# Using Blockchain Techniques to Create an Opinion-Based Whitelisting Procedure

DAVID JOHANSSON
SIMON ALM NILSSON
MASTER´S THESIS
DEPARTMENT OF ELECTRICAL AND INFORMATION TECHNOLOGY
FACULTY OF ENGINEERING | LTH | LUND UNIVERSITY

# Using Blockchain Techniques to Create an Opinion-Based Whitelisting Procedure

David Johansson, Simon Alm Nilsson
dat13djo@student.lu.se, dat13sni@student.lu.se

Department of Electrical and Information Technology
Lund University

Supervisor: Paul Stankovski
paul.stankovski@eit.lth.se

Examiner: Thomas Johansson
thomas.johansson@eit.lth.se

June 14, 2018

Printed in Sweden
Tryckeriet i E-huset, Lund

# Abstract

Malware has proven to be a persistent problem with an increasing amount of variations, and new attack vectors are constantly being taken advantage of. Security specialists are always on the hunt for new technologies useful in the fight against malware. Blockchain technologies bring promises of high integrity, decentralization, and transparency. The technology is very much in its infancy, but previous research has identified scalability as a weakness.

In this thesis, a prototype to be used by a group of users with limited trust for each other was designed and developed. The prototype is able to gather information from software and use it to allow the group to create a uniform opinion for whitelisting software. Different approaches for how each user can generate a vote with minimal user intrusion was discussed. To be able to assess the design's scalability and limitations, a thorough review of current research was performed. The goal of the review was to determine differences between blockchains and traditional databases with focus on aspects such as properties, performance, cost, and security.

A working proof of concept was developed, and its potential scalability was discussed. It was shown to scale similarly to Byzantine fault tolerant consensus algorithms often used in permissioned blockchains. An estimate of at most 100 to 1,000 users was motivated, and collected research indicate a throughput of single digits per second, with potentially 20 minutes of delay at 1,000 users. The usage of smart contracts had benefits of more transparency, higher integrity and decentralized verification of the result. Tests showed the performance of the smart contract used in the prototype scaled well with thousands of versions of programs and would not be a bottleneck. The analysis of current research papers was used to create a summarizing table and a decision tree that should be helpful for developers when deciding to use a blockchain or a traditional database in their systems.

# Acknowledgments

First and foremost we would like to thank our supervisor at Lund's Faculty of Engineering; Paul Stankovski, and our two supervisors at SecureLink; Christoffer Toft and Diana Selck. We would also like to thank all the people we came in contact with at SecureLink who contributed with ideas and a pleasant work environment.

Lastly, a big thank you to our friends and families for all the support you have given us throughout life.

# Table of Contents

# List of Figures

# List of Tables

# Abbreviations

**API** ..... Application Programming Interface
**BFT** ..... Byzantine Fault Tolerance
**BFT-SMaRt** .... Byzantine Fault-Tolerant State Machine Replication
**CA** ..... Certificate Authority
**CPU** ..... Central Processing Unit
**DAO** ..... Decentralized Autonomous Organization
**DLT** ..... Distributed Ledger Technology
**ECDSA** ..... Elliptic Curve Digital Signature Algorithm
**GDPR** ..... General Data Protection Regulation
**HDD** ..... Hard Disk Drive
**IoT** ..... Internet of Things
**JSON** ..... JavaScript Object Notation
**MSP** ..... Membership Service Provider
**PBFT** ..... Practical Byzantine Fault Tolerance
**PE** ..... Portable Executable
**PGP** ..... Pretty Good Privacy
**PKI** ..... Public Key Infrastructure
**PoC** ..... Proof of Concept
**SDK** ..... Software Development Kit
**SGX** ..... Software Guard Extensions
**SHA** ..... Secure Hashing Algorithm
**SSD** ..... Solid State Drive
**TLS** ..... Transport Layer Security
**YCSB** ..... Yahoo! Cloud System Benchmark

# Popular Science Summary

**Ever downloaded a program and wondered if it was safe to run? Can blockchains, the technology behind Bitcoin, be useful in an application for collectively voting on and whitelisting software? This thesis builds one such application and evaluates the limitations and scalability of the blockchain technology.**

How do you know if a program you just downloaded is safe to run, or if it is a virus? Perhaps you use your gut feeling, an anti-virus software, or a tech-savvy relative? In all of those cases you blindly trust one single party. If the judgment of said party is off, you might end up running a virus on your computer. This Master's Thesis investigates how *blockchains* can be used to allow multiple people to electronically vote on whether a certain program is good or bad. Furthermore, the thesis assumes there are ill-advised and villainous users swaying the opinion on software in the wrong direction — both upvoting viruses and downvoting harmless programs.

Think of a blockchain as a sturdy chain with information. It is impossible to remove a link once it has been attached to the chain. You can easily point to a link and everyone else is able to verify that it is indeed in the chain. By utilizing this property and storing votes on the chain, a blockchain with verifiable votes was created.

Blockchain is one of many technologies with the purpose of storing data, so-called *databases*. Blockchain is a new technology with many interesting properties. It is still not clear where blockchains should be used, and the judgement of many is clouded by the recentness of the technology. A comparison to databases was made with regards to properties such as; performance, security, and cost. Blockchains have interesting properties, but often scale negatively with the amount of users or have substantial delay. The thesis includes a table which shows how blockchains and databases scale, as well as a decision tree. This can be used to help developers decide on whether blockchain is a good fit for their application or not.

The proof of concept would be able to handle up to a third of the users acting in a malicious way before they would be able to sway the opinion in the wrong direction. However, it would not support more than a hundred users. If users would be willing to fully trust a single individual or organization, a program using a more traditional database would be able to have millions of users instead.

# Chapter 1
# Introduction

## 1.1 Background

Every time a user downloads software the user is at risk of infecting both the computer and the network the user is connected to. According to Symantec, one in every 131 emails contains malware [2]. A common threat is software that has been tampered with, injected with malicious behavior. Symantec reports over 357 million new malware variants for the year 2016 [2], continuing the trend of an increasing malware problem. The fight against malware is a difficult one and can be viewed as an arms race between malware developers and security experts. New techniques and methods are explored and used by both sides to gain advantage over the other.

The rise of cryptocurrencies has resulted in new techniques for verifiable transactions and records, most famously blockchain from Bitcoin. Blockchain techniques are secure by design [3] and offer a way for a peer-to-peer network of entities to together agree on a *truth*, offer immutability, and decentralization. The technique offers high integrity and distributed consensus. The addition of smart contracts, code that is executed and has its result verified by all peers in the network, also opens new options for decentralized solutions.

Blockchains are cited to have many advantages over traditional alternatives [4, 5] and the technology has gathered a significant amount of hype [6]. A wide array of industries and sectors experiment with the technology in hopes of becoming early adapters of its potential and the IT security industry is no exception. The blockchain technology however, is not always well understood, and comes with its own set of limitations. Scalability has been identified as an area of interest for future research [4], and even though new research in the area has been performed [1, 7, 8, 9, 10], it can be hard to get a grasp of the current situation with blockchains compared to central alternatives. It is of interest to explore important considerations of the technology and clearly compare them with other alternatives.

## 1.2 Scope

The objective of this thesis was to investigate if it would be beneficial to use blockchain techniques in a system designed to detect malware or otherwise help

increase end user security. Focus was placed on what considerations and limitations will come out of using blockchains instead of more traditional, centralized, alternatives. A proof of concept (PoC) capable of utilizing the strengths of blockchain techniques was designed and developed, and its scalability was discussed. It was also shown that the PoC scales similarly to permissioned blockchains in general, and scalability of blockchains was compared to centralized and distributed databases. The thesis aims to help developers of tomorrow's systems by providing a list of considerations to make before choosing permissioned blockchains for their applications, as well as identifying the advantages of implementing the program logic in a smart contract.

The scope of the thesis can be summarized by the following three questions:

1. What can be said about scalability of the design, both in terms of users and software? Is it worth it compared to a central alternative?

2. What considerations and limitations are there for applying the design in a permissioned blockchain?

3. What are the advantages when implementing the program logic in a smart contract compared to outside of the blockchain?

Answers to these questions will be provided throughout the report.

## 1.3 Method

The first month was spent on a comprehensive literature study on blockchain technologies, and finding a suitable blockchain project for the PoC. Information was primarily gathered from Lund University Libraries and Lund University Publications Student Papers. Finding information was not a problem because of the popularity blockchains have had for the past decade. Other areas that were also researched, though not as comprehensibly, include Portable Executable (PE) file format, malware detection, and software fingerprinting.

The second month consisted of developing the PoC that was later to be used in testing and evaluation. The development was performed in parallel, with one author developing for example the chaincode, while the other worked on getting the Java SDK client to work. An in-depth research phase on scalability of consensus algorithms and blockchains was conducted to help answer Question 1.

During the third month, tests were designed, implemented, and executed. The authors discovered that more time than necessary was designated for the testing phase. It was cut short, and time was instead spent on improving the chapter comparing blockchains to databases, Chapter 7. The tests were used to tie together the developed PoC with the section comparing blockchains to centralized databases, and finally answer the questions from Section 1.2.

The report was worked on for the entirety of the thesis process. The authors did not want to hastily write together a report, nor forget any details from previously finished work.

## 1.4 Related work

It is impossible to exclude Nakamoto's paper on Bitcoin [11] when talking about blockchains. This paper started the modern day interest in cryptocurrencies, and introduced blockchains. The application developed for this thesis does not use Bitcoin, but it is included in discussions as it is the largest blockchain project in terms of users.

Hyperledger Fabric is the primary blockchain project of interest in this thesis. The white paper specifying the original design can be found in [12]. More recent documentation includes the changes made in v1.0 to improve scaling [13].

Dinh et al. created a benchmark for blockchains where they compared the performance of Hyperledger Fabric, Ethereum, and a database called HBase [8]. Their numbers were used to motivate the choice of blockchain project, and in the comparison to traditional databases.

Kshetri, a proponent for the role of blockchains in the upcoming *Internet of things* (IoT) boom, has written a report about how blockchains might be utilized to solve upcoming challenges with IoT. His arguments are brought up when discussing advantages compared to centralized databases.

Johansson's thesis used blockchain technology, Hyperledger Fabric in particular, to achieve immutability in the Transport Data Logger system [6]. He also makes an argument for when to use blockchains, and more specifically, whether permissionless or permissioned blockchains should be used. This thesis built a new security related application on top of Hyperledger Fabric, and evaluated the result.

# Chapter 2
# Theoretical Background

This chapter will explain relevant background information needed in order to understand the thesis. Hashing is central in a blockchain, and also used in the PoC. Asymmetric encryption is needed to understand how identities are used in a permissioned blockchain. PE file format is used in the PoC to identify versions. Different ways to analyse an executable to detect malicious behavior is discussed in relevance to the PoC, and therefore explained here. The blockchain section goes through most of the important aspects of a blockchain.

## 2.1 Hash

A cryptographic hash function is a function that takes in an arbitrary amount of bits as input, and outputs a bit sequence of fixed length, see Fig. 2.1 for an example. Meaning, a hash of Shakespeare's entire work has the same length as a hash of "Hello, world!". Bruce Schneier [14] described hash functions as "One-way functions", and that they have two important properties. First, it is easy to calculate the hash of data, but infeasible to reverse the function and calculate the data given a hash. Secondly, they are collision-free. Essentially, it is impossible to find two messages resulting in the same hash value. A more formal description of these properties is given as:

- **Preimage resistance:** Given $y$, it should be practically impossible to find $x$ such that $h(x) = y$
- **Second preimage resistance:** Given $x$, it should be practically impossible to find $x'$ such that $h(x) = h(x')$
- **Collision resistance:** It should be practically impossible to find $x$, $x'$ such that $h(x) = h(x')$

Furthermore, it is ideal if the change of a single bit transforms the hash into something completely different. Each bit in the hash should change with the probability of one-half. This property is called the "avalanche effect", coined by Horst Feistel [15].

Hash functions are used in abundance with communication protocols, signatures, password handling, proof-of-work, and integrity verification. It is probably

the most versatile cryptographic primitive, but according to Schneier [16] it is also the "least-well-understood" primitive.

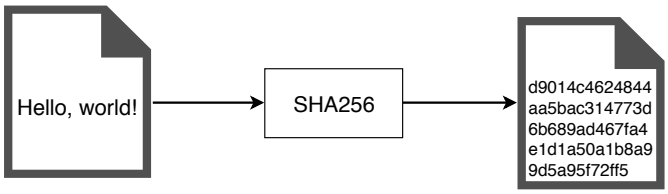


**Figure 2.1:** SHA256 digest of "Hello, world!"

## 2.2 Asymmetrical Encryption

Asymmetrical encryption, also known as public-key cryptography, is a relatively new way of encrypting information. The alternative, symmetrical encryption, has been in use for thousands of years. The Romans famously used the "Caesar cipher" to encrypt sensitive information. In symmetrical encryption, the same key is used to encrypt and decrypt the message — hence the name.

On the contrary, asymmetrical encryption does not utilize the same key for encrypting and decrypting data, key pairs are used instead. A key pair consists of two keys: a public and a private key. The public key is, as the name suggests, public and can be sent to anyone. The private key however, should never be shown to anybody apart from the owner.

### 2.2.1 Encryption and Decryption

Alice wants to send an encrypted message to Bob. Bob sends his public key to Alice, who then encrypts the message using Bob's public key. The encrypted message is sent through an unsecured channel to Bob. Bob uses his private key to decrypt the message, and retrieves the original plaintext. The process is illustrated in Fig. 2.2.

### 2.2.2 Signing and Verifying

Signing solves the problem of "How can I be sure this message originates from Bob?". The process is similar to that of encryption, but reversed. Now Bob "encrypts" the message using his private key, and Alice "decrypts" it using his public key. As Bob is the only person with access to his private key, he is the sole person able to generate a valid signature. It is worth mentioning that the signature is most commonly computed on the hash of a message, not the actual message itself. The signed hash is sent alongside the message. Alice calculates the hash of the message, decrypts Bob's signed hash and if the two are identical she can be sure the message originates from Bob. The process is illustrated in Fig. 2.3.

A signature can only be trusted if Bob's key remains a secret. An attacker with Bob's private key would have the ability to sign anything in Bob's name — making it impossible to differentiate between the two.

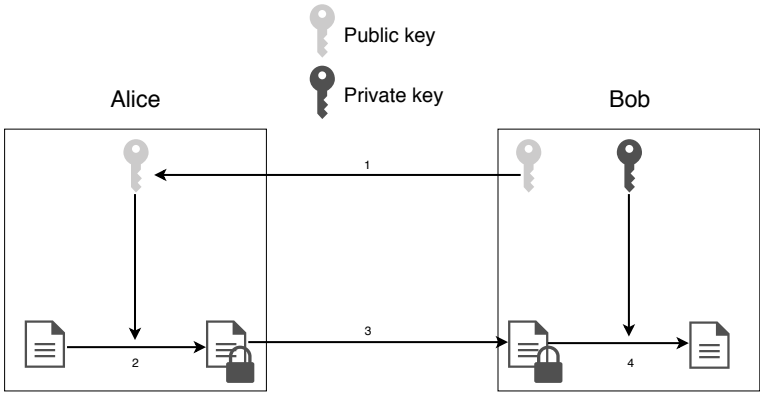


**Figure 2.2:** Asymmetrical encryption.
1. Bob sends his public key to Alice.
2. Alice encrypts the message with Bob's key.
3. Alice sends the encrypted message to Bob.
4. Bob decrypts the message with his private key.

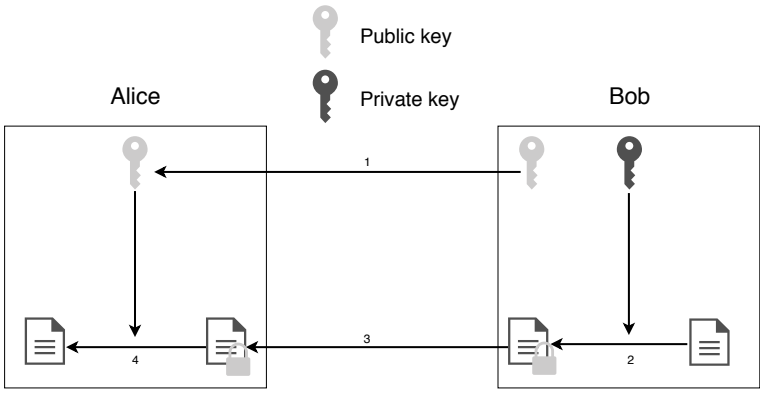


**Figure 2.3:** Signing.
1. Bob sends his public key to Alice.
2. Bob signs the message with his private key.
3. Bob sends the signed message to Alice.
4. Alice verifies the message with Bob's public key.

### 2.2.3 Certificates

How can Alice be sure she received Bob's public key, and not an attacker's? Alice could contact Bob through another communication channel, say a phone call, and ask if she has got the correct key. But this is infeasible for large scale communication between multiple parties, and in general when latency is of importance. Also, it assumes Alice knows Bob from the start. What if Alice has never met Bob, but would like to communicate securely?

If a third party, trusted by both Alice and Bob, verifies Bob's identity and signs

his public key, Alice will be able to trust the validity of the received key. A third party with the capability to make such decisions is called a *Certificate Authority* (CA). This structure, with trusted CAs handing out certificates, is a cornerstone for secure connections on the web. Transport Layer Security (TLS) uses *X.509*[1] certificates, signed by CAs. Every major web browser comes with more than 100 root certificates signed by CAs. The CA does not require anyone else to sign its certificate however, as it is trusted from the beginning. Instead, it signs its own certificate, as illustrated in Fig. 2.4. A structure with public keys is commonly called a *public-key infrastructure* (PKI). TLS uses a hierarchical structure, but examples of more distributed structures exist as well.

Certificates are similarly used to verify software. A software publisher is able to sign software with their certificate, and include the entire chain in the PE header. If signed, the chain is found in the `CertificateTable` entry in the PE header, as seen in Fig. A.1 in the Appendix. Operating systems contain additional CA certificates, and these are used to verify the integrity of software.

