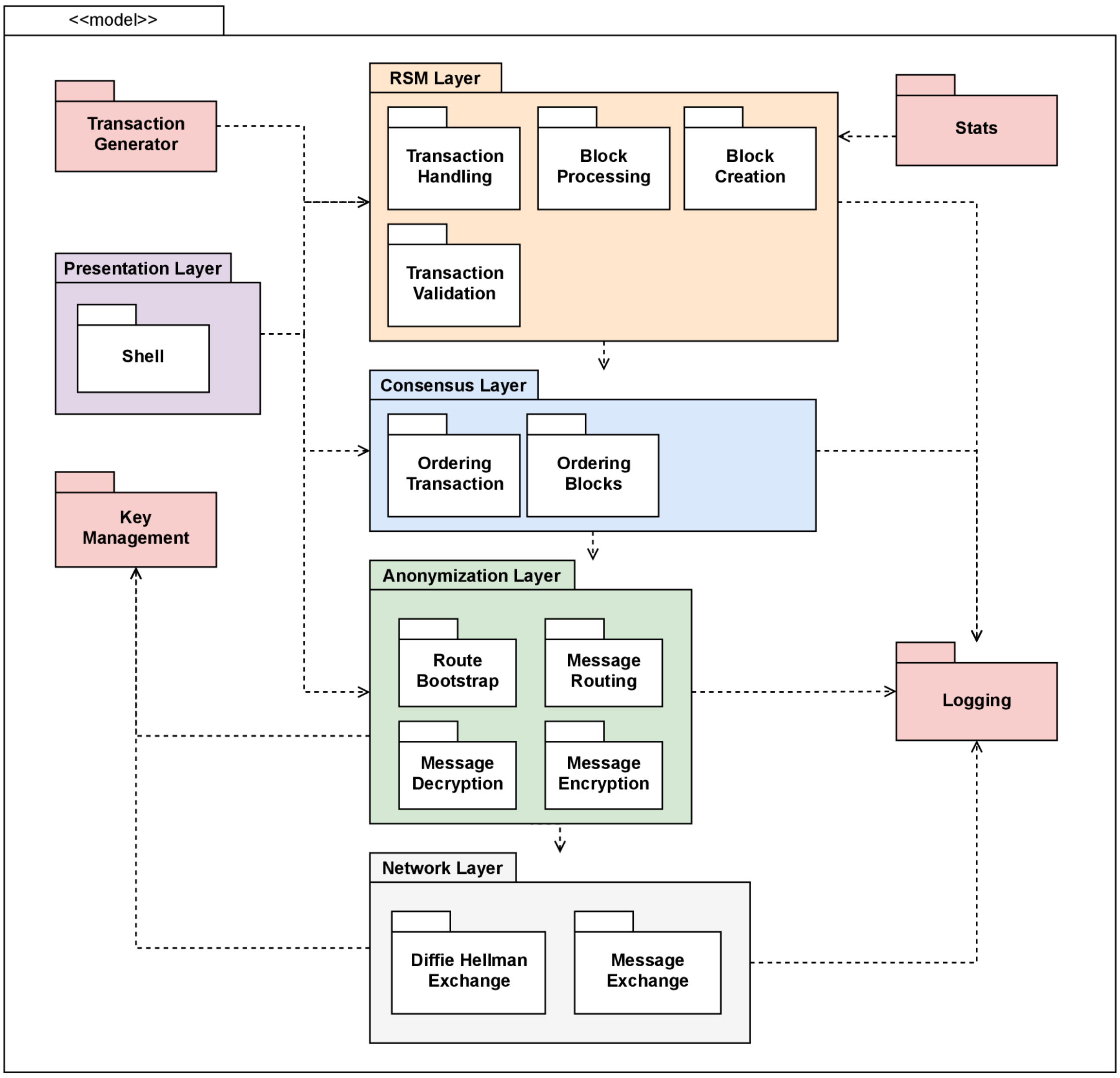


Figure 6.2: Diagram of modules and their dependencies inside the implementation of the proposed Proof-of-Stake protocol.

structure that holds data specified in Section 5.1. The *transaction* is a data structure that holds information about who is sending the crypto tokens to whom. The *transaction* also consists of its value, fee, nounce, and signature.

Another component of the *Node* class is *Stats*. This class stores and computes the statistical data that will be shown after the performance testing. The *node* has a collection that contains *NodeMeta* instances. The *NodeMeta* holds information about the node, e.g., public key, balance, stake, last used transaction nounce. The *TransportKey* class is a collection that stores Diffie Helman keys. The last part of the *Node* class is the *AnonymityLayer* class. That class is responsible for choosing a route among the nodes and for negotiating a Diffie Hellman key.

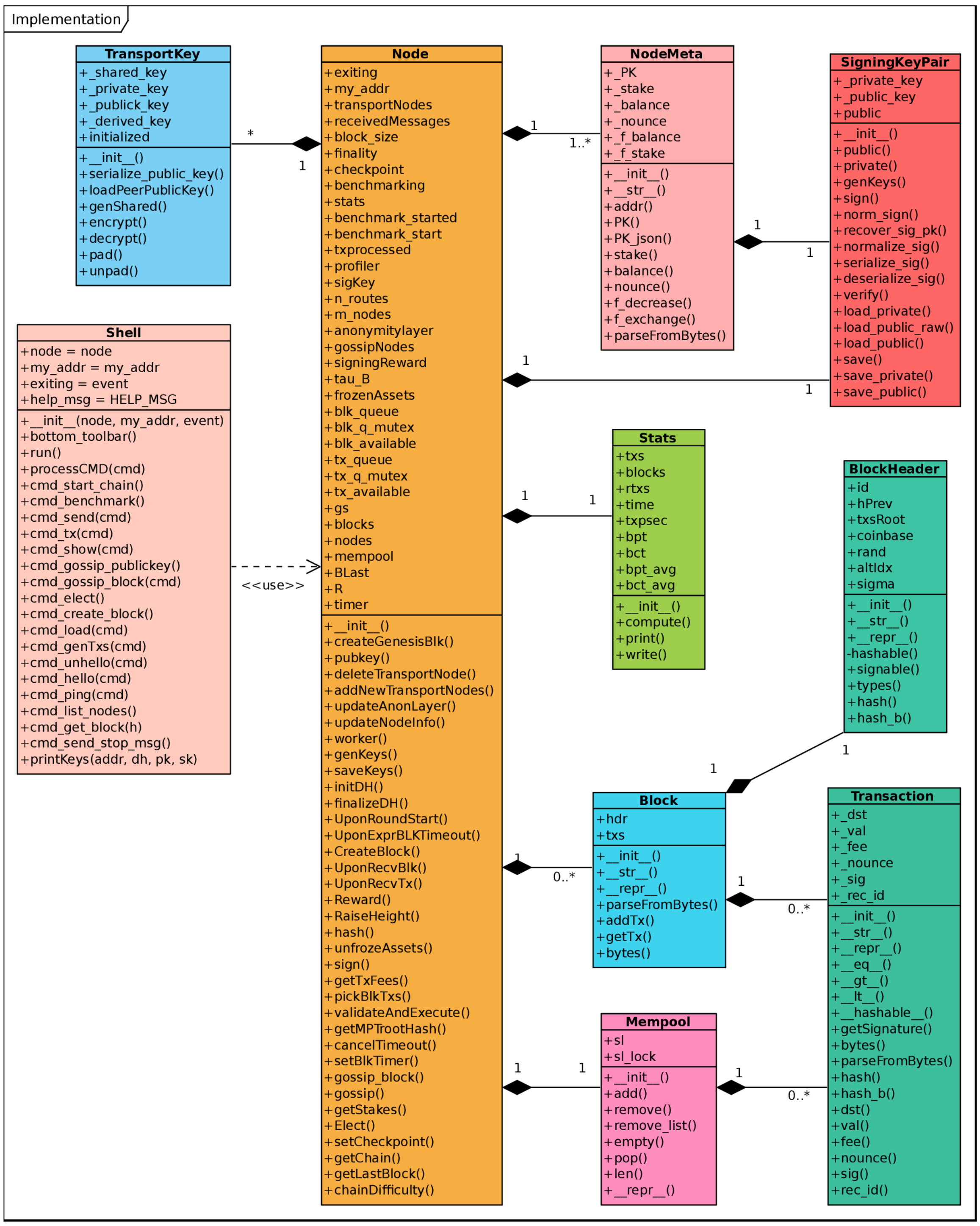


Figure 6.3: Fix colors, add MPT, shell. This figure shows the class hierarchy and their dependencies. The main class that holds whole functionality is the Node class. It consists of instances of other classes, e.g., NodeMeta, Block, . . . The NodeMeta holds information about a specific Node (public key, balance, . . .). The Block is a data structure that consists of Block Header and multiple Transactions. The node has one instance of Mempool in which are stored transactions that are not yet processed in the blockchain.

6.5 Toolset

To support the main application we designed two separate applications `settings_manager` and `tx_gen` and installation script. The first mentioned application will create the `config.py` file that is used as configuration file for the main application. The file consists of variables, e.g., block size, receive delay, finality, etc. This design was used due to easy-of-use manners when propagating values to multiple places in the application.

Settings Manager. The `settings_manager` can setup these values:

- **block size** - sets the number of transactions in the block,
- **finality** - sets the number of blocks that can be overturned in the current chain,
- **tau** - sets the round-reset timer,
- **receive delay** - sets the delay when receiving a block (used as network simulation parameter).
- **list of nodes** - sets the list of nodes addresses (it is created in sequence, e.g., ip:port, ip:(port + 1), ip:(port + 2), etc.)
- **generate transactions** - when set, the main application will generate a specified number of transactions before it starts.
- **a number of benchmark starting transactions** - threshold when the benchmark starts, i.e., the benchmark waits till the mempool size is at least *specified size*.
- **a number of benchmark stopping transactions** - threshold when the benchmark stops, i.e., the benchmark will stop the application and prints the statistics on the standard output.

```
> python ./settings_manager.py --help
usage: settings_manager.py [-h] [-starttt STARTTT] [-stoptt STOPTT] [-gmt
    GMT] [-bs BS] [-f FIN] [-t TAU] [-rd RD] [-nc IP_START PORT_START
    COUNT] [-o FILENAME]

optional arguments:
  -h, --help show this help message and exit
  -starttt STARTTT, --start-tx-treshold STARTTT
                        Set transaction treshold to start benchmark, default
                            set to 1000
  -stoptt STOPTT, --stop-tx-treshold STOPTT
                        Set transaction treshold to stop benchmark, default
                            set to 1000
  -gmt GMT, --generate-mempool-txs GMT
                        Generate Mempool Txs, default 0
  -bs BS, --block-size BS
                        Block size, default 100
  -f FIN, --finality FIN
                        Finality, how much block can be overturn, default 5
  -t TAU, --tau TAU Sleep time when the node is not the leader, default 60
  -rd RD, --recv-delay RD
                        Simulate network delay, default set to 0
  -nc IP_START PORT_START COUNT, --nodes-count IP_START PORT_START COUNT
                        Set IP, PORT and ID of nodes
  -o FILENAME, --output FILENAME
                        Output file
```

Figure 6.4: The run-time arguments of the settings manager.

Installation Script. The installation script is used to install all dependencies and setup the environment in which the main application can be executed. This script is used to set the testing environment in Metacentrum and for GitLab CI. The CI is set up to run unit tests and some of the basic use cases, e.g., create user key, create blockchain key, etc.

Transaction Generator. The second support application, `tx_gen.py`, is used to generate transactions and send them over the network to the node. This solution can be used to simulate peers in the network. The application can send each transaction individually or in bundles, e.g., 100 transactions in one message. The support application can be set to generate infinite or finite number of transactions.