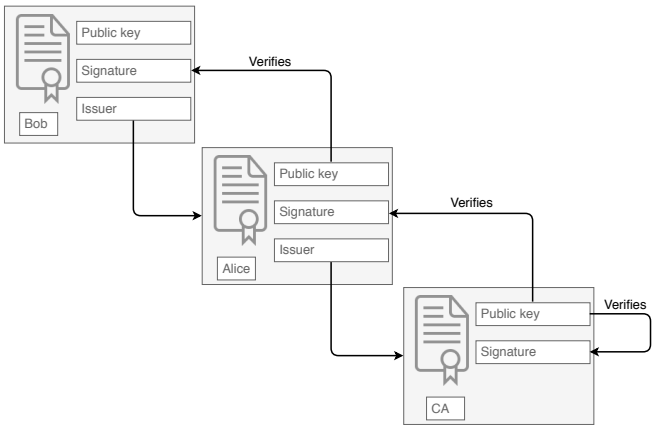


**Figure 2.4:** Certificate chain.

## 2.3 Portable Executable File Format

PE is the file format designed by Microsoft to be used with executable files, object code, and dynamic libraries [17]. A comparison could be made to ELF in Linux, if the reader is familiar with it. It contains information needed by Windows when loading the program into memory. The operating system is able to retrieve the location and size of sections such as `.data` and `.text` from the PE. Figure A.1 in the Appendix shows the entire layout of the PE format. The location of a PE header is always found at the address `0x3c`. This thesis is only interested in extracting a few features found in the PE header. Namely, `ProductVersion`, `FileVersion`, `InternalName`, and `FileName`. All are found in the `ResourceTable`.

[1] *X.509* is a public key certificate standard.

An optional signature of the file can also be found in the Data Directory, in the `CertificateTable` entry. This entry is empty if there is no signature and the size is set to zero. Otherwise, it contains the entire signature chain, and ideally ends with a CA signature.

## 2.4 Malware Detection

How are anti-virus programs able to tell malicious software from benign ones? There is no clear solution to the problem, and it is still an area of research. The malware industry reacts quickly to implement new circumvention and obfuscation techniques.

In general, malware detection is divided into three categories: static, dynamic, and heuristic analysis.

### 2.4.1 Static analysis

Static analysis refers to the method which does not execute any code. Instead, the code and/or binary executable is analysed. If the software in question is proprietary it might not be possible to analyse the code. Anti-virus software normally uses static analysis in order to determine if a program is benign or not [18]. A signature is calculated from a software in testing, and if the signature matches any known malware the software is flagged as malicious.

Static analysis' strong point is the speed of which an answer can be given regarding the maliciousness of software.

The downside with static analysis is that more often than not, the software is proprietary — only giving the analyst access to the binary executable, and not the code itself. Furthermore, the malware industry is quick to develop new obfuscation techniques and hide the functionality of malware. Moser et al. concluded that signature detection is not able to detect malware using obfuscation and transformation techniques [19].

### 2.4.2 Dynamic analysis

Dynamic analysis refers to the method that executes code and analyses the behaviour and functionality. The analysis is carried out inside a sandbox, where the monitoring program is able to retrieve information about the execution. Examples of interesting information include: URLs accessed, file usage, and system calls [20]. The information is gathered into a report which is used to determine if the software is malicious or not.

The downsides with dynamic analysis is the fact that the program has to be executed before a decision can be made, slowing down the process severely. However, given the nature of dynamic analysis, it is able to detect malware even if they use obfuscation techniques.

To counter this, the malware industry continually develops new techniques to detect sandbox environments. If a sandbox is detected, the malware behaves as a harmless program. The sandbox developers are constantly trying to counter the malware industry's detection techniques.

### 2.4.3 Heuristic analysis

Bazrafshan et al. [18] described heuristic analysis as a method using "data mining and machine learning to learn the behaviour of an executable file". The analysis can be performed either dynamically or statically. A visualization of both versions is found in Fig. 2.5.

In a paper by De Lille et al. [21] a proposed way to detect malware with combined static heuristic techniques achieved an accuracy of 97% with a cost of 16.1 seconds. The accuracy was defined as

$$(TP + TN)/(TP + FN + TN + FP)$$

where $TP$ is true positive, $TN$ is true negative, $FN$ is false negative, and $FP$ is false positive.

Heuristic analysis is commonly used by anti-virus program as a supplement to static signature-based methods. Anti-virus scanners use this to detect malware of which there are no signatures yet, but the algorithm cannot be too aggressive, lest it flags benign software.

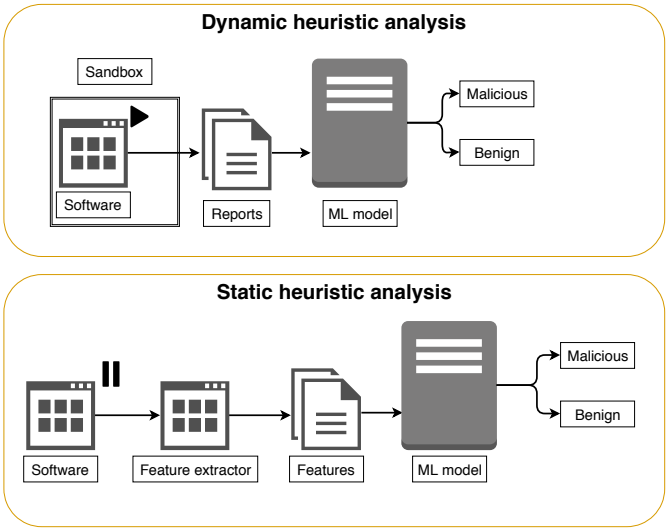


**Figure 2.5:** Heuristic analysis of software with machine learning classification

## 2.5 Blockchain

A blockchain is a chain of blocks, each block containing information. Blocks are chained together in chronological order by a header in each block containing the hash of the previous block header together with the hash of the data contained in the current block [11], as can be seen in Fig. 2.6. Each block added to a blockchain increases the integrity of all previous blocks, since even small changes in the data

represented in earlier blocks would result in a large change of the resulting hash. A blockchain is usually used in a peer-to-peer network where each node in the network keeps a copy of the blockchain and appends it with blocks according to a set of rules.

It is helpful to divide the blockchain concept into two subgroups, permissionless and permissioned.

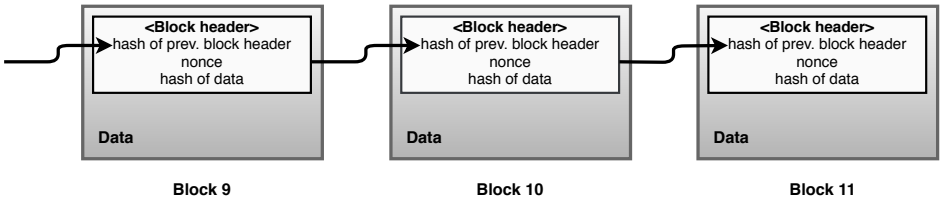


**Figure 2.6:** Chain of blocks.

### 2.5.1 Permissionless Blockchain

A permissionless blockchain requires no prior permission for a user to connect to the blockchain and expand it. Anyone is able to contribute as a verifier of the blocks to be added and the order in which they are added in the chain. These blockchains are therefore also referred to as "open" or "public" blockchains. The main advantage of a permissionless blockchain is the possibility of full decentralization [11].

Bitcoin [11] and Ethereum [22] are two famous examples of permissionless blockchains, both used for cryptocurrencies — digital currency that uses cryptographic primitives to secure its transactions.

#### Double-spending

One of the main difficulties with a digital currency is double-spending. Since the money is represented as a digital file, it is susceptible to falsification and duplication. A way to circumvent this is through the use of a trusted third party who keeps track of the money and makes sure a user cannot spend the same money in multiple transactions. This solution requires all users to fully trust the third party, therefore Nakamoto [11] suggested a different approach:

1. A client requests a transaction by broadcasting it to all nodes.
2. Each node collects new transactions into a block.
3. A time-consuming task called proof-of-work is performed by each node for their block.
4. When a node has completed its task, it broadcasts its block to all other nodes.
5. The other nodes accept the block only after they have verified the task has been performed, all the transactions in the block are valid and the money has not already been spent.

6. The nodes use the hash of the accepted block as the previous hash when working on creating the next block.

Anyone can then confirm the transaction has been processed by checking the block in the blockchain. Consensus based on proof-of-work lacks one important property: Consensus finality [23]. This property guarantees a valid block, appended to the blockchain at some point in time, will never be removed from the blockchain again. This is due to an effect called *forking*, and it arises when the network disagrees on which block to add to the blockchain. The effect is explained in further detail in Section 2.5.1.

### Proof-of-work

To reach consensus on ordering and validity of events in the blockchain, a system of proof-of-'X' is used. Commonly proof-of-work, as is used in Bitcoin [11] and Ethereum [22], but other variants exist such as proof-of-stake, proof-of-capacity, proof-of-research, and many others too numerous to mention.

The proof-of-work algorithm as presented by Nakamoto [11], uses inspiration from HashCash [24], where the main idea is "hard to compute, easy to verify". The block is hashed together with a nonce, a number with no significance other than to change the resulting hash. The goal is to find a nonce such that the hash fulfils a criterion, for example a number of leading zeros as used in Bitcoin. The work needed to find such a hash is exponential with the amount of leading zeros and can therefore easily be modified to keep the problem sufficiently hard. Verifying work will, on the other hand, only require hashing the block with its suggested nonce, and verifying that the criteria with leading zeros is fulfilled. Trying to calculate a hash that fulfils the criterion is referred to as "mining", and mining nodes are called "miners".

One large drawback with the proof-of-work approach is the amount of energy required by the mining process, especially since energy is consumed to solve a problem specifically designed to be time-consuming. One estimation is that a single Bitcoin transaction consumes as much energy as needed to power an average sized household in United States for almost twenty days [25]. All Bitcoin mining consumes more energy than all of Ireland, and almost as much as Denmark, at 29.05 TWh per year [26]. Explanations of alternatives to proof-of-work is given in Table 2.1.

| Consensus protocol | Explanation |
|---|---|
| Proof-of-work | Uses computational power to solve a hard task. Example is hashing given data with a nonce until a criterion on the hash is met. The criterion can be adjusted to extend the average time before a solution is found [11]. |
| Proof-of-elapsed-time | Uses Intel's trusted computing platform, SGX, to randomly choose an amount of time to wait [27]. SGX can create proof of the device waiting for that amount of time and the proof is easily verified by other SGX platforms. This can be seen as simulating proof-of-work, avoiding the unnecessary and energy consuming computations while still being time-consuming. |
| Proof-of-capacity | Also known as proof-of-space. Uses a non-trivial amount of storage to solve a challenge [28]. Writing data to a disk consumes far less energy than calculations on a processor would, making it less energy consuming than proof-of-work. |
| Proof-of-stake | Uses combinations of random selection and wealth or age of coins to select the creator of the next block. One example is PeerCoin [29] which uses time since a coin was last used in a transaction as the age of a coin. A user can increase the likelihood of being selected as the next creator of a block by setting up a transaction back to herself, meaning the age of the coins is reset when the block is created. This results in low energy consumption compared to proof-of-work. |

**Table 2.1:** Short explanations of different proof-of-'X' protocols.

### Fork

There is a risk that two nodes working on two different blocks find solutions to the proof-of-work problem at roughly the same time. This would result in two different chains of the same length and both being as valid as the other. This is referred to as a *fork* — similar to a river that forks into two different paths. Other nodes may work on any of the two forks and individually consider it the primary chain. A "longest-chain" rule is used in Bitcoin and many other blockchains, meaning the first chain to be extended with an extra block will be considered the primary chain by all others, and those who were working on the other fork will switch to

the now longer one. There is a risk of several forks being extended with new blocks at the same time and both staying alive as the "primary", but one of them will eventually be longer than the others, meaning other forks, and blocks added to them, will be abandoned and disappear, illustrated in Fig. 2.7. Due to forks, it is often recommended to wait until five blocks have been added after the block with the transaction of interest before considering the transfer as completed [30]. In Bitcoin, the work required to create a new block is adjusted so new blocks are found on average 10 minutes apart [23]. This results in Bitcoin having a practical transaction time of about an hour.

There exist examples where a blockchain has deliberately been forked by changing underlying rules and convincing a large part of the users to comply with the new rules. This is known as a "hard fork" if the changes makes the two forks incompatible with each other, illustrated in Fig. 2.8. One example is the hard fork to revert the so called "DAO hack". Decentralized Autonomous Organization (DAO), was a crowdfunding campaign using a smart contract where donors would vote on how the money should be spent. A mistake in the smart contract code was abused and a third of the DAO funds were stolen, with a value at the time of USD 55 million [31]. The hack locked the money inside the Ethereum chain for 28 days and a hard fork to undo the damages of the hack was suggested and approved by parts of the community [32]. The divide in the Ethereum community resulted in a split of Ethereum where Ethereum Classic was created and kept mining on the original blockchain while Ethereum started mining on the hard fork. Hofmann et al. [33] argue the hard fork raises concerns about immutability in public blockchains.

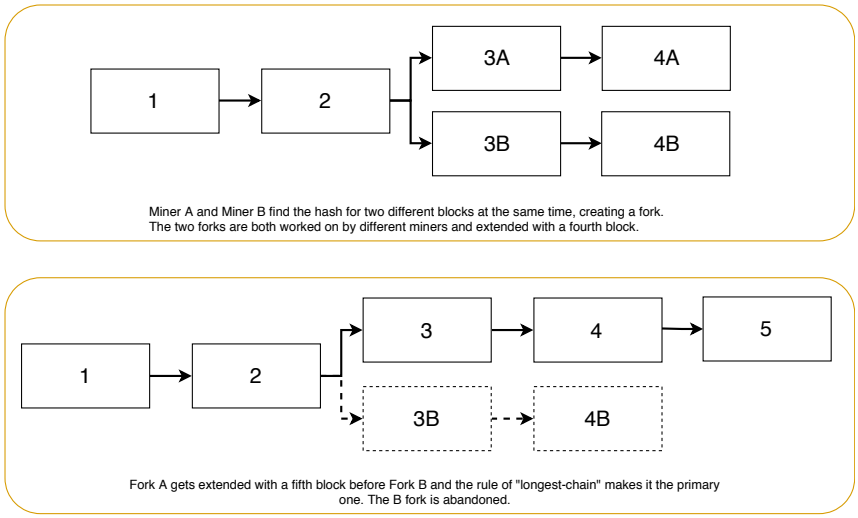


**Figure 2.7:** Blockchain fork.

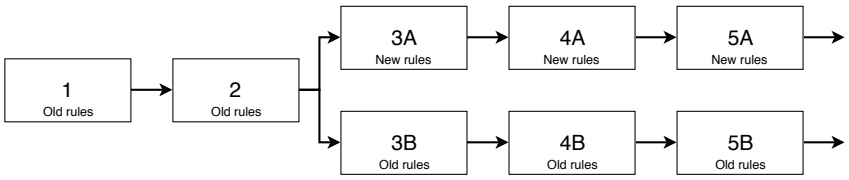


**Figure 2.8:** Hard fork in a blockchain. In the case of the DAO hard fork in Ethereum, the $B$ chain with miners staying with the old rules would be Ethereum Classic and the $A$ chain following the updated rules would be Ethereum.

### 2.5.2 Permissioned Blockchain

A permissioned blockchain is operated by identifiable and verifiable entities [12]. These entities are responsible for validating and ordering the transactions. Proof-of-work is not always used in permissioned blockchains, instead consensus algorithms are often used. Different consensus algorithms exist with individual trade-offs between throughput, scalability, and tolerance of hostile environments.

#### Byzantine nodes

The name Byzantine node comes from "The Byzantine Generals' Problem" [34], a problem described by Lamport et al. as early as 1982. The problem is as follows: Imagine several divisions of the Byzantine army are camped outside of an enemy city, each division commanded by its own general. The generals can communicate with one another only by messengers. It would be a disaster if some generals retreat while others attack. In order to win, they must decide upon a common plan of action. However, some generals may be treacherous, trying to prevent loyal generals from reaching an agreement.

The generals must have a predefined algorithm guaranteeing all loyal generals decide upon the same plan of action, while hindering a few traitors to decide on a bad plan.

A simple algorithm is to go for the option most generals have voted for. In a small example with three generals, one general ($A$) might vote for retreat and one ($B$) for attack. The last general, if traitorous, can then send $A$ a vote for retreat and $B$ a vote for attack. $A$ attacks, while $B$ retreats, leading to a loss for the generals. An example is illustrated in Fig. 2.9. Introducing one more loyal general to the example, and adding a rule of action in case of a draw (for example everyone retreats if it is a draw), would solve the problem in this example. Algorithms and protocols that are able to function with Byzantine nodes have a property called Byzantine Fault Tolerance (BFT).

The Byzantine Generals' Problem, though explained in a historical setting, is directly relatable to computer systems. It is reasonable to think faulty components might deviate from a protocol and unintentionally behave as a traitorous general, often referred to as "Byzantine behaviour". A Byzantine node is a node with Byzantine behaviour, either because it is malfunctioning or controlled by a malicious entity.

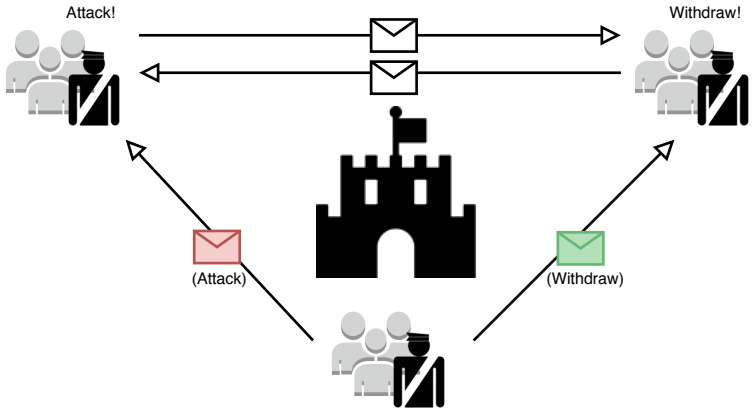


**Figure 2.9:** Byzantine Generals' Problem with three generals. Middle one is traitorous and sends different messages to the other two.

### Practical Byzantine Fault Tolerance

A common and well-researched consensus algorithm is Practical Byzantine Fault Tolerance (PBFT), presented by Castro and Liskov in 1999 [35]. The algorithm offers resiliency under the assumption that $n = 3f+1$ , where $n$ is the total amount of nodes and $f$ is the amount of Byzantine nodes, which is proven to be optimal. A simple explanation of the algorithm in a blockchain context:

1. A client sends a request to add data to the current primary node.
2. The primary node forwards the request to the other nodes.
3. All nodes try to validate and append the data to the blockchain, and send the result back to the client.
4. The client waits for replies until it has received $f + 1$ replies confirming or denying the request.

Any node can, and will, be selected as primary at different times. The optimal resilience of $n = 3f + 1$ can be verified with a worst-case scenario: $f$ nodes do not reply. This might be because they are offline, or their response is for some reason delayed. The client will be left with $2f + 1$ replies to figure out if the request was accepted or not. Since $f$ is the maximum amount of Byzantine nodes, there is still a majority of accurate replies in a situation where the $f$ nodes that have yet to respond are not faulty but just slow. This is illustrated in Fig. 2.10.

## 2.5.3 Hyperledger

Hyperledger is an umbrella project with different open source blockchain projects. It was started by the Linux Foundation in 2015 [36], and is supported by many large companies both in the technology industry and financial sector [37]. Some of

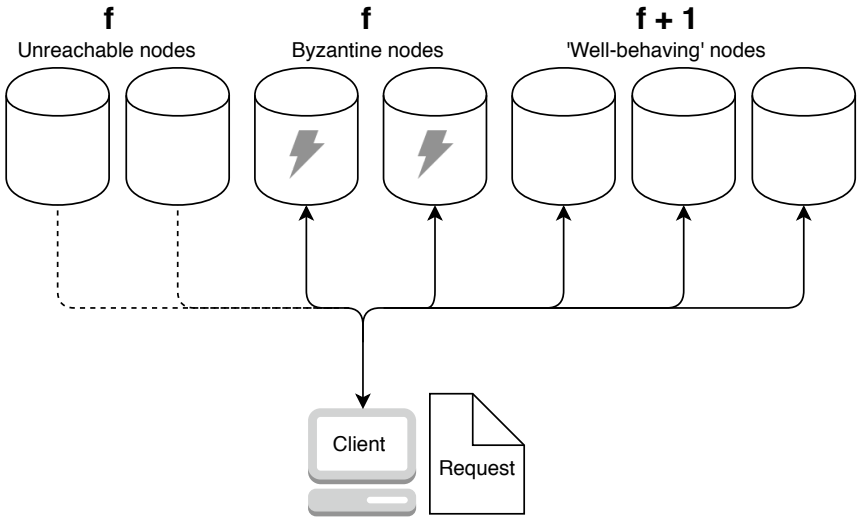


**Figure 2.10:** Resilience in PBFT.

their blockchain-based distributed ledger projects include Hyperledger Sawtooth, Hyperledger Burrow, and Hyperledger Fabric. Tools made for quicker and easier deployment, and creation of smart contracts have also been released.

### Hyperledger Fabric

Hyperledger Fabric [12], henceforth referred to as Fabric, is one of the more mature options when it comes to permissioned blockchain projects. It is made to be adaptable in a way where current and future standards should be easy to use with Fabric. The project uses member lists, where the default is a system with `X.509` certificates, but this can be changed to work with other standards and other methods of proving an identity. The system does not use consensus based on proof-of-'X', instead several consensus algorithms are implemented, for example Kafka[2] and PBFT. The consensus layer is modular, meaning that if faster algorithms are discovered in the future they can be implemented into Fabric.

Some key aspects about Fabric:

**Assets:** An asset in Fabric is represented as a key-value pair, similar to maps and dictionaries in many programming languages [38]. All mappings of a key is stored on the blockchain but the most recent value is the one that is retrievable. The value is a binary string, meaning that anything that can be digitally represented can be stored.

An example of an asset is a farm animal. The key can be a unique identification number or name of that farm animal and the binary string can be relevant

[2]A crash tolerant consensus algorithm. It is not BFT [13].

information such as weight, cost, and age.

**Membership services:** All members of the system need to be identifiable and approved to partake in the network. PKI is used to create certificates that can be tied to organizations, client applications, and end users. The identities are provided by the *Membership Service Provider* (MSP) in Fabric. The approach allows configuration of different "roles" with different access and different responsibility in a transaction [38]. This allows scenarios where transactions above a certain value will need to be approved by the company management before they are accepted in the blockchain. The modularity of Fabric makes it possible to synchronize the membership services to structures like Active Directories that might already exist within a company.

**Chaincode:** Chaincode is the term used in Fabric for smart contracts, see Section 2.5.4, and is a piece of code that is stored in the blockchain and run on nodes in the network. The code can access and change assets [38]. The Hyperledger white paper [12] defines a smart contract in the following way:

"A smart contract is a collection of business rules which are deployed on a blockchain, and shared and validated collectively by a group of stakeholders. A smart contract can automate business processes trustfully by allowing all stakeholders to process and validate contractual rules as a group.".

**Consensus:** Consensus in Fabric incorporates many parts. A transaction goes through several steps before it is finally added to the blockchain [39], see Fig. 2.11 for an illustration of the process.

1. Farmer $A$ wants to sell one of his farm animals to a customer $B$. One of them initiates the transaction through an application. The transaction is formed in a way where endorsement from both $A$ and $B$ is required for the transaction to be valid.

2. The transaction is sent to the two nodes representing $A$ and $B$. They both verify that the transaction proposal is valid and that the other party is eligible to make the changes required to the asset. The values in the transaction are used in the invoked chaincode and executed against the blockchain.

3. As a result, the transaction and the endorser's signature are sent back to the initializing application, which collects the responses from all endorsers, packages them into an update request and broadcasts it to the orderers.

4. The orderers order all requested transactions chronologically and collect them into a block that can be added to the blockchain. The block is then sent to all nodes in the network.

5. Each node validates all transactions in the block and that the endorsement policy was upheld (both $A$ and $B$ endorsed the transaction). If the block is valid, the nodes append it to their blockchain and notify the application which initiated the transaction that the transaction has been immutably appended to the chain.

The endorsement policy used in the chaincode invoked in step two can be made more advanced. A medical journal for the farm animal may exist and a third party, for example a veterinarian, may be required to also sign the transaction. The customer may work for a company and all purchases made by him could be required to also be signed by superiors.

In step four, an algorithm for consensus is used by the orderers. If PBFT is used, the block is only approved if more than one third of the total amount of orderers have agreed on the order.

### 2.5.4 Smart contracts

A contract, like a contract for renting an apartment, is written to be legally enforceable. Judges and lawyers are used to interpret the contract to ensure both parties fulfil their contractual obligations. In a smart contract, that responsibility is instead moved to the code and to the masses. Szabo defined smart contracts in 1997 [40]. They are often constructed as "if $X$, then $Y$", where $X$ are requirements to be fulfilled by a buyer before the service $Y$ is provided. With the rental agreement as an example, a requirement to fulfil by the tenant is to transfer rent to the landlord each month. More complex logic such as "if there are insufficient funds on a tenant's account one month, a fee is added to the total" can also be used.

The code is readable by everyone and the distributed nature of blockchains guarantees that the execution is performed as coded. This is achieved by having the code being run by all nodes. If execution of the code results in a transfer of money or a digital asset that exist on the blockchain, all nodes will come to this conclusion when running the code and the asset will be transferred.

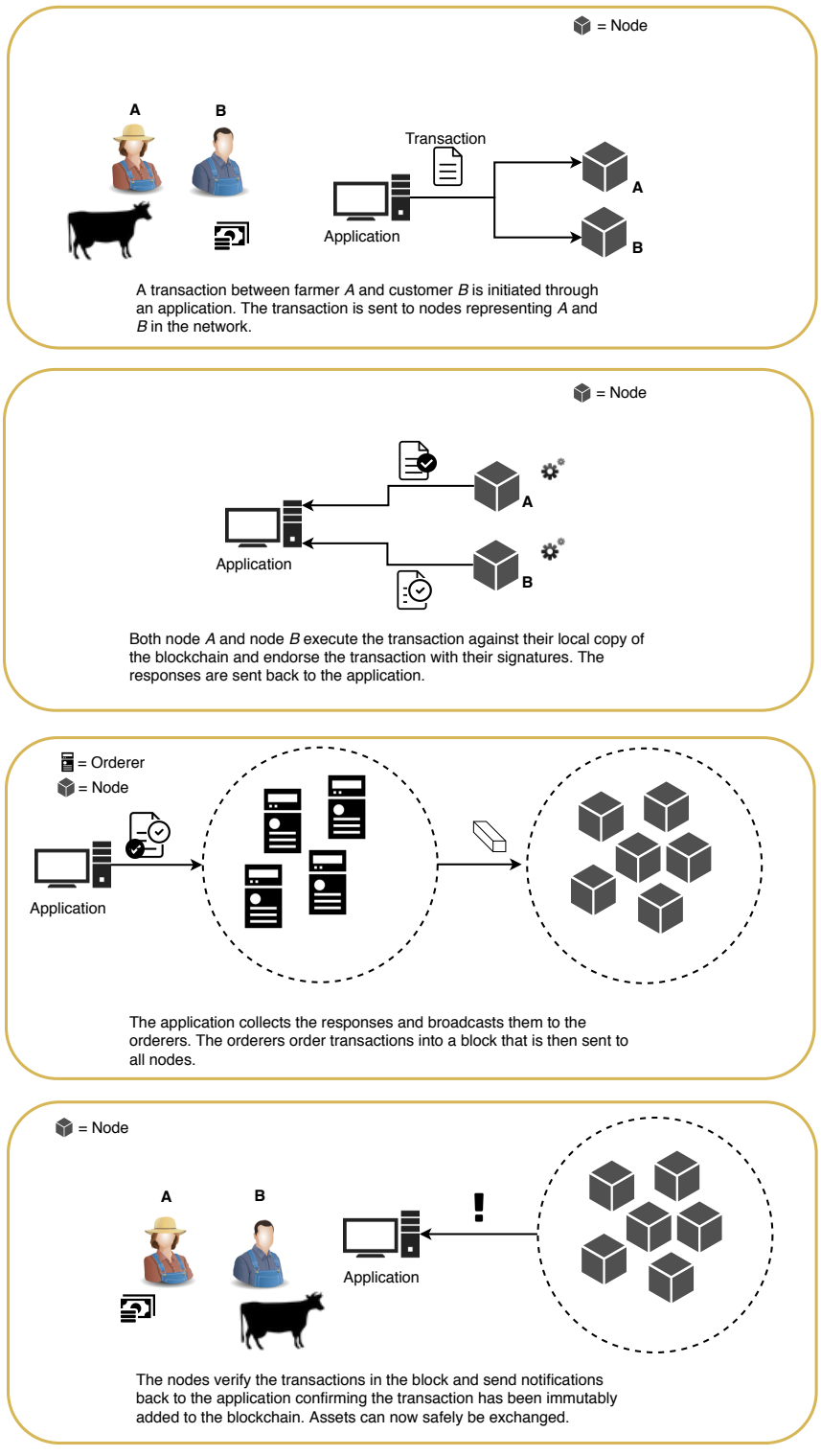


**Figure 2.11:** The transaction flow in Fabric v1.0

# Chapter 3
# A security related PoC

A security related PoC benefiting from the blockchain technology was designed and implemented to better understand potential limitations with the technology. The main idea behind the PoC can be summarized as follows: a collection of users share information of programs they run in an attempt to detect, and not execute, programs not trusted by the collaboration. A first assumption is that the participants do not fully trust each other, but believe a majority of the participants act in a "good" way. A centralized database would give full power to a third party to change or remove any data it pleases. A blockchain on the other hand would not require a trusted third party, and no data can be removed or changed as long as not a majority decides to do so.

The information the PoC gathers from executable files include: hash of the executable, name, version, and the certificate if it is signed. A positive or negative vote is cast by the user, currently by deciding if she will run the executable but other options are discussed in Section 4.1. This information from all participants is stored on the blockchain. The next time a user wants to execute an executable, a piece of chaincode is run on the blockchain and the total amount of points for that exact executable, and any other versions of the same program, are retrieved. The response includes information about other executables with the same name that are signed (or signed with a different certificate).

The PoC can be helpful in the following scenarios, all considering a collaboration of fifty users:

- An executable has 40 points. Running this executable does not deviate from the behaviour of the others.
- An executable has zero points. If the executable is something one could assume more people would run, for example Skype.exe, this might indicate a spoofed version from an unreliable source.
- An executable has two points. A previous version has 40 points.
- An unsigned executable has one point. Information of a signed executable with the same name exists and has 40 points. This might indicate that the executable is spoofed.

# Chapter 4
# Motivation of the Design

This chapter aims to explain and motivate two major choices for the PoC. Several ways for a participant, or entity, to form an opinion on an executable were explored and limitations discussed in the first part of this chapter. The second part motivates the choice of underlying blockchain structure.

## 4.1 How can an entity form an opinion?

The following sections will discuss various ways for a computer to form an opinion regarding a piece of software. The opinion is what will be shared on the blockchain, and allow others to make decisions based on the shared opinions.

### 4.1.1 Possession

A simple way to trust software is to simply base it on whether the computer has the program installed. If the program is found on the computer, it will send an update to the network. This does not require any malware detection — and results in the computer only seeing if a program is used by others on the network. Meaning, if a malware has spread to enough computers it will be deemed as trustworthy by the network.

It would provide protection against spoofing/phishing attacks. Say a user receives a phishing email, containing a Word document. The document then executes a macro to download a malware named "Skype" and tries to infect the computer. The victim will notice, after querying the distributed ledger, that other computers have reported entirely different hash and/or certificate. The system could then react and stop the execution of the program.

### 4.1.2 Previously seen software

This approach builds upon the "Possession" method. Instead of simply trusting software because it is installed on the computer, the entity will try to match software with previously seen ones. The comparison could be done in any amount of ways. One option is to use the PE header to find previous versions of the software. The system is then able to verify that the previous versions are signed with the same private key. Software with no known previous versions, or known

ones with a differing certificate, are considered as potential threats and the user receives a warning.

This approach tries to minimize the need of input from the user, and only requires it when no further steps can be taken automatically. The difficult part is how the entities are to reach the conclusion that a piece of software is malicious, after having upvoted it in the past. There is also a reliance on the user having the ability to determine the maliciousness of software based on the data that is presented by the system. Examples of data could be how other entities have voted, and information about the certificate.

### 4.1.3 Anti-virus software

The computer could trust the anti-virus software and view the program in question as safe if no warning is received. No warning is interpreted as the software is safe to run. A warning results in a negative opinion.

For this approach to work, there would have to be multiple variants of anti-virus software on the network. Otherwise, it turns into a slow version of said anti-virus software. However, if multiple variants are used the system would allow computers to make decisions based on the collective opinion of several anti-virus software providers, mentioned in Section 4.1.5.

As mentioned briefly in Section 2.4.1, most anti-virus programs use signature checking to determine if a program is malicious or not. This allows malware using obfuscation techniques to circumvent the anti-virus program. Consequently, malware would be able to gain upvotes from computers on the network, and achieve trustworthiness.

### 4.1.4 Sandbox and machine learning

A computer could upload a new program to a sandbox environment, where it would be monitored for suspicious behaviour. Afterwards, a machine learning model would classify the reports generated by the sandbox, and determine if the program is malware. This solution is more akin to the dynamic analysis method mentioned in Section 2.4.2.

This would require more time than simply using an anti-virus software, which might lead to more time for the malware to execute its code.

### 4.1.5 User's choice

If the choice is left to the user, as opposed to something the system enforces on the user, a more diverse network could emerge. Some users on the network might choose to use static analysis on their computer, while others use dynamic/heuristic methods. The resulting network would be similar to a distributed version of VirusTotal. Each computer on the network contributes with the result from one malware detection method.

VirusTotal is Google's online anti-virus service. Visitors are able to upload executable files, which VirusTotal then scans through with over 70 anti-virus programs. The result page displays what each of the anti-virus programs determined, see Fig. 4.1.

Entities without anti-virus or sandbox environments could still contribute to the network via the possession-based method that was proposed earlier in Section 4.1.1.

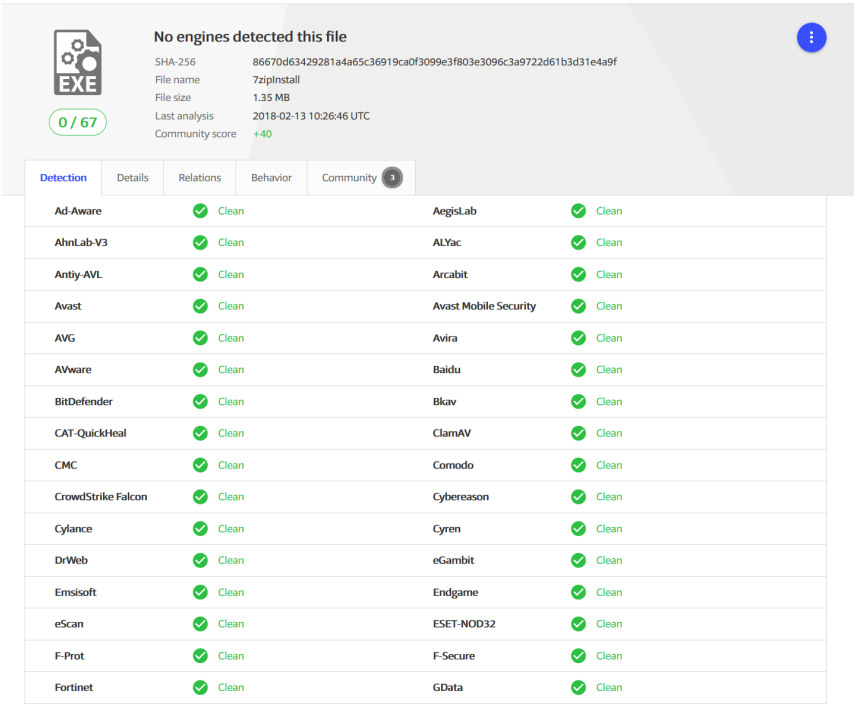


**Figure 4.1:** Result page from VirusTotal

## 4.2 Choice of blockchain structure

Fabric v1.1 has been chosen as underlying blockchain for the PoC, since it supports smart contracts, identifiable members, and BFT consensus without the need of resource heavy proof-of-work. Fabric has shown to scale better than many of its competitors [7, 41], and with its recent stable release, and some supposed real-world applications already, it gives the impression of being mature and stable. The following sections aim to further explain the decision to use Fabric v1.1 for the PoC.