6.6 Used Libraries

To support the implementation, we used 3rd party libraries that are mentioned bellow:

```
> python ./tx_gen.py --help
usage: tx_gen.py [-h] [--count COUNT] [--delta DELTA] [--keys KEYS]
    [--bundle BUNDLE]

optional arguments:
  -h, --help show this help message and exit
  --count COUNT Create COUNT bundles. Default set to 0
  --delta DELTA Delay between bundles. Default set to 0.0
  --keys KEYS Foleder, that holds the keys for signing the transactions.
  --bundle BUNDLE Size of a transaction bundle. Default set to 100
```

Figure 6.5: The run-time arguments of the transaction generator.

py-libsecp256k1. To use the signing algorithm ECDSA widely used in blockchains with parameters secp256k1, we used python library py-libsecp256k1[1]. It is a python interface for the libsecp256k1 c library.

sortedcontainers. The unprocessed transactions are stored in a mempool. The mempool is storing them sorted with transaction fee used as a sorting key. The motivation behind this is that the node wants to include the transactions with the biggest fees into the block, due the node will receive the fees as a reward. In order to have the transaction sorted all the time in the mempool, we used sortedcontainers[2].

eth-mpt. We used the eth-mpt[3] implementation of the Merkle Patricia Tree to store the global state of the blockchain, as proposed in the protocol.

[1]https://github.com/ludbb/secp256k1-py
[2]https://github.com/grantjenks/python-sortedcontainers
[3]https://github.com/popzxc/merkle-patricia-trie

Chapter 7

Analysis of proposed Proof-of-Stake Protocol

The analysis of the proposed protocol is described in this chapter. We tested the protocol in three scenarios, to prove that the solution has the desired properties.

7.1 Testing Plan

In order to test the implementation, we implemented a few features to the implementation itself. First is the `stats` module that stores information about how many transactions were received, processed, published. Another value that we store in the `stats` module is a block processing time and overall time from start to finish of the execution. We used those properties to compute overall stats for the evaluation of the protocol.

Another feature that is implemented is the transaction generator. This generator generates valid transactions with pseudo-random values. We used the `NumPy`[1] library for the pseudo-random implementation of the number generator. The generator uses a normal distribution to generate numbers, with the mean value set to 0.2, and standard deviation to 0.2 as well. The generator supports a feature that can create the transaction in batches or one by one. A delay can be inserted into sending badges to the node or into the network processing in the Node implementation. With those delays, we can simulate real-world network latencies that are not present in the lab itself.

The steps used in each test:

1. Create a virtual machine in OpenStack instance with at least 2GB RAM with operating system Ubuntu 20.04.
2. Initialize the virtual machine, install all dependencies and libraries.
3. Clone application inside the machine.
4. Generate settings for node using a dedicated tool (mempool threshold for node start, processed transaction threshold for node stop, number of transactions, that will be generated into the mempool, network delay, block size, reset timer τ, node list with addresses).

[1] `https://numpy.org/`

5. Generate a public and private key for each node that will be participating in the test case.
6. One randomly chosen node will generate blockchain private and public keys.
7. The node which generated the blockchain key will share the key with other participating nodes.
8. One node is in advance marked as stat counter.
9. We start the nodes in headless mode.
10. Depending on the genesis block, each node checks if it is the leader and if so, she will generate a block. The node automatically gossips the newly created block into the network. This broadcast will jump-start the chain reaction of the whole chain.
11. When the node achieves the threshold of processed transactions, it will send a stop message to every node.
12. The one node that has the statistics will print it into a specified file.

7.2 Test Cases

Multiple test cases were created to provide data that will be compared with algorithms that were described in Chapter 2. The scenarios are:

- Run all nodes on one machine using localhost to share messages, without using anonymization layer.
- Run all nodes on one machine using localhost to share messages, with anonymization layer turned on.
- Run all nodes on separate machines using real network to share messages, with anonymization layer turned on.

The first test case is used to set a baseline for comparison. The second test case is used to discover the overhead of the anonymization layer and its impact on the node's performance. The third test case is to show how the real network affects the protocol itself.

These three test cases are designed to run with different properties to find out if the protocol exceeds the already implemented state-of-the-art proof-of-stake protocols. The variable properties in the test cases are *block size*, *reset timer* τ. Thus we picked values shown in Table 7.1 that will be tested.

7.3 Results

We concluded the experiments with the attributes specified in Table 7.1, and the results are shown in Figure 7.1 and Figure 7.2. The first run of experiments was run on localhost without the anonymization layer turned on. The results are drawn with a blue line. The second run of experiments was the same as the first one, but the anonymization layer was turned on. The results are shown with the red line in the figures. The last set of experiments was run on different machines connected with regular network settings. As you can see in the statistics, the throughput was not affected by the anonymization layer or the network itself. We measured the block processing time, and it was not affected by these environmental settings. As we can see in Figure 7.2, the throughput peaked around the value 770 tx/s with block size 2000 transactions. The block processing time is increasing exponentially, and it is not feasible to use a block size bigger than 5000. When the block size is bigger than this threshold, the network latency increases exponentially as well.