### 4.2.1 Support for smart contracts

The implementation will assume an environment with limited trust for individual entities, and the data submitted should be on a "one-entity-one-submission" basis. Scenarios where an entity would like to update its submission can be imagined, for example if new information has surfaced or if a certificate has been updated with a new signature. To allow an entity to update its vote, while still keeping

the old vote stored on the blockchain, the application would need to keep track of, and only count, the most recent vote. This is easily managed by a smart contract, where a "state" inside of the smart contract and future changes to that state will be tracked by the mechanics which execute the smart contract. Another advantage with smart contracts is that the code will be easily viewable and verifiable by all participants. This stays true to the spirit of the project, where trust for third parties should be kept to a minimum.

There exist both permissionless and permissioned blockchain projects with support for smart contracts. A permissioned blockchain with identifiable members can easily be utilized by the PoC, where the goal is one-member-one-vote. Creating something similar in a permissionless setting, where one user should not be able to connect a thousand different devices and upvote malicious software, is complicated. A system with identities and the ability to somehow reward good behaviour and punish bad would need to be implemented. To cite Johansson on the topic "This would in effect just be implementing a heavy permissioned layer on top of the permissionless blockchain system. Instead, linking blockchain network identities to physical entities can be made *with* a permissioned system that is purpose-built for doing just that." [6]. As Johansson also brings up, using a permissionless blockchain comes with costs that are harder to predict [6], considering the transaction fees in permissionless blockchains. Since there is no monetary interest for users of the PoC, spending money to submit and update data might not be enticing.

### 4.2.2 Scalability

Scalability is another factor of interest. Various publications point to an advantage of permissioned blockchains over permissionless ones based on proof-of-'X', when looking at throughput and transaction speed. A comparison by Dinh et al. [41] between Fabric, Parity, and Ethereum states that Fabric has lower latency per transaction and higher throughput across all tests in the blockchain benchmark framework BLOCKBENCH. Pongnumkul et al. performed tests on Ethereum and Fabric v0.6 with the consensus layer disabled, in order to test the execution layer [7]. They concluded that "Hyperledger Fabric consistently performs better than Ethereum both in [terms] of throughput and latency". In some of their tests, the execution time of a smart contract in Fabric was a tenth of Ethereum's time, something they noted shows a large difference in data access and management for the two platforms.

These advantages are achievable only when few nodes are used in Fabric, more nodes results in less difference between the two. In Fabric v0.6, every node is part of the consensus algorithm. In versions released after v1.0, only nodes assigned to be an orderer is part of the consensus. An implementation of the design mentioned in Chapter 3 will require that a user has trust for a majority of orderers. This can be achieved either by every member being an orderer, or that a small selection of orderers exist allowing users to trust a majority of them.

Dinh et al. [41] had problems with Fabric v0.6, where it stopped working when used with more than 16 nodes in some of their tests. It would be a severe limitation to the PoC if a maximum of 16 members is possible. The report,

however, attributes this to the queue being shared by client requests and consensus messages, and the experiment used clients with a fixed request rate of 320 requests per second per client. The PoC would not need this amount of transactions, at least not after an initialization phase. The frequency of requests per client would probably be in the single digit per day, compared to 320 requests/sec · 3,600 sec/hr · 24 h/day = 27,648,000 requests/day. The solution could potentially scale to nodes in at least the hundreds, even considering PBFT's complexity of $\mathcal{O}(n^2)$ [6].

Since Ethereum is open source [42], it is possible to use it in a private setting where only permissioned nodes can connect. The blockchain would still use proof-of-work for consensus, which raises some extra security issues as it would not require a large investment to obtain 51% of the computing power in a small, private blockchain network. The identity management would also have to be done more centralized if a private fork of Ethereum was chosen. The authors believe the application would be interesting and helpful in a permissionless blockchain, but would require a web of trust to function, see Section 9.1. Due to the application being used in a permissioned setting, Fabric will be better suited, since that is what it is designed for.

### 4.2.3 Other contenders

There is another interesting project under the Hyperledger umbrella with a permissioned structure: Hyperledger Sawtooth. Sawtooth recently got its first stable release, and promises high scalability and low execution time of chaincode [43]. The system currently only offers proof-of-elapsed-time, using Intel SGX, as the consensus algorithm. This locks the application to only support one CPU vendor, and as Johansson [6] also stated, Fabric seems like the more mature project of the two at the time of writing.

### 4.2.4 Differences between Fabric v1.1 and v0.6

In September 2016, Fabric v0.6.0-preview was released on GitHub. In July 2017, the organization presented its first stable release, v1.0.0. This version implemented some major changes, and this section aims to explain some differences between the preview version and their most recent release, v1.1.

#### New architecture and the removal of PBFT

The underlying architecture was changed to increase throughput and scalability in Fabric v1.0 [44]. In v0.6, only nodes and clients exist. A client is an application using their SDK or own implementation to create transactions. Only one type of node exists and every node can be considered a "full-node". All nodes execute chaincode and partake in establishing consensus, and a native implementation of PBFT is used for BFT consensus. In v1.0, Fabric moves to a new approach with an "execute-order-validate" architecture. Nodes are separated into peers and orderers, and only a subset of peers (called endorsers) needs to execute a specific chaincode. The orderers are responsible for the ordering of transactions, and the consensus, while the remaining peers only validate the results presented by the endorsers

after they have executed the chaincode [45]. The changes introduce support of non-deterministic smart contracts and parallel execution of smart contracts. Also, every node is no longer required to execute every smart contract and nodes can even have their own policies for how resource heavy a smart contract is allowed to be before it is dropped [45]. The changes also allows more flexibility for the trust model, where trust assumptions for a smart contract may be different from that of the ordering of transactions.

With the changes, they removed support for PBFT, and in effect only offer algorithms for consensus that are crash tolerant and not BFT. Adding Simple-PBFT is part of their development plan [44, 45] and a yet-to-be-endorsed implementation of BFT-SMaRt exists [46], so the lack of BFT consensus is only temporary.

# Chapter 5
# Implementation

The following section will explain the design and implementation of the PoC in detail. The client, the chaincode, and the blockchain network are all thoroughly explained and illustrated.

## 5.1 Extracting information from executables

The PE header was used to extract information about an executable. Some fields in the PE header are optional, but most legitimate programs will contain the information used by the PoC.

### 5.1.1 Name and version

A Java library called "PE/COFF 4J" [47] was used to extract the name and version from the resource tables found in the PE header.

### 5.1.2 Signature and certificate

As previously mentioned in Section 2.2.2, signatures are used to verify the integrity and origins of digital files. Software from major developers are often signed, although it is optional. The entire certificate chain can be found in the entry called `CertificateTable`, found in the Optional Header — see Fig. A.1 in Appendix A. Locating this entry is done by reading the values at offset `0x98`/`0xa8` (32/64 bit) from the start of the PE header. The size of `CertificateTable` is found after the offset, at `0x9c`/`0xac`. After reading these two values, the PoC is able to retrieve the entire certificate chain and generate a fingerprint of it by hashing it.

## 5.2 Abstract view of the logic

The main flow of logic in the PoC is somewhat complex. A flowchart was drawn to help visualize it. Fig. 5.1 illustrates the logic when a new program is encountered on an entity. If the program is unknown to the network, it will try to match the certificate with previously known versions of said program. A problem is when developers renew their certificate, as they do on a regular basis, the new certificate will not match with the previous one — making it hard to compare them. The PoC

queries the user for the next step at this point. A solution closer to production would most likely have to find a better way to determine if a program is malicious or not.

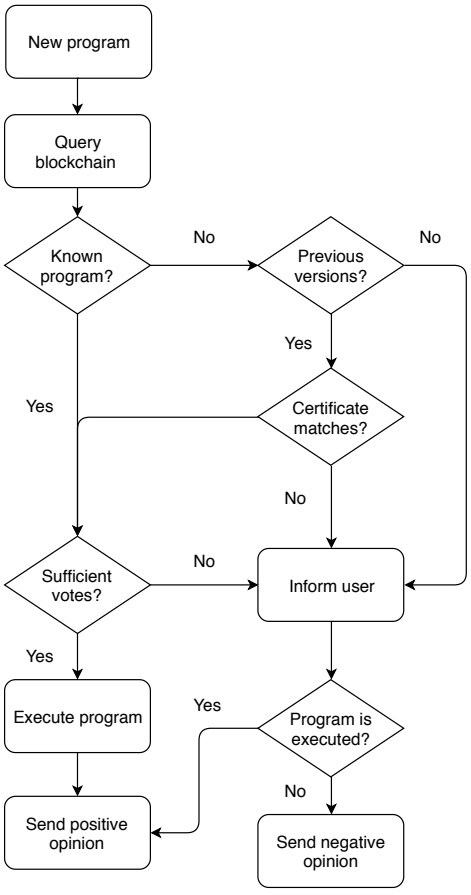


**Figure 5.1:** Flowchart of the system

## 5.3 Fabric network

Each participant in the network will run one peer and one orderer on their computer. Each peer is connected to all other peers and each orderer is connected to every other orderer. An illustration is found in Fig. 5.2. The orderers are set up to use a community developed BFT-SMaRt module [46]. Fabric uses Docker for execution of chaincode, and all participants will have a Docker container running the chaincode.

Fabric is configured to use `X.509` certificates for identification and signing of transmissions. The endorsement policy used requires $\frac{n}{3} + 1$, where $n$ is the total amount of participants in the network, to endorse the chaincode execution for it

to be considered valid and added by the peers to the ledger. The policy is chosen to keep the same BFT properties as with BFT-SMaRt.

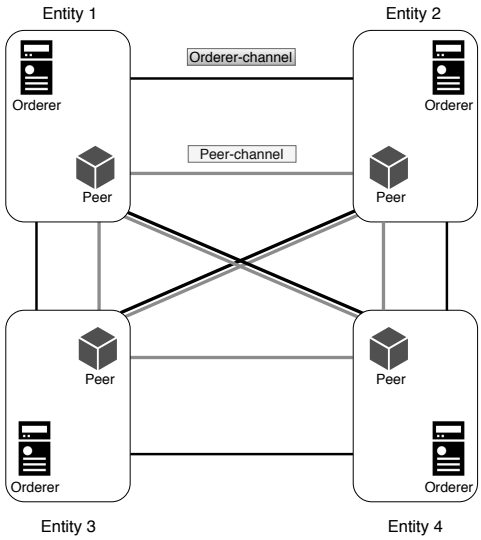


**Figure 5.2:** PoC network with four participants. The PoC can be scaled for more participants.

## 5.4 Chaincode

Fabric uses a state database, and data is stored as a [key]:[value] mapping. The key is a string and the value is an array of bytes, which makes it possible to store anything and map it to a key. The chaincode in the PoC defines a structure in JSON format which makes it possible to map the Vote structure seen in Fig. 5.3 to an array of bytes. The key used is the Base64 representation of the sender's certificate hashed with the hash of the program. Since the hash is unique for each program, and the certificate cannot be forged, any new votes from the same user on the same program would just replace the old one. The old value would be kept in previous block of the blockchain, but only the most recent value would be used by the chaincode. This fulfils the requirement of "one user — one vote" as well as making sure old votes are kept even if they are updated. An illustration of the mapping can be seen in Fig. 5.4.

The chaincode provides three functions: `createVote(string[] args)`, `queryOwn(string[] args)`, and `query(string[] args)`. `queryOwn(string[] args)` takes one argument which is the hash of the program. It simply returns the user's own vote on a particular program, in the form of a byte array.

`query(string[] args)` requires four strings:

- Hash — the SHA256 hash of the programs executable.
- PEName — the name of the program, according to the PE header.

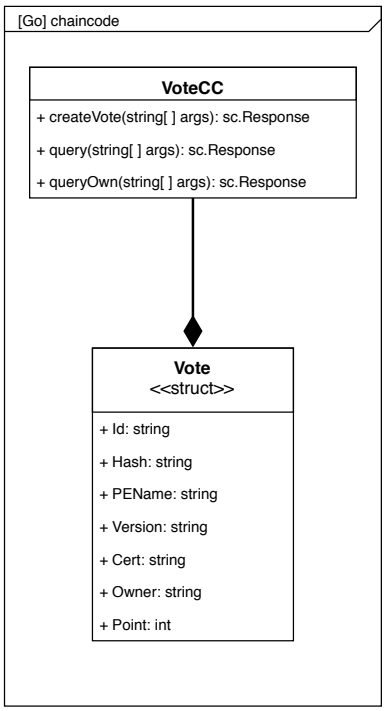


**Figure 5.3:** UML of the VoteCC.go chaincode and the JSON structure defined by it.

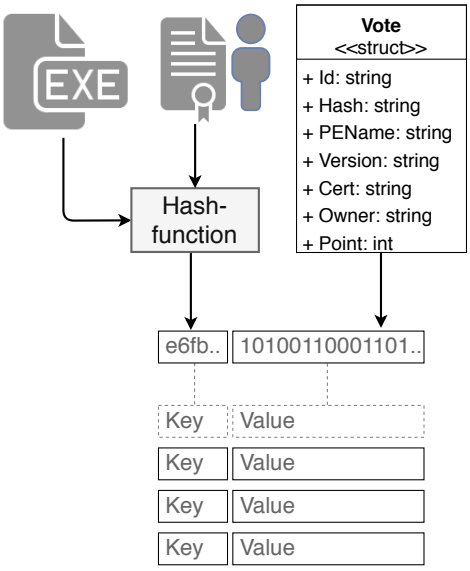


**Figure 5.4:** The key-value mapping used by the chaincode. The key is chosen as the SHA256 hash of the executable and the Base64 representation of the certificate. The value is the byte array corresponding to the JSON structure "Vote".

- Version — the version of the program, according to the PE header.
- Cert — the SHA256 hash of the certificate used to sign the program. The "certificate fingerprint".

It responds with a JSON structure in the form of a byte array which the client parses. Three different types of information is retrieved from the blockchain:

- Total points, and from how many users, on the hash.
- Total points, and from how many users, on hashes with the same name and same certificate, but with different version numbers.
- Total points, and from how many users, on hashes with the same name but different certificates.

`createVote(string[] args)` requires an additional two strings, namely

- Owner — the name of the voter.
- Point — the points given to the program, $p \in \mathbb{Z}, -1 \leq p \leq 1$.

## 5.5 Client

The client consists mainly of a Java background process. The process uses Windows' process monitor to detect started processes, and extracts the information

necessary for querying the chaincode. The possibility to manually give the path to an executable exists as well, and also the ability to scan a file through the Windows Context menu. Fig. 5.5 illustrates the structure of the package that detects started processes and extracts necessary information.

Even though the target platform was Windows, some features are only present on the Linux version. For instance, a significant amount of time was spent on getting the BFT-SMaRt fork to work on first Windows, and later Linux. The Windows version using BFT-SMaRt never got to a functional state, and instead uses the native implementation of Kafka. The Linux version using BFT-SMaRt is a severe kludge solution. Nevertheless, it was functional enough for testing.

`ProcessRetrieverThread` is responsible for finding newly started programs. It keeps track of when the latest search was performed. Programs started after the previous search are considered as unseen. These programs are put into a `List` in the `SystemProcesses` class, and this is where the main program retrieves information from with the method `getProc()`.

`FileRetrieverThread` extends the same abstract class as `ProcessRetrieverThread`. This class is used when running the PoC on a Linux system. The solution for finding running processes was not portable, and as the Linux version of the PoC was only to be used for testing BFT-SMaRt, `FileRetrieverThread` was created. It simple listens for changes made to a specified directory, and considers a change to be a program starting. `.exe` files are placed in said directory to simulate them starting.

The client follows the logic described in Section 5.2. If the hash of the executable has points equal to $\frac{n}{3} + 1$, where $n$ is the total amount of members in the network, the program is considered trustworthy. Likewise, it is considered favourably if another version of the executable has reached the threshold. If the threshold is not reached, the client has no opinion of the software, again to keep to the BFT properties of the rest of the service.

During development and testing a GUI was used to visualize what would happen according to the flowchart, illustrated in Fig. 5.1. The GUI is shown in Fig. 5.6-5.9. "Approved program!" means the GUI would not have shown itself to the user, and the program would be allowed to run. Conversely, "Program not approved" would require user input before deciding to run the program or not. If the user were to press "No", the PoC would terminate the program in question, and enter a negative vote into the blockchain.

The GUI contains technical information that might not be suitable for users with little technical knowledge. This would be streamlined in a consumer version so as not to confuse the users.

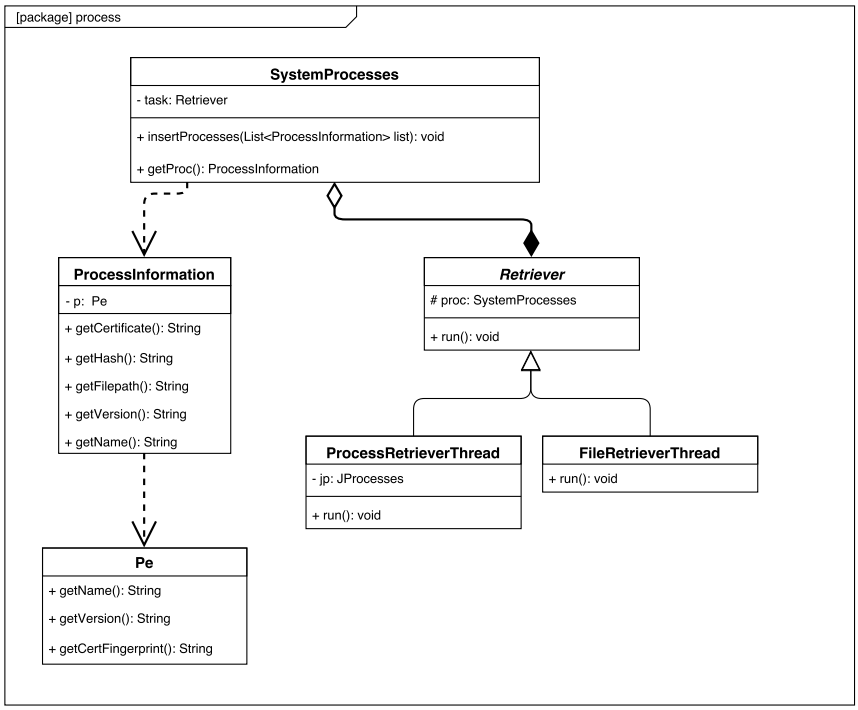


**Figure 5.5:** UML of the process Java package.

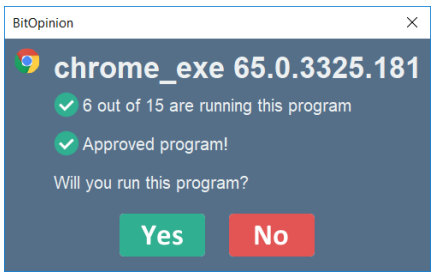


**Figure 5.6:** Hash matches with an approved program

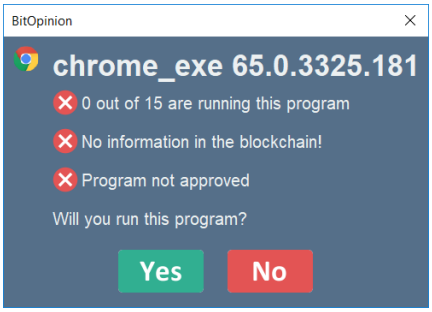


**Figure 5.7:** No information available

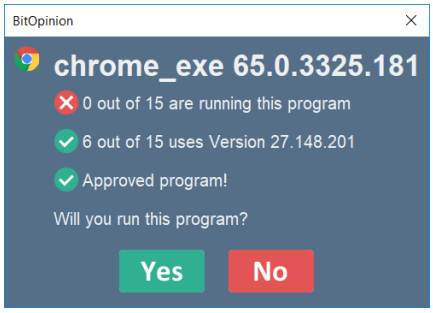


**Figure 5.8:** Previously seen version available

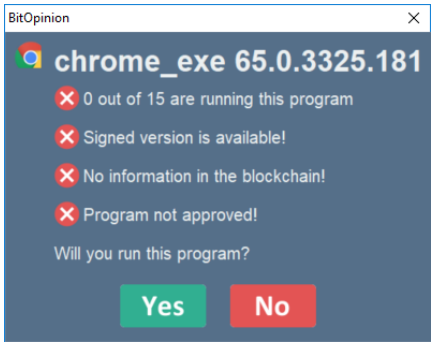


**Figure 5.9:** Signed version available

# Chapter 6
# Testing

## 6.1 Versions used

- Docker 1.13.1
- jcs47/hyperledger-bftsmart 7f459d01
- jcs47/fabric 323402c7
- mcfunley/juds 6621191c
- Ubuntu 17.10

## 6.2 Traffic between orderers

Ordering in permissioned blockchains is often a bottleneck and greatly impairs scaling. The PoC uses BFT-SMaRt for ordering events. The BFT-SMaRt algorithm is similar in theory to PBFT and should require the same amount of packages to reach consensus, $\mathcal{O}(n^2)$. The implementation, and especially the wrapper created to function with Fabric, may however give other results. The following test aims to link the PoC to the underlying theory of the components used to better discuss the scalability of the design.

### 6.2.1 Method

A network containing $n$ number of orderers with different port numbers on localhost and a fixed number of peers was started. The PoC was used to send a vote on an executable (Firefox Installer). The traffic was captured using Wireshark and packages between the very first package that contained the string "createVote" (sent from a peer to an orderer), to the last package containing the same string were counted.

Amount of orderers, $n$, was chosen in intervals between four and 40.

### 6.2.2 Results

The result of the test is found in Table 6.1. The result is plotted in Fig. 6.1.

| Orderers | Total packets |
|---|---|
| 4 | 207 |
| 10 | 1,055 |
| 16 | 3,439 |
| 22 | 5,080 |
| 28 | 7,636 |
| 34 | 10,876 |
| 37 | 13,862 |
| 40 | 14,325 |

**Table 6.1:** Traffic between orderers for certain amount of orderers.

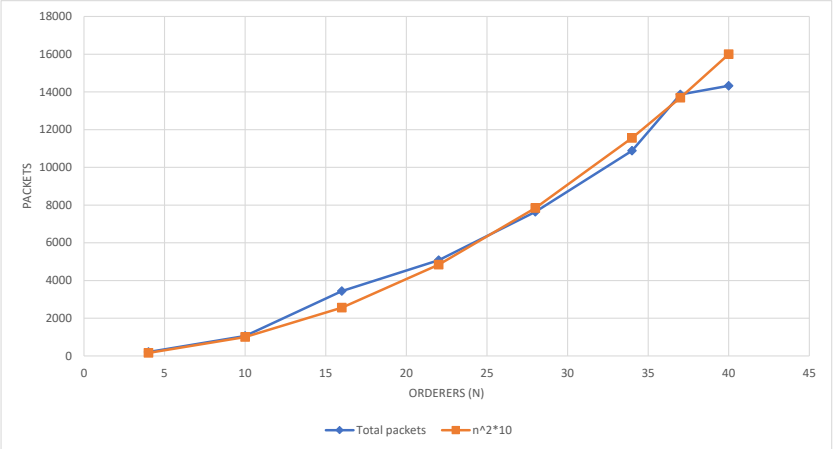


**Figure 6.1:** Total packets between all orderers based on amount of orderers, when the consensus algorithm agrees on the ordering of a new block. $n^2 \cdot c$, $c = 10$ for reference.

### 6.2.3 Evaluation

Plotting the result against a $n^2 \cdot c$ reference, see Fig. 6.1, shows that the network traffic required by the PoC indeed results in $\mathcal{O}(n^2)$ packets. The only outlier is at 40 orderers, but this is probably caused by the setup used for the PoC.

That the network traffic required by the PoC scales similarly to the BFT-algorithms often used in permissioned blockchains makes it possible to draw conclusions about the PoC's scalability from research papers addressing the scaling

| Versions | Response time [ms] |
|---|---|
| 0 | 53.7 |
| 10 | 56.1 |
| 50 | 64.1 |
| 100 | 78.4 |
| 200 | 104 |
| 500 | 184 |
| 1,000 | 316 |
| 1,500 | 420 |
| 2,000 | 548 |
| 3,000 | 770 |

**Table 6.2:** Response time against number of versions per program

of BFT consensus algorithms and especially other research on Fabric's scalability.

## 6.3 Response time

One of the questions in Section 1.2 is "What can be said about scalability of the design, both in terms of users and software? Is it worth it compared to a central alternative?". In order to answer this, the response time of the chaincode was tested with an increasing number of versions per program. The chaincode in the PoC iterates through all versions of a particular program upon a query, and is identified as a potential bottleneck of the implementation.

### 6.3.1 Method

Programs were entered into the blockchain, with values such that $0 \leq x \leq 2999, x \in \mathbb{Z}$ versions existed for any given program. The programs had the following appearance: `Hash: h{0-2999}, Name: n, Version: v{0-2999}, Certificate: c, Owner: o{0-2999}, Points: 1`.

The response time was measured with an increasing number of versions per program. Each query was performed 100 times and the resulting time is the best response time from those queries.

```
#!/bin/bash
python -m timeit -n 100 -r 3 \\
"__import__('os').system('./build/bin/peer chaincode query \\
-C mychannel -n vote -v 2.6 -c \'{\"Args\": \\
[\"queryOnEverything\", \"h1\", \"n\", \"v1\", \"c\"]}\'')"
```

### 6.3.2 Results

The final result is illustrated in Fig. 6.2, or Table 6.2 for more detailed information.

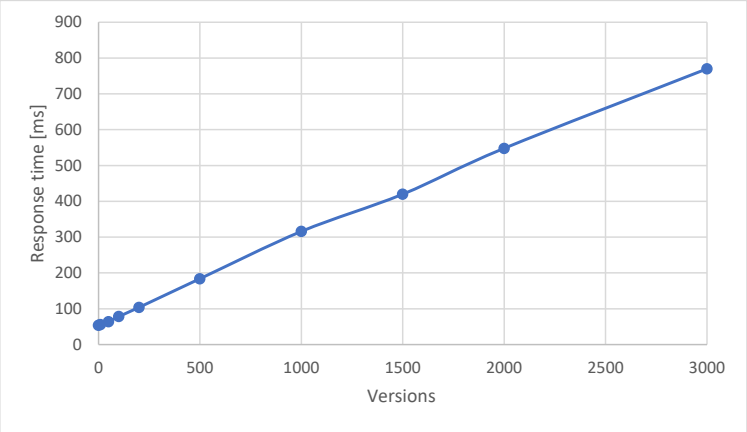


**Figure 6.2:** Response time against number of versions per program

### 6.3.3 Evaluation

A clear linear pattern can be seen in Fig. 6.2, and even though the test went up to 3,000 versions per program, the best response times never went above one second.

Calculating the linear equation for the graph:

$$k = \frac{770 - 0}{3000 - 0} \approx \frac{1}{4}$$
$$m = 53.7$$
$$\Rightarrow y = \frac{x}{4} + 53.7$$

Nielsen wrote that a one-second delay allows users to "keep their flow of thought" [48]. Assuming one second is the maximal delay acceptable by users for applications such as the PoC:

$$\frac{x}{4} + 53.7 = 1000 \Rightarrow x = 3785.2$$

It is able to handle up to 3,785 versions of a program before the response time becomes too slow.

## 6.4 Storage

In order to figure out how many programs/votes the PoC is able to handle. the following test was performed. Multiple programs were entered into the blockchain and a script measured how the size increased. The query response time was also measured.

### 6.4.1 Method

A setup of the PoC with six peers and four orderers utilizing BFT-SMaRt was started. The following script inserted 4,500 votes, and noted the blockchain size after each insert. The endorsement policy specified that every peer had to endorse the transaction.

```
#!/bin/bash

for i in `seq 1 4500`;
do
        ( cd "$GOPATH"/src/github.com/hyperledger/fabric/ ;
            ./build/bin/peerchaincode invoke -C mychannel -n vote -v 1.1 -c \\
            "{\"Args\":[\"createVote\", \"\", \"h1${i}\", \"n1${i}\", \"v${i}\", \\
            \"c${i}\", \"o${i}\", \"1\"]}"
        )
        sleep 2
        echo "$i $(du -b ~/tmp | grep mychannel | awk '{print $1;}')" >> disk.txt
done
```

### 6.4.2 Results

The resulting graph is found in Figure 6.3.

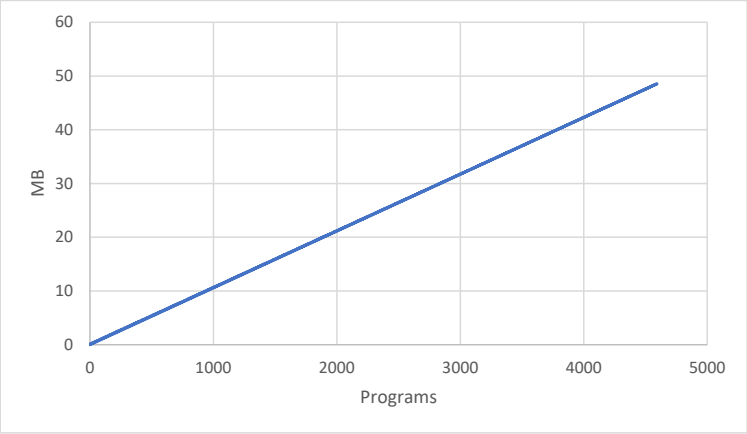


**Figure 6.3:** Disk usage against number of programs

### 6.4.3 Evaluation

The response time was not affected by the increasing program number, as was expected. It remained constant at roughly 55 ms for a query.

Each insert, however, resulted in an increase of 8-12 kB, the average being 10.6 kB, magnitudes larger than the actual data that is being inserted. A brief

investigation into the matter revealed that Fabric stores each participating endorser's certificate along with their signatures in the blockchain. In this test, the same seven certificates (700-800 bytes in size each) from the endorsement phase was stored together with the signatures (less than 100 bytes each), for every vote. This is not optimal for the PoC where each update is of a small nature. It could be circumvented by collecting votes into a batch and sending it to be endorsed after a certain time frame — increasing the ratio of votes per certificate in the blockchain. Or by optimizing Fabric itself, storing the certificates once in the blockchain and referring to them when they have been used to endorse a transaction, proposed in [49]. This would keep the integrity of each transaction, while minimizing the size. A third solution, also proposed in [49], is to utilize gzip[1] and compress the certificates. The issue reporter was able to reduce the size of certificates from 700 bytes to 400 bytes, and larger blocks to a third of their size.

[1] *gzip* is a free compression format and application.

Chapter 7

# Comparing blockchains to centralized databases

With blockchains explained in Section 2.5, how do they compare to centralized databases? What advantages and disadvantages come with the usage of blockchains? This chapter provides a comparison between centralized databases and blockchains with focus on aspects such as performance, properties, cost, smart contracts, and security. The performance of one public blockchain, the pioneering Bitcoin [11] that first introduced blockchains to the masses, and one permissioned blockchain that utilizes BFT, Fabric [12] is evaluated.

## 7.1 Properties

Blockchains are, in their basic form, databases. They, however, differ from traditional databases fundamentally and have special properties. The properties differ from one blockchain to another, but a common thread is consensus, integrity, and robustness.

Below are properties inherent to Fabric's design.

- Consensus
- Integrity
- Immutability
- BFT
- Accountability & transparency
- Crash tolerance & data loss prevention
- Robustness
- Smart contracts
- No need for a central unit

The main advantage of blockchains, and the reason the technology was first introduced, is the fact that there is often no need for a central unit, no trusted third party. The database is distributed over every entity in the network, and

the entities collectively manage it — there is no single point of failure in the system, and the replication offers data loss prevention. Kshetri [50] argues that the decentralization will play a huge role in the upcoming IoT boom. He reasons that by using smart contracts and similar techniques, devices can communicate more securely and without risk of a third party manipulating data. The trusted third party would be removed as a single point of failure, something he argues will be even more important with the exponential growth of network and server capacity that is expected to come with the IoT boom. Instead of communicating through centralized server farms, blockchains would be utilized.

Integrity comes naturally with the core concepts of a blockchain [4]. As was explained in Section 2.5, every new block with data contains the hash of the previous block, and small changes in any of the previous blocks would be noticeable in all blocks ahead of the modified one. The "real" version of a blockchain is not decided by a single third party whom everyone else must blindly trust, but instead the version that a majority of members uses is considered the real one. This also gives high immutability, since well-behaving members will follow and uphold an agreed upon protocol that would not allow removal or changes of blocks in the blockchain. Integrity in a database is not something that comes naturally with the design, but rather something that can be added on top.

## 7.2 Implementing in smart contracts

Fabric has the ability to run code "on the network" with smart contracts, or chaincode as they are called in Fabric. Implementing code in a smart contract comes with both advantages and disadvantages.

### 7.2.1 Advantages

Smart contracts allow entities on the network to verify the correctness of transactions, and that the agreement between relevant parties are fulfilled.

The core concepts behind smart contracts give a number of advantages:

- Autonomy — An agreement can be made without the need of third parties or lawyers. Potentially saving time and lowering costs.
- Decentralization — Gives several benefits; backups, trust, guaranteed outcomes, and transparency.
- Auto-sufficiency — Once the smart contract is available on a blockchain, the creator does not have to partake in the process anymore. The contract can be used and finish transactions by itself.

Smart contracts are entirely independent from the will of the parties once installed on the blockchain [51]. Contrary to traditional contracts, where parties are bound to the contract by legal obligations, smart contracts bind parties to the terms by locking assets until the terms have been met. In traditional contracts, a party could deem it to be more profitable to break the contract and agreement. Smart contracts leave fewer options to act in this way. This property also hinders a party from being able to reverse a transaction, known as *chargeback fraud* [51].

Sklaroff argues smart contracts may reduce the need of litigation, because of the immediate and irrevocable nature of smart contracts [52].

It could potentially lower costs of managing contracts, due to the reduced need of lawyers and accountants [53]. In fact, Cuccuru argues smart contracts remove the need of designing penalty clauses [51].

In a program where the main goal is to limit the need to trust a third party, having the code run inside the blockchain with the execution being the same for everyone might be beneficial. The code would then also be clearly visible, transparent for everyone to see, instead of hidden in binaries of an executable.

### 7.2.2 Disadvantages

Even though smart contracts hinder malicious users from cheating other parties involved in a transaction, it is still possible to exploit bugs present in the smart contract. One example is the DAO hack [31, 32, 33, 54].

There is no way to modify or terminate smart contracts [55]. Once a smart contract has been entered into a blockchain, there is no way to remove it or terminate its execution. Comparatively, legal contracts in real-life have ways for parties to nullify and revoke terms. Sklaroff listed properties that are available in legal contracts, but impossible to attain in today's smart contracts [52]. One difference of interest is the varying amount of interpretability in human languages compared to programming languages [51, 52]. Marino et al. [55] proposed ways to implement standards from real-life legal contracts in Ethereum's smart contracts.

Because smart contracts are implemented in programming languages they add another issue: most people do not possess the technical knowledge required to understand code, including lawyers. Smart contracts could be hidden behind a user interface, but according to Cuccuru, this only covers up the technicalities [51]. It does not eliminate the "semantic barriers".

Traditional contracts are often covered by a non-disclosure agreement, forbidding the parties from disclosing information to the outside world [56]. Smart contracts on the other hand are, for the most part, public. Fabric solves this issue by the concept of channels. Their documentation describes a channel as: "a private 'subnet' of communication between two or more specific network members, for the purpose of conducting private and confidential transactions" [57]. Every member in a channel has an identity provided by the membership service provider.

Multiple parties have to run the code during the endorsing phase, and every endorser has to sign its result from the execution of the chaincode. The signatures are collected by the proposing entity and sent to the orderers, who put the state change into the blockchain and broadcast the new state to every entity on the network, requiring involvement of several parties and adding extra steps to the code execution. Furthermore, smart contracts are executed in virtual environments, Docker containers in Fabric and Ethereum Virtual Machines in Etherum, making it yet more computationally expensive.

Alharby and Moorsel came to the conclusion, in their mapping of research relevant to smart contracts from a technical perspective, that insufficient research regarding performance and scaling has been done [58]. Another area with lacking research is actual applications of smart contracts. To quote them:

> "Although the concept of smart contract has gained a lot of attention, there are only a few applications developed by the literature."