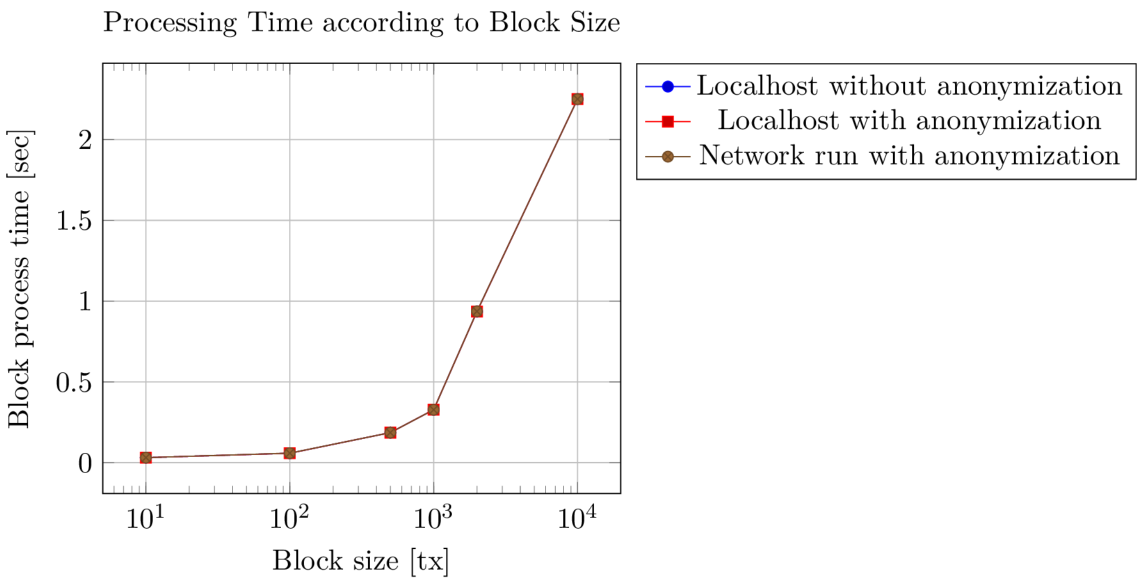


Figure 7.1: This figure shows the connection between block size and the processing time of the block. As you can see the block processing time is increasing linearly with the block size.

The Figure 7.3 show us the performance difference between the situation where is the verifying of the transaction signatures enabled and disabled. We stuck with the verification turned off because usually, the verification is made off-chain by the validating nodes.

Another experiment was to measure the throughput affected by the size of offline nodes in the network. We concluded the run with multiple sizes of offline group in the network (10, 20, 30, 40, 50%). The results can be seen in Figure 7.4.

Block size $[tx]$	Reset timer τ $[s]$
10	2
100	20
500	20
1000	200
2000	200
10000	200

Table 7.1: This table shows variables that are used for every test case. When we extend the block size we must extend the reset timer as well, due to extended block processing time. Before the experiment starts, we filled the mempool of each node to have sufficient amount of transactions for processing.

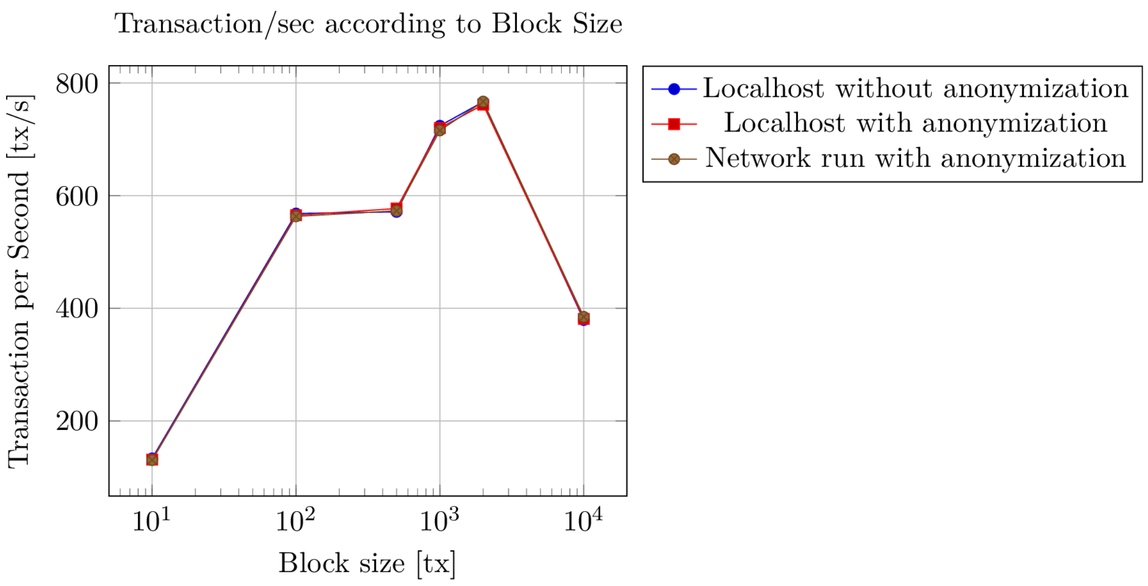


Figure 7.2: This figure shows the throughput of the protocol based on the block size. The peak is around the block size with 2000 transactions. The peak value is 766 tx/sec.

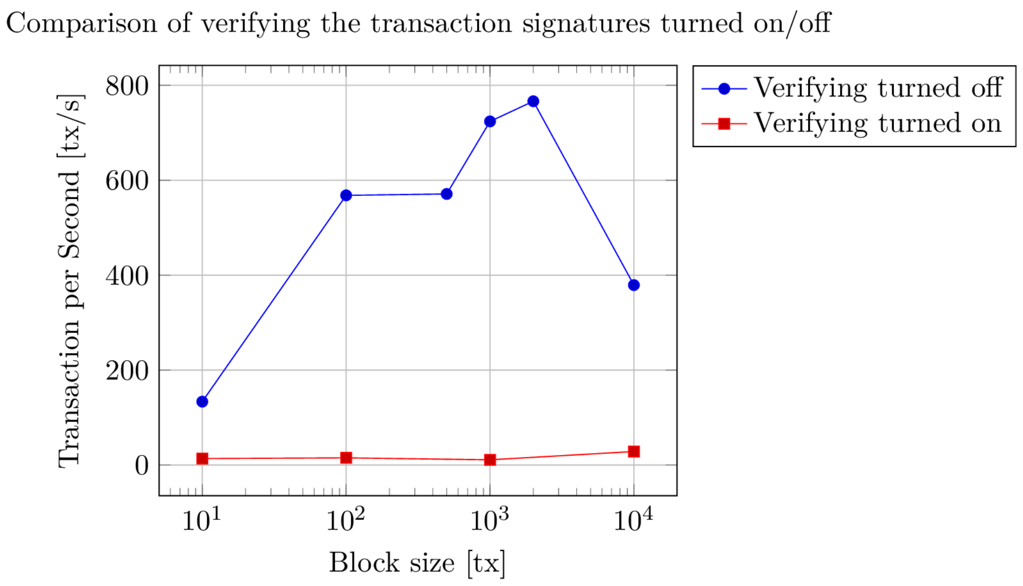


Figure 7.3: This figure shows the difference in throughput when the verifying of the transaction's signature is turned on and off.