Every area they identified with insufficient research is presented below.

- Performance and scaling
- Smart contracts outside of Ethereum
- Applications
- Criminal activities
- High quality peer-reviewed research

Most smart contract implementations execute contracts sequentially, regardless of whether the contracts use the same values or not. The same limitation is not found in Fabric, where peer $A$ does not necessarily run the same contract at the same time as peer $B$ [45]. Another difference is the use of endorsement policies in Fabric, where each smart contract is able to specify its own rule set for how many endorsers are required. As a result, not every peer has to execute every smart contract. All peers have to verify that the endorsement policy is upheld, and to maintain efficiency, there is an upper limit to the number of supported endorsement policies and their complexity [45].

Performance of smart contracts varies wildly between blockchains and is something that should be considered. As mentioned in Section 4.2.2, Pongnumkul et al. [7] performed tests on the time needed to execute smart contracts. In one of their tests, the execution time of the same smart contract in Fabric was only a tenth of Ethereum's. Pongnumkul et al. came to the conclusion that the difference likely comes from great differences in data access and management between Ethereum and Fabric.

## 7.3 Performance

Blockchains are inherently less efficient than their centralized alternatives. Scherer [9] wrote a short comparison between blockchains and centralized databases. In it he mentions three additional burdens that every node in the network has to bear, compared to a centralized database.

**Signature verification.** Before putting a transaction in the blockchain, multiple signatures are collected from different entities on the network. Asymmetric signatures are computationally expensive operations. Fabric uses Elliptic Curve Digital Signature Algorithm (ECDSA) as a default and, while it is a fast asymmetric algorithm, it still requires a considerable amount of computations. Running the following command on a computer with an Intel Core i7-4970:

```
$ openssl speed ecdsap256
```

Shows that the computer is able to perform 23,314 signs/s. Verifying is roughly half as fast at 10,137 verifies/s.

**Consensus mechanism.** The complexity for BFT is $\mathcal{O}(n^2)$, where $n$ is the number of nodes in the network. Croman et al. reported queue-congestion so bad that the network froze fully when adding more than 16 nodes during their scalability tests [1]. A centralized database has no need to achieve consensus with any other entity, as it is the sole authority when it comes to data.

**Redundancy.** Blockchains require multiple entities to process every single transaction. Once again, this is not needed in a centralized database. Sending packages over a network is time expensive in a computing context. This becomes a bottleneck for large networks. Bitcoin restricts their blocks to 1 MB, since larger blocks would result in higher latency when propagating out new blocks to the network.

### 7.3.1 Bitcoin

Bitcoin was the very first cryptocurrency to use blockchain technology. It is today by far the largest in market capitalization at a total value of over USD 150 billion [59] and, according to a review from 2016 by Yli-Huumo et al. [4], over 80% of their 41 extracted scientific papers related to blockchains had a primary focus on Bitcoin. Bitcoin is therefore included in this comparison, as a representative of public blockchains and to give contrast to the performance values of both databases and permissioned blockchains.

Bitcoin's performance issues are notoriously bad and, as mentioned in Section 2.5.1, a single Bitcoin transaction currently consumes as much energy as a household in USA does in 20 days [25]. Visa is a global payments technology company [60] with a centralized solution consisting of two data centers in North America [61] to handle most of the transactions through their payment system. A comparison on Digiconomist's web page [25] of the energy consumption of one Bitcoin transaction versus 100,000 Visa transactions can be seen in Fig. 7.1. Digiconomist's comparison shows an enormous difference between the two. Domingo, former CEO of a blockchain startup, argued in [62] that the comparison to Visa/-MasterCard is not a fair one, as Bitcoin is an entire transaction system in addition to a currency — Visa/MasterCard is a single piece of a transaction system. He argued that the comparison would have to include the power usage of every single bank in the world. Domingo estimated that this power usage exceeds Bitcoin's, 100 TWh versus 28.67 TWh. While this might be true, it does not consider the services banks facilitate beyond transactions.

Furthermore, there is the problem of transaction speed. Largely due to the risk of forks, mentioned in Section 2.5.1. Bitcoin is a payment system, but it is recommended to wait for total of six blocks before considering the transaction as completed. A transaction time of about an hour is not suitable for mainstream usage. Moreover, Bitcoin is estimated to be limited to $< 10$ tx/s [1]. Visa is capable of handling over 65,000 transactions per second according to their website [60].

Recent work has presented solutions to Bitcoins scalability issues, one example being Poon and Dryja's work on using micropayment channels on top of the Bitcoin blockchain [10], where an unlimited amount of transactions can be carried out

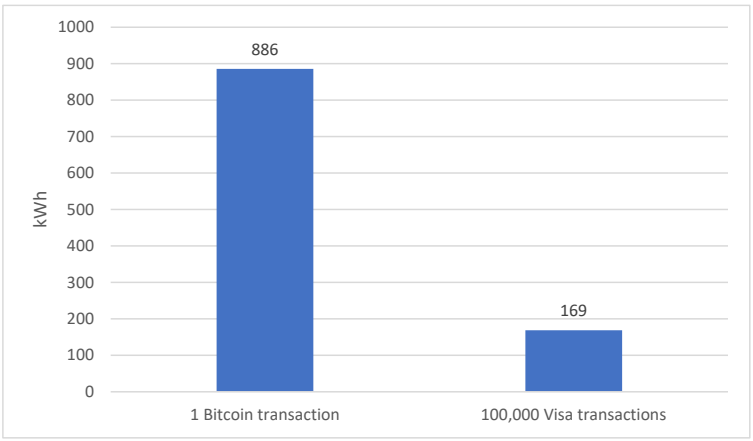


**Figure 7.1:** Comparing the energy consumption of one Bitcoin transaction to 100,000 Visa transactions.

without much effect on the blockchain. Poon and Dryja stated that the solution, named "Lightning Network", would offer a way for Bitcoin to scale to billions without risk of centralization.

### 7.3.2 BFT in Bitcoin

Croman et al. [1] proposed various other ways to improve the scaling of Bitcoin. One of those were the usage of consensus algorithms, such as the BFT replication protocol. They measured how BFT scales with number of nodes and the size of each batch, and the results is shown in Table 7.1. This comparison is also of relevance to the PoC developed for this thesis, since Fabric is going to use PBFT in the future. Croman et al. noted that "Scaling to hundreds of nodes, however, would greatly degrade the performance of the system", and this can be seen in Table 7.1 as well. Every increase in the number of nodes greatly decreases the number of transactions that the system is able to handle. From 113,000 tx/s at four nodes to 2,400 tx/s at 64 nodes.

### 7.3.3 Hyperledger Fabric

Fabric is included in this comparison as it is the blockchain used in the developed PoC and its scalability is therefore of great interest. It was also argued in Section 4.2 that Fabric is both mature and scales well compared to other options. Fabric utilizes BFT algorithms for its consensus, something that is common for permissioned blockchains. It is therefore a good representative for permissioned

| # of Nodes | Batch size | Latency (s) | Throughput (k tx/s) |
|---|---|---|---|
| 4 | 32,768 | 0.29 | 113 |
| 8 | 8,192 | 0.58 | 14.0 |
| 8 | 32,768 | 1.48 | 22.2 |
| 16 | 8,192 | 0.69 | 11.9 |
| 16 | 16,384 | 1.04 | 15.8 |
| 32 | 2,048 | 0.48 | 4.3 |
| 32 | 8,192 | 0.93 | 8.8 |
| 64 | 2,048 | 0.82 | 2.4 |
| 64 | 8,192 | 1.79 | 4.5 |

**Table 7.1:** Results from Croman et al [1].

| Orderers | Throughput, tx/s |
|---|---|
| 4 | ∼4,000 |
| 7 | ∼3,000 |
| 10 | ∼2,000 |

**Table 7.2:** Throughput of BFT-SMaRt with an increasing amount of orderers. Configured to use transactions of a total size of 4,000 bytes, 10 transactions per block, and the amount of receiving peers is the same as the amount of orderers.

blockchains.

IBM's working paper on Fabric states that they were able to achieve more than 3,500 tx/s [63]. Their tests were performed using three orderers with Kafka, and five endorsing peers. A recent study by Vukolić et al. used the modularity of Fabric to add support for BFT-SMaRt and measured the throughput [64]. The study only tested throughput with up to 10 orderers, but a clear decline could be observed, see Table 7.2. The sample size might be too small to draw conclusions, but the throughput would quickly become insufficient for most use cases if the trend were to continue.

Dinh et al. [41] compared various blockchains, including Fabric v0.6, with a centralized in-memory database, called H-Store. They used two benchmarking tools, "Yahoo! Cloud System Benchmark" (YCSB) and "SmallBank". YCSB contains simple transactions, while SmallBank benchmarks more complex ones. The results are illustrated in Fig. 7.2. Dinh et al. concluded the gap in throughput remains too vast for blockchains to be truly disruptive as an alternative for systems currently reliant on databases. They also stated that blockchains have a great deal to learn from traditional databases about high-performance data processing.

Using BFT means you have to put trust in two-thirds of the orderer nodes. This is a somewhat stricter assumption than Bitcoin, where there is no need to trust any particular entity. The Bitcoin network is functional as long as over 50%

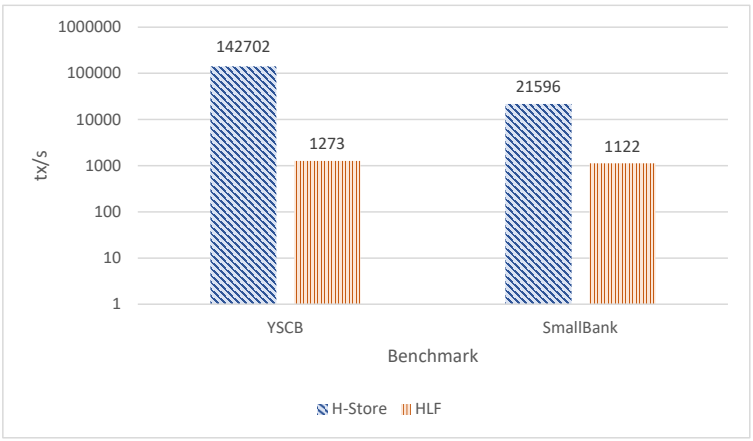


**Figure 7.2:** Comparison between H-Store and Fabric. Fabric is using 8 servers and 8 concurrent clients.

of the network complies with the rules [65].

### 7.3.4 Latency

The latency of a transaction is the difference in time from when the transaction was first deployed on the blockchain and the time the transaction is immutably appended to the blockchain.

In Dinh et al.'s tests [41], latency was measured. Fabric v0.6 had a latency of around 50 seconds with eight orderers during peak throughput in their test using Smallbank. This can be compared with Ethereum's 114 seconds and Parity's[1] 7 seconds with similar setups.

Vukolić et al. added support for two different BFT implementations to Fabric v1.1, WHEAT [67] and BFT-SMaRt. The latency of the two was measured, and BFT-SMaRt had almost twice the latency in all of their tests [64]. The tests were performed with four orderers placed across the world and BFT-SMaRt had close to 600 ms of latency. This result is similar to Dinh et al.'s tests [41] with four orderers and four clients, where the measured latency was 900 ms.

Dinh et al. measured how latency was affected by an increasing number of nodes in Fabric [8]. They measured eight to 32 nodes, with eight clients sending requests. The resulting latencies landed between 20 and 60 seconds respectively. The increase in nodes seems to result in a linear increase in latency, see Fig. 7.3. In another test they had a fixed amount of nodes and clients, and instead varied

[1] *Parity* is an Ethereum derivative that uses consensus based on round-robin [66].

the transaction rate. They found a large increase in latency when nearing 200 tx/s, going from just a few seconds up to the aforementioned 20 s. After this initial increase, the latency is somewhat stable at roughly 20 s. These tests were performed on Fabric v0.6.

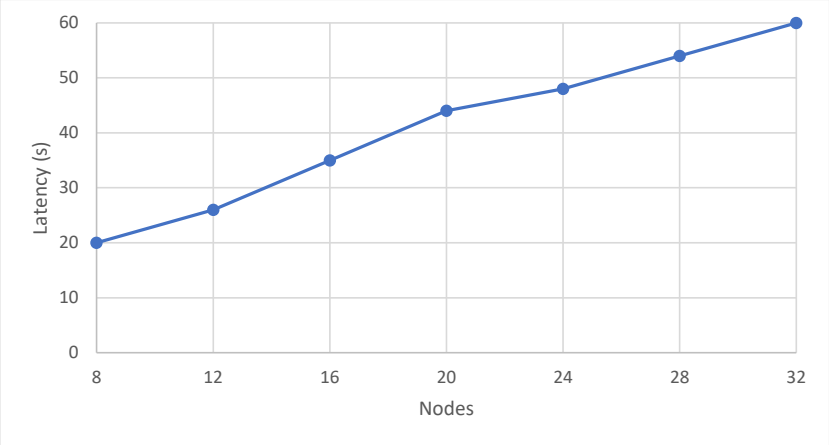


**Figure 7.3:** Latency in Fabric v0.6 with 8 clients and an increasing number of nodes.

In contrast, End Point compared various NoSQL databases using the YSCB benchmark tool [68]. Their tests were performed on databases in cluster sizes ranging from one to 32. A test with read-modify-write operations with one node gave the best result for Cassandra with 8,579 tx/s, and the worst for HBase with 325 tx/s. At 32 nodes the results had gone up to 201,440 tx/s for Cassandra, and the worst result for MongoDB with 2,264 tx/s.

They measured latency as well, with an increasing amount of database nodes. Again, best and worst at one node was 29,895 $\mu$s for Cassandra, and 229,391 $\mu$s for HBase. Finally, at 32 nodes the results were 40,358 $\mu$s for Cassandra, and 2,606,085 $\mu$s for MongoDB.

## 7.4 Cost

Apart from being a database on which values are stored, blockchains have protocols to achieve consensus and distribution over a network. These layers add complexity. Consequently, the costs that come with blockchains most likely reflect this. From the experience on this thesis, it took multiple days before the authors

were comfortable enough to start working on the PoC. After that, the development itself was slow. This might be attributed to Fabric's complexity, even when comparing it to other blockchain designs. Fabric is intended to be adaptable to suit large number of use cases with its modular architecture [13], and the flexibility comes with a noticeably more complex setup. Fabric also has multiple types of nodes that come in the shape of clients, peers, anchor peers, endorsing peers, and orderers. When using Kafka there are also Kafka brokers, and "ZooKeepers". The chaincode in Fabric is executed in Docker containers, adding yet another layer of complexity to the design.

Steve McConell argues, in a chapter named "Effects of Project Size on Errors" in his book Code Complete [69], that larger projects have a higher frequency of errors per thousand lines of code than smaller ones, and this probably holds true when developing on a complex foundation such as a blockchain. The development time is therefore longer, and maintenance becomes more expensive.

A blockchain is stored on all entities, compared to a central database where the database is only stored on the third party. The blockchain in Ethereum is over 300 GB and growing [70]. Yli-Huumo et al. estimated that the blockchain used in Bitcoin could grow 214 PB per year if it ever were to reach the same level of throughput as Visa [4]. Even if a private blockchain is used, considerations still have to be made about what type of data to store in the blockchain, since all data will have to be transferred and stored on all other nodes as well. If high transaction speed is required, slow HDDs might not be an option and quicker SSDs might be required, on all participants.

## 7.5 Security

The centralized database model is mature, especially relational databases. They have been in use since the 70's. Blockchains on the other hand were first thought of in 1991 by Haber and Stornetta [71]. There was little interest until Nakamoto's paper put them to use in a cryptocurrency in 2008 [11]. Much of the research performed on blockchains are related to the security of the technology [4].

Surveys by Li et al. [72] and Conti et al. [73] listed some attacks on Bitcoin that were identified at the time of writing in 2017, a summary is found in Table 7.3.

Of course, individual database projects might be less mature than individual blockchain projects, but relational databases have been in use for decades longer than blockchains. A review of research papers on blockchain by Yli-Huumo et al. concluded that a large part, 14 out of 41 reviewed papers, were related to challenges and limitations in blockchain and Bitcoin security [4].

Another aspect worth considering is who sees the data. In a database, only the third party has to see and verify the data other than the two parties involved in a transaction. In a blockchain however, all other participants see some data and are supposed to be able to verify the correctness of the transaction. Privacy in a public blockchain is a concern, and it is not always wanted that all parties in a permissioned blockchain should see all data in cleartext. The same review by Yli-Huumo et al. found that 10 out of 41 reviewed papers related to blockchains were about privacy issues and countermeasures to increase anonymity [4]. Ways around

| **Security flaw** | **Description** |
|---|---|
| 51% attack | A miner possessing > 50% of the total mining power is able to manipulate the chain however it pleases. |
| Double-spending | Spending the same coin twice. |
| Selfish mining attack | A miner working on a private fork, only disclosing blocks when the public fork is about to catch up. This leads to an increased revenue for the selfish miner, and wasted work by nodes mining on the public fork. |
| BGP hijacking attack | Rerouting traffic going to miners. This can be done to split the mining network, or to delay information going to a certain miner. |
| Liveness attack | Delaying the confirmation time of a transaction for as long as possible. |
| Balance attack | Delay network communication between groups with equal mining power. The attack results in the ability to rewrite blocks with a high probability. |
| Sybil attack | Generating numerous identities and using them to gain influence. |

**Table 7.3:** Security flaws in Bitcoin.

this exist using encryption, but it is still something that has to be considered during development.

## 7.6 Blockchains & General Data Protection Regulation

In 2016, the Council of European Union passed *General Data Protection Regulation* (GDPR) [74]. GDPR is designed to give EU citizens more control over personal data collected by companies. Any company storing personal data from EU citizens has to comply with GDPR, no matter the location of their headquarters. Among the key changes is the "Data subject right" commonly called *Right to be Forgotten* [75]. An EU citizen is able to demand that the company removes all relevant personal data to that particular citizen. This is a problem for blockchains, which do not possess the ability to remove data. GDPR was designed with the model of centralized entities controlling the data in mind, but blockchains are often entirely decentralized with no single entity able to modify previously inserted data. It is still not clear how blockchains will be regulated by GDPR.