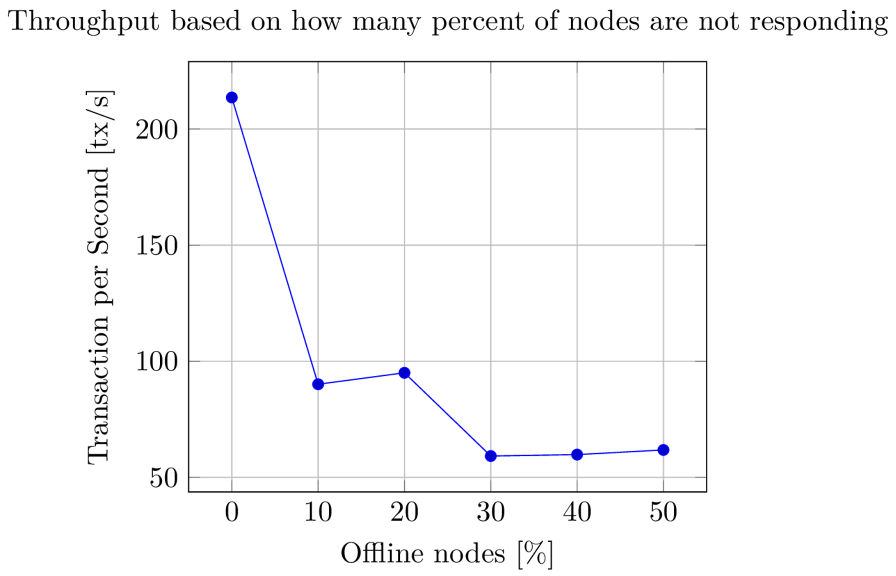


Figure 7.4: This figure shows how an offline group can affect the throughput. As you can see, the throughput decreases when the number of offline nodes increases in the network.

Chapter 8

Conclusion

The goal of this thesis was to implement a blockchain consensus protocol that is secure, DoS resistant, easily scalable, and with a native implemented anonymization layer.

We learned how blockchain networks work and the difference among the various consensus protocols. We studied and analyzed individual proof-of-stake protocols with their properties, i.e., throughput, scalability, and security. The main proof-of-stake protocols that we analyzed are Tendermint, Algorand, and LaKSA. We studied how the anonymization networks, name Tor and I2P, work. Next, we figured out how to implement those ideas into the proposed protocol.

The main part of the thesis was to implement a proof-of-concept of the proposed protocol by the security@FIT group and conclude experiments upon the implementation. The experiment part was used to support the idea, but the real-world performance would need an overhaul. The critical factor is the relatively low throughput in current implementation due to chosen language and support libraries. We encourage to use a compiled language instead of interpreted one, e.g., Go.

This thesis can be extended with the implementation of smart contracts, which could significantly improve the usability of the final protocol. The reason is that the leading players in the industry do not use proof-of-stake consensus protocol yet, or the smart-contract language is not usable.

Bibliography

[1] Alangot, B., Reijsbergen, D., Venugopalan, S. and Szalachowski, P. Decentralized lightweight detection of eclipse attacks on bitcoin clients. In: IEEE. *2020 IEEE International Conference on Blockchain (Blockchain)*. 2020, p. 337–342.

[2] Antonopoulos, A. M. and Wood, G. *Mastering ethereum: building smart contracts and dapps*. O'reilly Media, 2018.

[3] Apostolaki, M., Marti, G., Müller, J. and Vanbever, L. SABRE: Protecting bitcoin against routing attacks. *ArXiv preprint arXiv:1808.06254*. 2018.

[4] Apostolaki, M., Zohar, A. and Vanbever, L. Hijacking bitcoin: Routing attacks on cryptocurrencies. In: IEEE. *2017 IEEE Symposium on Security and Privacy (SP)*. 2017, p. 375–392.

[5] Ariyapperuma, S. and Mitchell, C. J. Security vulnerabilities in DNS and DNSSEC. In: IEEE. *The Second International Conference on Availability, Reliability and Security (ARES'07)*. 2007, p. 335–342.

[6] Bano, S., Sonnino, A., Al Bassam, M., Azouvi, S., McCorry, P. et al. SoK: Consensus in the age of blockchains. In: *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*. 2019, p. 183–198.

[7] Berger, C. and Reiser, H. P. Scaling Byzantine Consensus: A Broad Analysis. In: *Proceedings of the 2nd Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers*. New York, NY, USA: Association for Computing Machinery, 2018, p. 13–18. SERIAL'18. DOI: 10.1145/3284764.3284767. ISBN 9781450361101. Available at: https://doi.org/10.1145/3284764.3284767.

[8] Blockchainwiki. *Smart contract*. 2017-10-6. Available at: https://en.bitcoinwiki.org/wiki/Smart_contract.

[9] Bojja Venkatakrishnan, S., Fanti, G. and Viswanath, P. Dandelion: Redesigning the bitcoin network for anonymity. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*. ACM New York, NY, USA. 2017, vol. 1, no. 1, p. 1–34.

[10] Bonneau, J., Miller, A., Clark, J., Narayanan, A., Kroll, J. A. et al. Sok: Research perspectives and challenges for bitcoin and cryptocurrencies. In: IEEE. *2015 IEEE symposium on security and privacy*. 2015, p. 104–121.

[11] Boverman, A. *Timejacking & Bitcoin*. 2011-05-21. Available at: http://culubas.blogspot.com/2011/05/timejacking-bitcoin_802.html.

[12] Bradner, S. *Key words for use in RFCs to Indicate Requirement Levels* [Internet Requests for Comments]. RFC 2119. RFC Editor, march 1997. Available at: https://www.rfc-editor.org/rfc/rfc2119.txt.

[13] Buchman, E. *Tendermint: Byzantine fault tolerance in the age of blockchains*. 2016. Dissertation. University of Guelph, School of Engineering.