Berberich and Steiner wrote an article in which they discussed the difficulties of having blockchains comply with GDPR [76]. Simply encrypting the data does not take it outside the scope of GDPR, and is in fact even required if the data warrants it. The data would have to be entirely anonymous to not be covered by GDPR. They noted that the wording of GDPR might take blockchains' nature of persistence into account, as previous data in the blockchain is required to process new blocks.

## 7.7 Distributed database

A distributed database is an alternative to a single, centralized, database. An illustration of the underlying concepts of centralized/distributed databases and blockchains can be seen in Fig. 7.4. Blockchains are a form of distributed databases, which means that motivating the choice of one over the other basically comes down to the properties they possess. A need for smart contracts, for instance, makes the choice trivial. Distributed databases might be able to achieve crash tolerance, but BFT is commonly not seen in database systems outside of blockchains. There are proposed protocols for middleware which would allow database systems to share snapshots[2] with BFT [77].

Distributed databases are able to handle more transactions per second than centralized solutions [68, 78, 79]. Distributed are more complex however, and as argued in Section 7.4, this leads to higher development and maintenance costs. Consequently, distributed databases are used when centralized solutions are not able to cope with the amount of transactions per second required, or when the data needs to be accessible even if one database becomes unreachable.

[2] *Snapshot* refers to the state of a system at a particular point in time

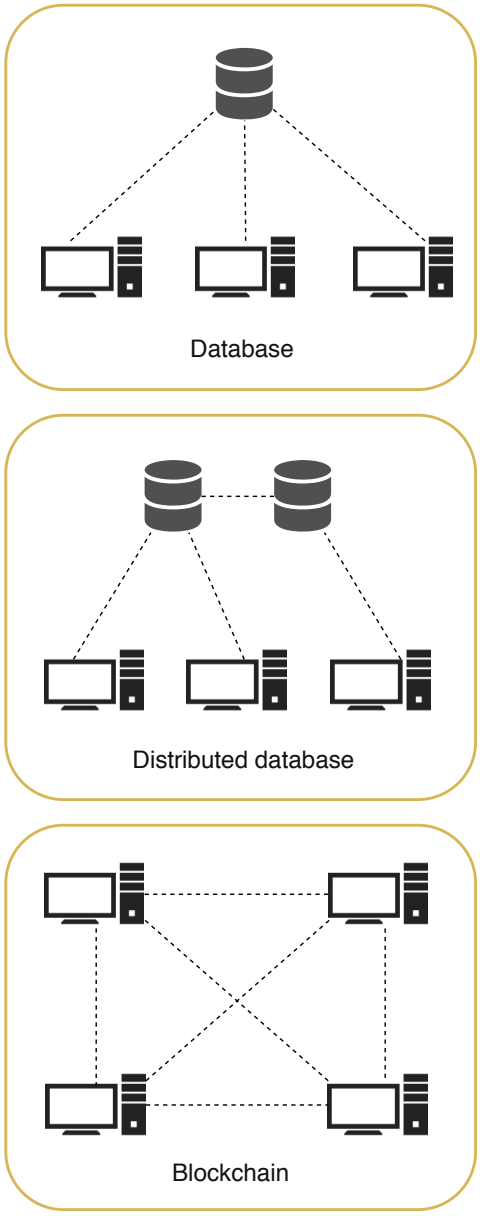


**Figure 7.4:** Core concepts of databases and blockchains.

## 7.8 Summarizing table

A table summarizing some differentiating properties between blockchain and databases is shown in Table 7.4.

The table is meant to be used as a guidance when considering blockchains for a project. The columns reflect the sections presented throughout Chapter 7 and

the references to literature exist in these sections.

Croman et al.'s work on the scalability of BFT consensus, first cited in Section 7.3.2, resulted in 113,000 tx/s at four nodes [1]. This number was expected to be closer to the numbers from [8, 41, 64] for BFT consensus in Fabric. There might be differences in the implementations that justify some discrepancy, but most of the difference can probably be explained by differing test setups. Croman et al.'s tests used a very large batch size. Looking at their result in Table 7.1, changing the batch size from 32,768 to 8,192 at eight nodes lowers the throughput from 22,200 tx/s to 14,000 tx/s. 8,192 is still a very large batch size, and Vukolić et al. performed tests with varying batch sizes in Fabric [64]. Tests with batch sizes of 10 and 100 were performed, still much smaller than Croman et al.'s 8,192. With a batch size of 100 and a transaction size as small as 20 bytes, Vukolić et al. were able to reach around 90,000 tx/s with seven nodes. Vukolić et al. state that a more realistic size of a transaction would be around 1,000 bytes, and the transaction packets from the PoC developed in conjunction with this thesis are about 6,000 bytes in size when used with four orderers. Vukolić et al.'s result with a size of 4,000 bytes per transaction shows a more humble throughput of around 4,000 tx/s for four nodes. Croman et al.'s result, even though interesting, has therefore been excluded from the table. Instead [8, 41, 64] have been used.

The value of 4,000 tx/s at four nodes comes from Vukolić et al.'s test [64] with an implementation of BFT-SMaRt in Fabric v1.1. The upper value at eight nodes is based on Vukolić et al.'s test of seven nodes, which reached a throughput of 3,000 tx/s. The lower value is the result from Dinh et al.'s test with eight nodes in Fabric v0.6 (PBFT instead of BFT-SMaRt). The upper-bound given at 16 nodes is based on the result from Dinh et al.'s test with 16 nodes, which reached 1,000 tx/s. This can likely be seen as an upper bound after looking at the potential trend in Vukolić et al.'s test, where the throughput was already down to 1,000 tx/s at 10 nodes. Adding six nodes in Vukolić et al.'s test dropped the throughput to one-third. If the same were to happen between 10 and 16 nodes, the resulting throughput would be as low as $\sim$ 300 tx/s. Similarly, Dinh et al.'s result at 32 nodes of 1,000 tx/s is used as an upper bound for 32 nodes in the table.

For traditional databases, the values from End Point's benchmark was used [68], as they tested cluster sizes from one to 32 nodes with consistent setups.

There is lacking research on the performance of Fabric and BFT-consensus at higher number of nodes. The tests performed with lower amount of nodes [1, 8, 41, 64], as well as the theory behind BFT where more nodes require more participants to be involved and more messages to be sent before consensus is reached, see Section 2.5.2, indicates that the throughput would at least continue to decrease with more nodes. The values at 100 and 1,000 nodes for both the throughput and latency are estimates based on the trends visible from the smaller tests, most notably the test in [8]. The drop in throughput is roughly 50 tx/s for every four added nodes. At 1,000 nodes the throughput would be close to 0, if such a network is even possible at all. The authors were not able to find any network running more than 64 BFT consensus participants.

An alternate table made by Bano et al. [80] contains more blockchains projects and their respective properties, but has no information about how the number of nodes affects performance.

| | Throughput [tx/s] | | Latency [s] | Crash Tol. | Security | TTP* | Cost | Additional |
|---|---|---|---|---|---|---|---|---|
| **Centralized** | 8,579 | | $2.99 \cdot 10^{-2}$ | No | Mature<br>Tested | Yes | Low | Simplest<br>Single point of failure |
| **Distributed** | Nodes | tx/s | | | | | | |
| | 4 | 32,005 | $3.20 \cdot 10^{-2}$ | | | | | |
| | 8 | 65,822 | $3.12 \cdot 10^{-2}$ | Yes | Mature<br>Tested | Yes | Medium<br>High | Scalable |
| | 16 | 117,375 | $3.52 \cdot 10^{-2}$ | | | | | |
| | 32 | 201,440 | $4.04 \cdot 10^{-2}$ | | | | | |
| **Bitcoin** | 7 | | 3,600 | Yes | Immature<br>Area of research | No | High | High energy cost<br>Large chain<br>Integrity |
| **Fabric** | Nodes | tx/s | | | | | | |
| | 4 | 4,000 | 0.9 | | | | | |
| | 8 | 1,250-3,000 | 20 | | | | | |
| | 16 | 1,200 | 35 | Yes | Immature | No | High | Smart contracts<br>Integrity |
| | 32 | 1,000 | 52 | | | | | |
| | 100 | 100** | *140*** | | | | | |
| | 1,000 | *5*** | *1,200*** | | | | | |

* *Trusted Third Party.*

** The values are estimates.

**Table 7.4:** Summary of the comparison chapter.

## 7.9 When to use blockchain

Previous work has created decision trees to be used in the decision process. The authors build upon Johanssons' [6], Peck's [81], and Maull et al.'s [82] work to create a new decision tree. The decision tree builds upon Johansson's as a basis and includes questions from Maull et al.'s. Finally, the resulting tree was cross-checked with Peck's.

The comparison provided throughout Chapter 7 and the summarizing table created, see Table 7.4, puts much focus on Bitcoin and Fabric. Other options to both Bitcoin and Fabric exist. Ethereum is the blockchain used for the second largest cryptocurrency by market capitalization [59], and is a public blockchain that, according to [83], is soft capped at a throughput of 10 tx/s, similar to Bitcoins 7 tx/s. Sawtooth was mentioned in Section 4.2 as an alternative to Fabric, and supports PoET "designed to be a production-grade protocol capable of supporting large network populations that include Byzantine actors" [84]. The numbers in the decision tree are specific to Fabric and Bitcoin but the decision tree should still prove to be useful even if similar projects to Bitcoin and Fabric where considered instead.

Fabric has been able to achieve at the very least 4,000 tx/s, with subsecond latency. It is able to keep up with transactions at a low number of nodes. But as the number of nodes increases throughput is dramatically decreased [63]. No research on Fabric using more than 32 orderers have been found. With an increasing amount of nodes come an increasing transaction latency. The delay reaches 1 minute soon after 32 orderers. Applications using Fabric should function even if the latency were to go above 1 minute.

There is no point in utilizing a blockchain if the application does not require shared write access in the database. A traditional database is able to provide the necessary functionality for such an application

Blockchains remove the need for trusted third parties. There is thus no place for blockchains in a third party requiring system. Even if no third party is required, it is more convenient to utilize one if it is possible to trust it with the data.

Bitcoin's maximal throughput is 7 tx/s, with a latency of 60 minutes before a transaction is considered to be in the blockchain. An application with performance requirements above this should remove Bitcoin from its considerations.

The authors make the same argument as Johansson: if there is a direct link between participation and real-world entities then permissioned blockchains are the most suitable. Use permissionless for anything else. This thesis' decision tree has "Bitcoin" and "Fabric" as examples for "permissionless" and "permissioned" due to the focus that has been placed on these two in Chapter 7. The decision tree can be seen in Fig. 7.5.

An important question to ask before choosing a blockchain structure over a traditional database is whether the integrity and BFT properties are essential. If these properties are not of great importance, then these advantages should be weighted against the added complexity of the system, the longer transaction times, lower throughput and the added costs.

Costs are not taken into consideration in the tree, but the cost criterion is more complex than is suitable for this tree. The authors suggest calculating a

development cost forecast, for both blockchains and traditional databases. The forecast should include the increased development difficulties and maintenance costs, as well as the cost of storage. This is then used as the basis for the final decision if the project should include blockchains or not.

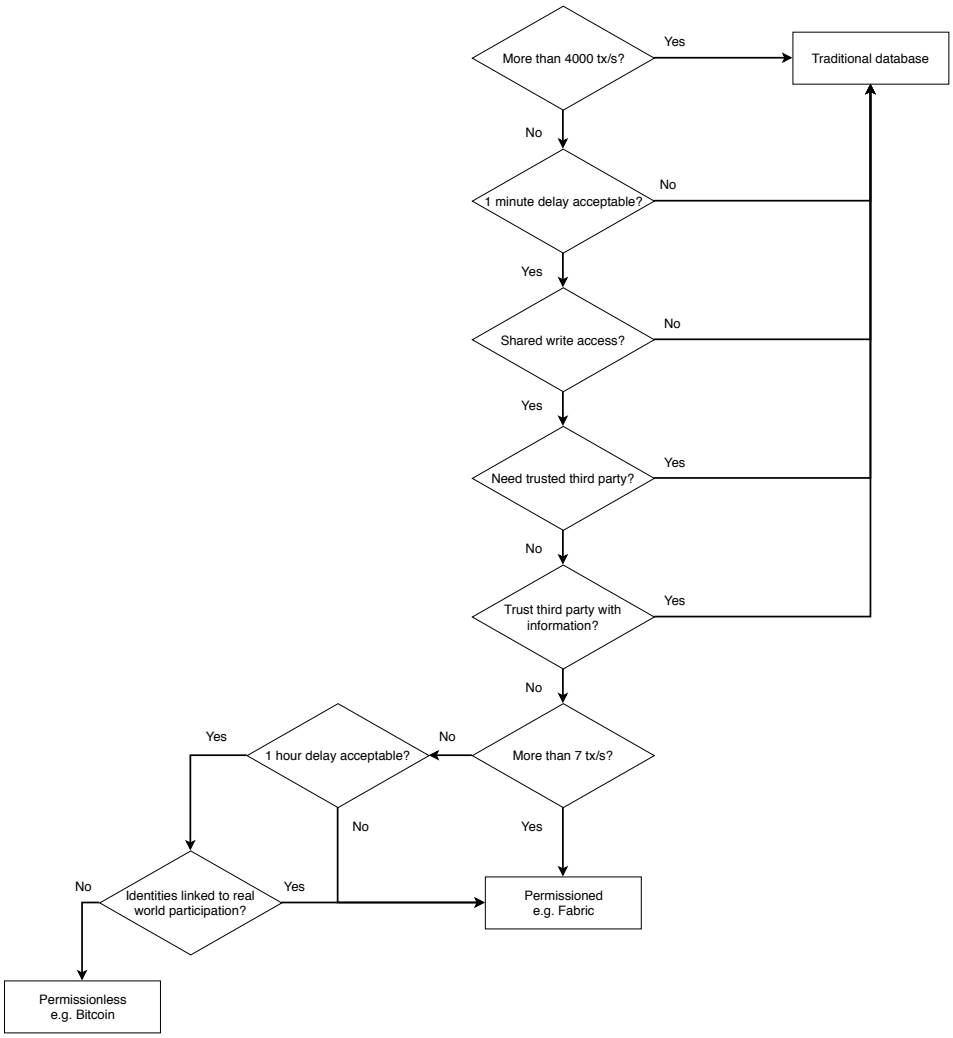


**Figure 7.5:** Decision tree for blockchain usage.

# Chapter 8
# Discussion

## 8.1 What can be said about scalability of the design, both in terms of users and software? Is it worth it compared to a central alternative?

The question is divided into three parts. First, throughput and latency is discussed. Then, amount of programs and storage requirements, and lastly, if it would be worth it compared to a central alternative.

### 8.1.1 Network traffic

The PoC will require three phases of approval before a new vote is stored:

- Endorsement. The client will broadcast the transaction proposal to all peers and wait for a response from $\frac{n}{3}+1$ of them. These peers will have to execute the chaincode. This will require sending $\mathcal{O}(n)$ packets out from the client and $\mathcal{O}(n)$ packets back to the client.
- The orderers will agree on the ordering of events. Theory and the test from Section 6.2 shows that with BFT-SMaRt this will require $\mathcal{O}(n^2)$ packets.
- The orderers will broadcast the new block to all peers and the client will wait for confirmation from $\frac{n}{3}+1$ that the block has been immutably appended to the blockchain. This will also require sending $\mathcal{O}(n)$ packets out from the client and $\mathcal{O}(n)$ packets back.

With big $\mathcal{O}$ notation, this would result in $2\mathcal{O}(n) + \mathcal{O}(n^2) + 2\mathcal{O}(n) = \mathcal{O}(n^2)$ amount of packets. Even with the somewhat clashing design choices of Fabric separating endorsers and orderers, and the strict trust assumption of the PoC, the complexity is still decided by the BFT algorithm. The network traffic is a potential bottleneck of the design and one of the biggest differences compared to if the design had utilized a centralized database instead. Seeing that the implementation is as good as it can be with BFT consensus makes it easier to argue about its scalability.

Tests with Fabric utilizing BFT algorithms for the ordering seems to rarely be performed with more nodes than the lower double digits. One study by Dinh et al. [8] used Fabric v0.6 with PBFT to perform tests with up to 16 orderers, and then had difficulties with the network freezing fully. This is interesting since the

test performed in Section 7.3.3 also froze several times when using 40 nodes. This is also the reason why tests were not performed with more than 40 orderers. Dinh et al. concluded that their freezing was caused by queue-congestion. A cause for the test in Section 6.2 freezing was searched for, but nothing conclusive was found.

Another study [23] from 2016 states the area is not well explored and that tests have only been performed with fewer than 20 orderers. In Section 7.3.3, the result from Vukolić et al.'s report [64] on the scaling of Fabric v1.1 with BFT was discussed and it was observed that in the interval four to 10 total orderers, every three more orderers added resulted in a drop of 1,000 tx/s. If these tests were performed with more orderers, the transaction speed would most likely decrease to triple or maybe even double digits. This is worrisome, since for most applications, more users would result in a need of more transactions. This holds true for the PoC as well, but luckily only a handful of transactions should be required by every user each day. If each user votes 10 times per day, which seems overly high after an initialization phase, 1,000 users would result in $\frac{1{,}000 \cdot 10}{24 \cdot 3{,}600} \approx 0.1\,\mathrm{tx/s}$. On the other hand, the same amount of users would result in approximately $1,000^2 = 1,000,000$ packets being sent per placed vote. It seems likely the bottleneck will not be the reduced transaction speed, but rather the huge amount of packets being sent and the large amount of connections (thousands) being opened. Furthermore, with each signature comes a certificate as well which results in a packet size of 1,000 bytes per signature. Just the packets with all endorsements sent from the client to the orderers would be $(\frac{n}{3}+1) \cdot 1,000$ bytes in size (334 kB at 1,000 users).

The latency at 1,000 users would most likely be much larger than the numbers of 20 - 60 seconds mentioned in Section 7.3.4, and a high latency would make the program less effective. The result from Dinh et al. [8] mentioned in Section 7.3.4 indicates a linear increase in latency between eight and 32 users, and if that trend were to continue up to 1,000 users, the delay would be about 1,200 seconds or 20 minutes. 20 minutes of delay is a long time, and the faster the ledger can be updated the better decisions can be made for other users, but 20 minutes could prove to be acceptable. It might also be unfair to assume the delay will scale this way since Dinh et al.'s tests also had a high amount of transactions per second (about 1,000 tx/s). The lower amount of transactions in the PoC might result in a lower delay as well, since there is a lower risk of congestion. In Dinh et al.'s results for example, there was a large increase in latency when nearing 200 tx/s, going from just a few seconds up to 20 seconds [8].