[14] Buterin, V. Long-range attacks: The serious problem with adaptive proof of work. *Ethereum Blog, May*. 2014.

[15] Buterin, V. Slasher: A punitive proof-of-stake algorithm. *Ethereum Blog URL: https://blog. ethereum. org/2014/01/15/slasher-a-punitive-proof-of-stake-algorithm*. 2014.

[16] Buterin, V. and Griffith, V. Casper the friendly finality gadget. *ArXiv preprint arXiv:1710.09437*. 2017.

[17] Cachin, C. and Poritz, J. A. Secure intrusion-tolerant replication on the Internet. In: IEEE. *Proceedings International Conference on Dependable Systems and Networks*. 2002, p. 167–176.

[18] Castro, M., Druschel, P., Ganesh, A., Rowstron, A. and Wallach, D. S. Secure routing for structured peer-to-peer overlay networks. *ACM SIGOPS Operating Systems Review*. ACM New York, NY, USA. 2002, vol. 36, SI, p. 299–314.

[19] Chaum, D. L. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*. ACM New York, NY, USA. 1981, vol. 24, no. 2, p. 84–90.

[20] Chohan, U. W. The double spending problem and cryptocurrencies. *Available at SSRN 3090174*. 2017.

[21] Cohen, J. D. and Fischer, M. J. *A robust and verifiable cryptographically secure election scheme*. Yale University. Department of Computer Science, 1985.

[22] Daian, P., Pass, R. and Shi, E. Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake. In: Springer. *International Conference on Financial Cryptography and Data Security*. 2019, p. 23–41.

[23] David, B., Gaži, P., Kiayias, A. and Russell, A. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In: Springer. *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. 2018, p. 66–98.

[24] Dexter, S. *1% Shard Attack Explained – Ethereum Sharding (Contd..)*. 2018-03-11. Available at: https://www.mangoresearch.co/1-shard-attack-explained-ethereum-sharding-contd/.

[25] Dingledine, R., Freedman, M. J. and Molnar, D. The free haven project: Distributed anonymous storage service. In: Springer. *Designing Privacy Enhancing Technologies*. 2001, p. 67–95.

[26] Dziembowski, S., Faust, S., Kolmogorov, V. and Pietrzak, K. Proofs of space. In:. Springer Verlag, 2015, vol. 9216, p. 585–605. ISBN 9783662479995.

[27] Eskandari, S., Clark, J., Barrera, D. and Stobert, E. A first look at the usability of bitcoin key management. *ArXiv preprint arXiv:1802.04351*. 2018.

[28] Franklin, M. A survey of key evolving cryptosystems. *International Journal of Security and Networks*. Inderscience Publishers. 2006, vol. 1, 1-2, p. 46–53.

[29] Fromknecht, C., Velicanu, D. and Yakoubov, S. A Decentralized Public Key Infrastructure with Identity Retention. *IACR Cryptol. ePrint Arch.* 2014, vol. 2014, p. 803.

[30] Gaži, P., Kiayias, A. and Russell, A. Stake-bleeding attacks on proof-of-stake blockchains. In: IEEE. *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*. 2018, p. 85–92.

[31] Gilad, Y., Hemo, R., Micali, S., Vlachos, G. and Zeldovich, N. Algorand: Scaling byzantine agreements for cryptocurrencies. In: *Proceedings of the 26th Symposium on Operating Systems Principles*. 2017, p. 51–68.

[32] Gray, J. et al. The transaction concept: Virtues and limitations. In: *VLDB*. 1981, vol. 81, p. 144–154.

[33] Hanke, T., Movahedi, M. and Williams, D. Dfinity technology overview series, consensus system. *ArXiv preprint arXiv:1805.04548*. 2018.

[34] Heilman, E., Kendler, A., Zohar, A. and Goldberg, S. Eclipse attacks on bitcoin's peer-to-peer network. In: *24th {USENIX} Security Symposium ({USENIX} Security 15)*. 2015, p. 129–144.

[35] Heleness. *Blockchain-based notarization platform on Ethereum*. 2018. Available at: https://ethresear.ch/t/blockchain-based-notarization-platform-on-ethereum/5326.

[36] Hellebrandt, L., Homoliak, I., Malinka, K. and Hanacek, P. Increasing Trust in Tor Node List Using Blockchain. In: *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, 2019, p. 29–32. ISBN 9781728113289.

[37] Higgins, S. *Bitcoin mining pools targeted in wave of ddos attacks*. 2015.

[38] Homoliak, I., Venugopalan, S., Reijsbergen, D., Hum, Q., Schumi, R. et al. The Security Reference Architecture for Blockchains: Towards a Standardized Model for Studying Vulnerabilities, Threats, and Defenses. *IEEE Communications surveys and tutorials*. IEEE. 2020, p. 1–1. ISSN 1553-877X.

[39] Huston, G. and Bush, R. Securing bgp with bgpsec. In: Citeseer. *The Internet Protocol Forum*. 2011, vol. 14, no. 2.

[40] Hyperledger. *Hyperledger Architecture, Volume 1: Introduction to Hyperledger Business Blockchain Design Philosophy and Consensus*. 2017. Available at: https://www.hyperledger.org/wp-content/uploads/2017/08/Hyperledger_Arch_WG_Paper_1_Consensus.pdf.

[41] Karakostas, D. and Kiayias, A. Securing Proof-of-Work Ledgers via Checkpointing. *IACR Cryptol. ePrint Arch.* 2020, vol. 2020, p. 173.

[42] Karantias, K., Kiayias, A. and Zindros, D. Proof-of-burn. In: Springer. *International Conference on Financial Cryptography and Data Security.* 2020, p. 523–540.

[43] Kiayias, A., Russell, A., David, B. and Oliynykov, R. Ouroboros: A provably secure proof-of-stake blockchain protocol. In: Springer. *Annual International Cryptology Conference.* 2017, p. 357–388.

[44] King, S. and Nadal, S. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. *Self-published paper, August.* 2012, vol. 19, p. 1.

[45] McCorry, P., Shahandashti, S. F. and Hao, F. A smart contract for boardroom voting with maximum voter privacy. In: Springer. *International Conference on Financial Cryptography and Data Security.* 2017, p. 357–375.

[46] Micali, S., Rabin, M. and Vadhan, S. Verifiable random functions. In: IEEE. *40th annual symposium on foundations of computer science (cat. No. 99CB37039).* 1999, p. 120–130.

[47] MihailoBjelic. *On the probability of 1% attack.* 2018-11-01. Available at: https://ethresear.ch/t/on-the-probability-of-1-attack/4009/6.

[48] Miller, A., Juels, A., Shi, E., Parno, B. and Katz, J. Permacoin: Repurposing bitcoin work for data preservation. In: IEEE. *2014 IEEE Symposium on Security and Privacy.* 2014, p. 475–490.

[49] Miller, A., Xia, Y., Croman, K., Shi, E. and Song, D. The honey badger of BFT protocols. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security.* 2016, p. 31–42.

[50] MME. *Conceptual framework for legal and risk assessment of crypto tokens.* 2018. Available at: https://www.mme.ch/fileadmin/files/documents/180501_BCP_Framework_for_Assessment_of_Crypto_Tokens_-_Block_2.pdf.

[51] Nakamoto, S. *Bitcoin: A peer-to-peer electronic cash system.* Manubot, 2019.

[52] National Notary Association, C. *What is notarization?* 2009. Available at: https://www.nationalnotary.org/knowledge-center/about-notaries/what-is-notarization.

[53] Park, S., Kwon, A., Fuchsbauer, G., Gaži, P., Alwen, J. et al. Spacemint: A cryptocurrency based on proofs of space. In: Springer. *International Conference on Financial Cryptography and Data Security.* 2018, p. 480–499.

[54] Park, S., Pietrzak, K., Alwen, J., Fuchsbauer, G. and Gazi, P. Spacecoin: A cryptocurrency based on proofs of space. *IACR Cryptology ePrint Archive.* 2015, vol. 2015, p. 528.

[55] QuantumMechanic. *Proof of stake instead of proof of work.* 2011-06-11. Available at: https://bitcointalk.org/index.php?topic=27787.0.

[56] Reijsbergen, D., Szalachowski, P., Ke, J., Li, Z. and Zhou, J. ProPoS: A Probabilistic Proof-of-Stake Protocol. *ArXiv preprint arXiv:2006.01427*. 2020.

[57] Schneider, F. B. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys (CSUR)*. ACM New York, NY, USA. 1990, vol. 22, no. 4, p. 299–319.

[58] Sgnornic, Lerner, Atheros, Guaka, Aceat64 et al. *Weaknesses*. 2020-06-27. Available at: https://en.bitcoin.it/wiki/Weaknesses#DenialofService.28DoS.29attacks.

[59] Shares, D. *Major DDoS Attacks Hit Bitcoin.com*. 2017-03-18. Available at: https://news.bitcoin.com/ddos-attacks-bitcoin-com-uncensored-information/.

[60] Siim, J. Proof-of-stake. In: *Research Seminar in Cryptography*. 2017.

[61] Sonnino, A., Bano, S., Al Bassam, M. and Danezis, G. Replay attacks and defenses against cross-shard consensus in sharded distributed ledgers. In: IEEE. *2020 IEEE European Symposium on Security and Privacy (EuroS&P)*. 2020, p. 294–308.

[62] Syta, E., Jovanovic, P., Kogias, E. K., Gailly, N., Gasser, L. et al. Scalable bias-resistant distributed randomness. In: Ieee. *2017 IEEE Symposium on Security and Privacy (SP)*. 2017, p. 444–460.

[63] Syverson, P., Dingledine, R. and Mathewson, N. Tor: The secondgeneration onion router. In: *Usenix Security*. 2004, p. 303–320.

[64] Szalachowski, P. (Short Paper) Towards More Reliable Bitcoin Timestamps. In: IEEE. *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*. 2018, p. 101–104.

[65] Tran, M., Choi, I., Moon, G. J., Vu, A. V. and Kang, M. S. A stealthier partitioning attack against bitcoin peer-to-peer network. In: *IEEE Symposium on Security and Privacy (S&P)*. 2020.

[66] Wang, W., Hoang, D. T., Hu, P., Xiong, Z., Niyato, D. et al. A Survey on Consensus Mechanisms and Mining Strategy Management in Blockchain Networks. *IEEE Access*. IEEE. 2019, vol. 7, no. 99, p. 22328–22370. ISSN 2169-3536.

[67] Wilcox O'Hearn, Z. Names: Decentralized, secure, human-meaningful: Choose two. *Online] https://web. archive. org/web/20011020191610/http://zooko. com/distnames. html[retrieved 2018-04-21]*. 2003.

[68] Wüst, K. and Gervais, A. *Ethereum eclipse attacks*. ETH Zurich, 2016.

[69] Øverlier, L. *Anonymity, Privacy and Hidden Services: Improving censorship-resistant publishing*. 2007. Available at: http://hdl.handle.net/10852/9783.

Appendix A

Contents of the included storage media

On attached compact disk are located source codes of the proof-of-concept protocol (in directory `src`), and thesis including LaTeXsource code (in directory `thesis`). The materials that were used at Excel@FIT are located in directory `excel`.

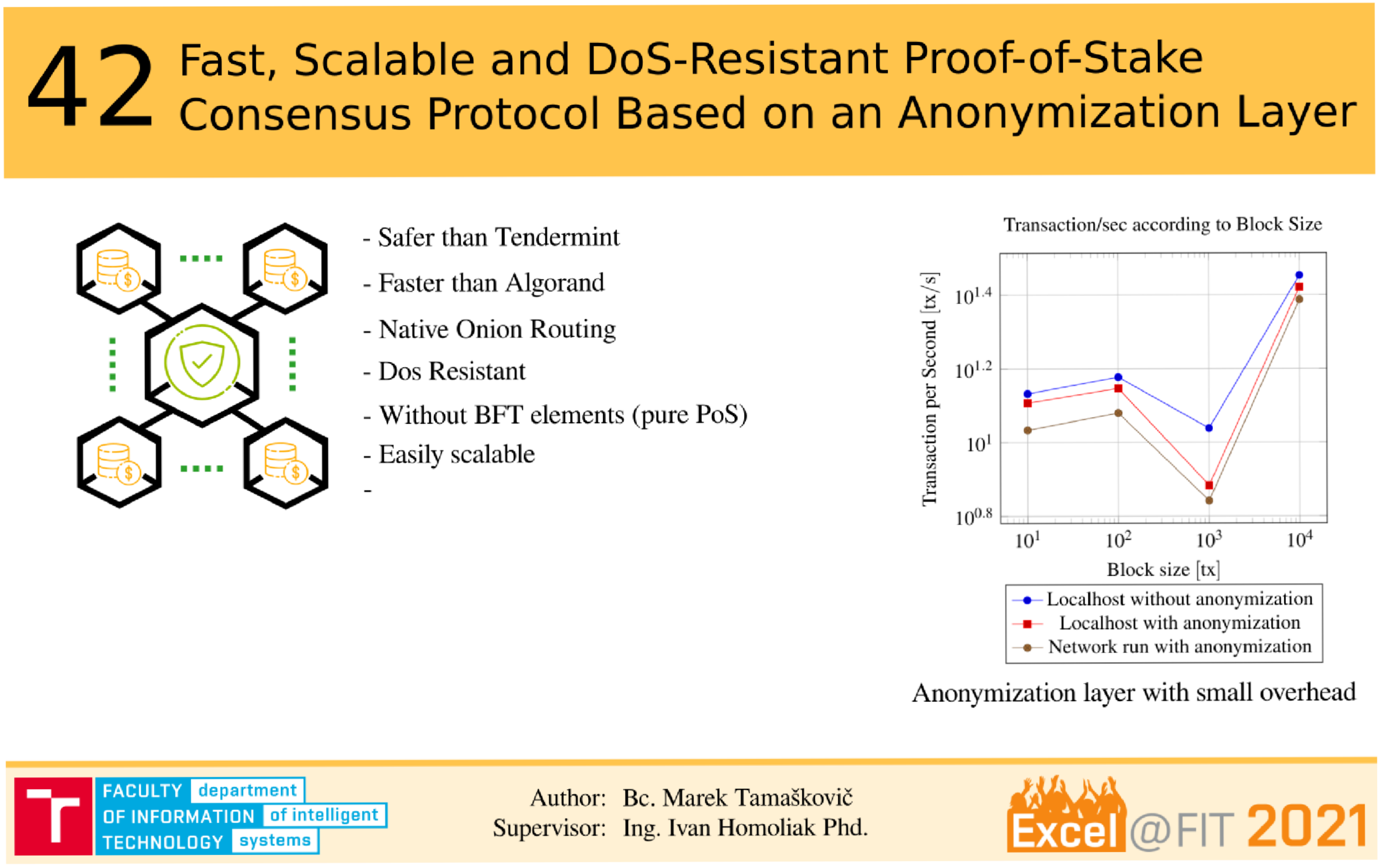
- `sources` – Here you can find zip file with all sources:
  - `sources_master.zip`-- zip file with all sources:
    - `Results/` – contains raw results measured by the protocol itself,
    - `chain/` – main implementation of the consensus protocol is located here,
    - `excel/` – materials that were used at Excel@FIT,
    - `img/` – images that are used in `README.md`,
    - `logs/` – helping directory for storing log files,
    - `metacentrum/` – configuration scripts that are used to install dependecies on virtual machine and to setup Gitlab CI,
    - `tests/` – contains unit tests,
    - `License` – MIT License,
    - `README.md` – notes to the project,
    - `config.py` – configuration file of the consensus protocol,
    - `main.py` – main script,
    - `requirements.txt` – python module requirements,
    - `settings_manager.py` – helper script to setup `config.py`,
- `thesis` – contains pdf and sources of this thesis,
  - `xtamas01_sources.zip` – LaTeX sources to compile this thesis,
  - `xtamas01.pdf` – this thesis,
- `excel` – contains materials used at Excel@FIT:
  - `slide.svg` – Pitch slide used at Excel@FIT,
  - `logo.svg` – Logo slide used at Excel@FIT,

Appendix B

Excel@FIT Material

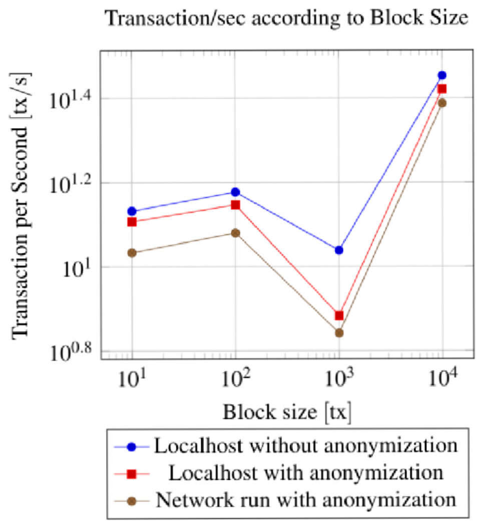
Here are located materials that were used at Excel@FIT. Materials are shown in Figure B.1 and Figure B.2.

Figure B.1: Logo used at Excel@FIT



42 Fast, Scalable and DoS-Resistant Proof-of-Stake Consensus Protocol Based on an Anonymization Layer

- Safer than Tendermint
- Faster than Algorand
- Native Onion Routing
- Dos Resistant
- Without BFT elements (pure PoS)
- Easily scalable
-



Anonymization layer with small overhead

FACULTY OF INFORMATION TECHNOLOGY department of intelligent systems

Author: Bc. Marek Tamaškovič
Supervisor: Ing. Ivan Homoliak Phd.

Excel@FIT 2021

Figure B.2: Pitch slide used at Excel@FIT