Overall, looking at the network traffic, a thousand users seems to be the absolute ceiling for the PoC as long as the trust assumption is not changed. It would not be too surprising if the network were to have problems already at 100 users, considering the lack of research with so many orderers, what research with fewer orderers, and what the authors' experience seems to indicate. As, for example, Croman et al. noted in their research [1] on BFT in Bitcoin; "Scaling to hundreds of nodes, however, would greatly degrade the performance of the system". 100 users would also result in a lower latency of around 140 seconds, again looking at the trend from Dinh et al.'s tests [8].

If a trusted third party was to be introduced to manage the data instead of using a blockchain, the scaling would be logarithmically better. Using the numbers for H-Store mentioned in Section 7.3.3 of 21,596 tx/s and the estimated low amount

of 10 tx/day from each user would mean the H-Store database would be capable of handling 186,589,440 users, see Equation 8.1 and 8.2.

$$\frac{10\,\text{tx/day/user}}{24\,\text{h/day} \cdot 3,600\,\text{s/h}} = 1.2 \cdot 10^{-4}\,\text{tx/s/user} \tag{8.1}$$

$$\frac{21,596\,\text{tx/s}}{1.2 \cdot 10^{-4}\,\text{tx/s/user}} = 185,589,440\,\text{users} \tag{8.2}$$

However, when querying an executable using the implemented PoC with Fabric, only the local peer is queried (since a query does not result in any changes to the blockchain). If a central database were used, all queries would also have to be considered when counting transactions per user. If each executable is queried each time it is started, a better estimate would probably be around one new query (or vote) every 10 minutes. This would be around 150 tx/day per user and, using 150 tx/day instead of 10 tx/day in Equation 8.1 and 8.2, would mean that H-store still would be able to handle around 12,439,296 users. If sharding was used, this number would increase.

A solution utilizing a centralized database would also result in sub-second latency, as mentioned in Section 7.3.4.

### 8.1.2 Amount of programs

Since Fabric uses a database for retrieval of states, a query of a program is not slowed down by the total amount of programs in the blockchain, which was confirmed by the test in Section 6.4. However, the smart contract developed iterates through all versions of a program during a query. Amount of versions was therefore identified as a limiting factor when looking at the scalability. For a query, only the local copy of the blockchain is used, since a query does not create any changes to the blockchain. The test in Section 6.3 showed that the response time of the PoC scales linearly with the number of versions of a program. The response time seems to stay under 1 second for up to almost 4,000 versions of a single program. It seems very unlikely that this will ever be a bottleneck.

Disk usage was however identified as a problem. The data of each vote contains several SHA256 hashes and strings, which would be around a couple of hundred bytes in total for each vote. This would not be a problem, though unfortunately, as mentioned in Section 6.4, in addition to the data, the signatures and certificates of all endorsers are also saved on the blockchain. The certificates and signatures are stored so that the transaction can be verified at a later date, as well as making it possible for the blockchain to be recreated by nodes that join the network later or for other reasons need to recreate the blockchain and the stateDB. This offers strong integrity and allows the blockchain to be replicated in a trusted way, but comes with the very obvious cost of requiring additional disk space.

As mentioned in Section 8.1.1, each certificate together with corresponding signature is around 1,000 bytes in size. With 1,000 users, a vote would need $\frac{1000}{3} + 1 = 334$ endorsers. This would mean 334 kB of storage for each vote. Increasing users would result in a $\mathcal{O}(n^2)$ increase in storage need. Using the same assumption as in Section 8.1.1, 10 votes per day for each user would result in $10 \cdot 334\,\text{kB} \cdot 1000 = 3,340,000\,\text{kB}$ or 3,34 GB increase in blockchain size per

| Users | Endorsers | MB/day | GB/year |
|---|---|---|---|
| 4 | 2 | 0.08 | 0.03 |
| 6 | 3 | 0.18 | 0.07 |
| 10 | 4 | 0.4 | 0.15 |
| 20 | 8 | 1.6 | 0.58 |
| 30 | 11 | 3.3 | 1.20 |
| 40 | 14 | 5.6 | 2.04 |
| 50 | 18 | 9 | 3.29 |
| 75 | 26 | 19.5 | 7.12 |
| 100 | 34 | 34 | 12.4 |
| 1,000 | 334 | 3,340 | 1,220 |

**Table 8.1:** Increase of the blockchain size with an increasing amount of users. The size is calculated as endorsers · size of each certificate together with the signature (1 kB) · 10 votes per day per user · users.

day. Looking at Table 8.1, 100 users is more reasonable at 34 MB per day. This would still result in a yearly increase of the blockchain size of 12 GB. Fabric's web page mentions pruning[1] as a post-v1 feature [85], but it is not currently available. Pruning the blockchain would result in old transactions being removed, and even if the state-changes are kept from these transactions, it would not be possible to verify the changes again in the future. Even if this information loss is deemed acceptable for most users, the option to prune is still missing in Fabric v1.1. It would be hard to motivate more than 100 users with the current disk requirements this would result in.

A centralized database would not need an endorsement phase and the certificates would not be stored. Each vote would instead only be a couple of hundred bytes in size. 12 million users, as suggested in Section 8.1.1, would result in around $0.2\frac{\text{kB}}{\text{vote}} \cdot 10\frac{\text{votes}}{\text{day}} \cdot 12,000,000\,\text{users} = 24\,\text{GB/day}$. This is a very large number, but it would probably be possible to shrink it down using smart models in a relational database, and the data would only be stored on the trusted third party. The trusted third party would pose as a single point of failure, but each user would not be required to use several gigabytes of their own storage.

### 8.1.3 Is it worth it compared to a central alternative?

The question "Is it worth it compared to a central alternative?" is a bit subjective in nature. The developed PoC seems usable even with the aforementioned scalability problems. One hundred users and a delay of a couple of minutes is acceptable, and benefits of higher integrity, immutability and transparent/verifiable voting though smart contracts are great advantages that might outweigh the drawback

[1]The data in old blocks are removed to reduce the size of the blockchain. The block header, and therefore also the hash of the data, is kept, meaning that it is possible to show that the removed data was once part of the block.

of the required disk space. The program would work well as long as the users have somewhat similar systems, since whitelisting happens only when 34% of the users have voted positively on an executable. If, however, trust for a third party would already exist, the other benefits of the blockchain structure might not compensate for the added scalability of a centralized, or distributed, database where data from millions of users could be used instead.

It should be noted that Fabric's design changes mentioned in Section 4.2.4 have the potential to greatly increase Fabric's scalability in some use cases. The idea is that when for example five organizations have a collaboration and common stake in transactions, they can use one orderer each, with the assumption that members in the organization have full trust in the organization's own orderer. Each of the five organizations can have hundreds of participants, each participant with their own peer, but only five orderers are used for the decision of the order of transactions (and the slow BFT algorithm). The trust assumption for many scenarios however, including the PoC in this thesis, requires one orderer per participant.

The design mentioned in Chapter 3 results in both advantages and disadvantages when it comes to scalability compared to worst-case scenarios. The trust assumption, where no one fully trusts anyone and a user has no clear connection to other users, requires one orderer and one peer for every user. It also requires a third of the members to endorse a transaction. On the other hand, the application requires a very low amount of transactions and can accept high delays before blocks are appended. If designers of another application were to choose between a blockchain and a database, they would have to decide on how many users they would like to support, both considering the unexplored nature of high numbers of orderers in Fabric and the $\mathcal{O}(n^2)$ scaling of the BFT algorithm it relies on, but also when looking at throughput and latency. As a reminder, in Section 4.2 it was argued that Fabric had many advantages over other permissioned blockchains when looking at scalability and maturity. A reasonable conclusion is that other blockchain projects relying on BFT algorithms would scale just as poorly. In Section 7.3, the performance of Bitcoin, a well-known public blockchain, was discussed and using Bitcoin would result in even higher delays and lower throughput than with Fabric. The literature study performed to answer this question resulted in a table, Table 7.4, and a decision tree, Fig. 7.5 that hopefully will prove helpful when deciding between blockchains and traditional databases.

## 8.2 What considerations and limitations are there for applying the design in a permissioned blockchain?

A permissioned blockchain was chosen because of the nature of the PoC. A user should not be able to vote multiple times on the same program, masking as different users. By having a permissioned blockchain and identities tied to every participant, the PoC is able to hinder such attacks.

The considerations and limitations of permissioned blockchains are based on the results found in Chapter 6, the comparison in Chapter 7, and the experience from designing and programming a PoC on top of Fabric.

### 8.2.1 Identities

Every entity on the network has an identity in the form of a certificate provided by the MSP. The private keys must be kept safe, lest an attacker is able to pose as a valid user on the network and execute malicious transactions. This consideration is not particularly unique to permissioned blockchains, but holds true for any protocol with private/secret keys, for instance TLS and SSH.

### 8.2.2 Trust

In a network with $n$ users $n = 3f + 1$, $f + 1$ "well-behaving" orderers have to be online in the network at any given time. If the number of unreachable orderers ever were to reach $f + 1$, the network would become unusable. The role of orderer should therefore only be assigned to servers running around the clock. The design choice in the PoC of making every participant an orderer might not be sustainable in real-life, if there is a high risk that many of them turn off their machines when not using them.

Compare this to Bitcoin were the network is functional as long as $> 50\%$ of the mining power behaves correctly, and there is no reliance on any particular node to be online and correct.

It also places a substantial trust in the MSP to act correctly, as is the case with CAs in TLS. This assumption is somewhat less in the case of Fabric where rules can be placed requiring a majority or even all users to agree before a new member is allowed into the network, and thus have the ability to deny entry.

The benefit of linking real-world identities to the identities used in the blockchain is the same as its drawback: Someone must be able to give a certificate to a user, and the others must be able to verify or at least trust that the certificate has been given to the right person. In a corporate setting, an *Active Directory* might be used where these identities already exist. Another case could be a classroom, a sports organization, or a section of an online forum where the members can be verified, and they can raise concerns if they are not in control of their certificate. Nevertheless, the MSP will in some sense require additional trust compared to the rest of the system.

### 8.2.3 Immature projects

Permissionless blockchains gathered interest first after Nakamoto published his paper in 2008 [11], making them an immature technology — especially when being compared to traditional databases. Permissioned blockchains came after this, and are yet more immature. Fabric, and the entire Hyperledger umbrella project, started in December 2015 [36]. As was mentioned in Section 4.2, Fabric seems to be one of the more mature permissioned blockchain projects. As of writing, Fabric is no more than 2.5 years old, and v1.0 released less than one year ago [86]. Fabric is still prone to large scale design changes, as the one from v0.6 to v1.0, described in Section 4.2.4, resulting in non-trivial and time-consuming upgrades. Upcoming in v1.2 is for instance: pluggable endorsement, service discovery, and changes to the confidentiality model [87]. They also lack important features like installers,

pruning, and BFT consensus, all of which they plan on adding but have no exact time frame for [44, 45, 85, 88].

Developing a security critical application on blockchains at this stage would require thorough testing of both the application and underlying blockchain layer.

### 8.2.4 Scaling

As was discussed in Section 8.1, scaling is a consideration when building upon permissioned blockchains. Permissioned blockchains have better throughput than their permissionless alternatives, but the difference shrinks, and eventually turns negative, as the number of nodes increases.

## 8.3 What are the advantages when implementing the program logic in a smart contract compared to outside of the blockchain?

Answering this question in a general sense is best done by referencing back to Section 7.2. In the case of the PoC, implementing the logic in smart contracts had a positive effect. The decision to implement some logic in a smart contract was motivated in Section 4.2.1. The performance degradation was tested in Section 6.3 and was found to not be substantial enough to make it the primary bottleneck of the system. The consensus algorithm will be unable to scale long before the smart contract does. Because Fabric runs smart contracts in Docker containers and not in a virtual machine as Ethereum does, the performance hit in general is less of a problem [41, 89].

Instead, the smart contract allows every entity to verify all changes being made to the blockchain, because of the endorsement policy. This ensures no one is able to enter erroneous values, unless the network has been compromised with $> \frac{n}{3}$ Byzantine nodes, where $n$ is the total amount of users. The code is also visible and verifiable by all participants, the same transparency is not guaranteed when the functionality is hidden in a client's binaries.

Having the code in smart contracts allowed the PoC to utilize the built-in state database, and this was convenient compared to writing a client for manually parsing the blocks in the blockchain. No further development was needed to store all the votes. Using a more traditional database would have required the development of both a server with a database, and a client to query the server.

# Chapter 9
# Conclusions

Blockchain technology is an interesting byproduct of the modern cryptocurrency craze. While some argue it will solve problems far and wide, others view it as a technology at its peak of inflated expectations [90]. The truth is found in between the extremes, and the question is where it is located. Blockchains are still very much in their infancy, and potential use-cases are constantly being investigated and evaluated. Especially the Financial Technology sector is interested in blockchains and how the technologies could be used to transform finance procedures [91].

This Master's Thesis investigated one new use-case for blockchains, and discussed how it might be implemented using Fabric and smart contracts. A promising PoC was developed with the potential to be helpful in enhancing end user security, but is far from ready for any commercial use. It suffers from complex configuration, questionable scalability, and an intrusive user interface.

Nevertheless, it brought insights into the development process of applications on top of blockchains, and how the peer-to-peer network is configured. These insights, together with tests of the PoC, were later used to motivate the answers to the original questions. The PoC was tested to gather information about scaling and performance. Problems with the scaling of nodes, and transaction size were identified in the tests.

A comparison to traditional databases was made, with regard to properties, performance, cost, and security. The comparison resulted in Table 7.4 and a decision tree, illustrated in Fig. 7.5. This comparison, together with the table and the decision tree, can hopefully be of assistance to developers considering including blockchains in their projects, and be of guidance about what considerations that have to be made in these areas when choosing a blockchain.

## 9.1 Future work

The developed PoC relies on identities in permissioned blockchains to assure one-person-one-vote, but if the PoC was used in a public blockchain it could be used collectively by thousands of people. In Section 4.2, drawbacks of implementing identities on top of a public blockchain was discussed. Work by Alexopoulos et al. [92] suggests different options, where they explored the possibility to use a permissionless blockchain in trust management for identification. They suggested

that their abstract design can be used for a PGP web of trust[1]. If a web of trust were to be implemented on a public blockchain for authentication, the same web of trust could be used as a weight-factor in the PoC developed for this thesis. If a user has strong trust for some members, their votes and data submissions could factor more into the verdict than those from members the user has low or no trust for.

One disturbing observation was the lack of research performed with Fabric with more orderers than 10 to 32, especially considering the difficulties that have been observed with more orderers. There are some obvious issues with setting up a network with hundreds of nodes for testing, but research on latency and throughput for 50, 100 or even 500 nodes would be helpful for the BFT-consensus-reliant area of blockchains. It is easy to imagine situations were the trust assumption require one orderer for each participant, and having a ceiling on amount of participants already at design phase would certainly be beneficial.

[1] A directed multigraph $G = (V, E)$ where each $v \in V$ is an entity, e.g. a person and each $e \in E$ is a trust relation. A formula is then used where partial trust or full trust for several entities vouching for an unknown entity may be enough to partially trust the unknown entity as well.

# References


[1] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, Dawn Song, and Roger Wattenhofer. On Scaling Decentralized Blockchains. *Financial Cryptography & Data Security (9783662533567)*, page 106, 2016.

[2] Symantec Corporation. Internet Security Threat Report. `https://www.symantec.com/content/dam/symantec/docs/reports/istr-22-2017-en.pdf`, 2017. Accessed: February 12, 2018.

[3] W. Beer. *Bitcoin Blockchain: A compact introduction into the blockchain of Bitcoin*. 2017.

[4] Jesse Yli-Huumo, Deokyoon Ko, Sujin Choi, Sooyong Park, and Kari Smolander. Where is current research on blockchain technology?—a systematic review. *PLoS ONE*, 11(10):1 – 27, 2016.

[5] Edoardo Gaetani, Leonardo Aniello, Roberto Baldoni, Federico Lombardi, Andrea Margheri, and Vladimiro Sassone. Blockchain-based database to ensure data integrity in cloud computing environments. 2017.

[6] Björn Johansson. Assessing blockchain technology for Transport Data Logger, 2018. Student Paper.

[7] Suporn Pongnumkul, Chaiyaphum Siripanpornchana, and Suttipong Thajchayapong. Performance Analysis of Private Blockchain Platforms in Varying Workloads. *2017 26th International Conference on Computer Communication and Networks (ICCCN), Computer Communication and Networks (ICCCN), 2017 26th International Conference on*, 2017.

[8] Tien Tuan Anh Dinh, Ji Wang, Gang Chen, Rui Liu, Beng Chin Ooi, and Kian-Lee Tan. BLOCKBENCH: A Framework for Analyzing Private Blockchains. 2017.

[9] Mattias Scherer. Performance and scalability of blockchain networks and smart contracts, 2017.

[10] Joseph Poon and Thaddeus Dryja. The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments. 2016.

[11] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System, 2008.

[12] David Voell, Frank Lu-Nick Gaski, Ram Jagadeesan, Renat Khasanshyn, Hart Montgomery, Stefan Teis, Tamas Blummer, Murali Krishna Katipalli, and Mic Bowman. Hyperledger Whitepaper. `https://wiki.hyperledger.org/groups/whitepaper/whitepaper-wg`, 2016. Accessed: January 26, 2018.

[13] Hyperledger Architecture Working Group. Hyperledger Architecture, Volume 1. `https://www.hyperledger.org/wp-content/uploads/2017/08/Hyperledger_Arch_WG_Paper_1_Consensus.pdf`, 2017. Accessed: February 27, 2018.

[14] Bruce Schneier. Cryptanalysis of MD5 and SHA: Time for a New Standard. `https://www.schneier.com/essays/archives/2004/08/cryptanalysis_of_md5.html`, 2004. Accessed: February 2, 2018.

[15] Horst Feistel. Cryptography and Computer Privacy. *Scientific American*, 228(5):15–23, 1973.

[16] Bruce Schneier. Cryptanalysis of SHA-1. `https://www.schneier.com/blog/archives/2005/02/cryptanalysis_o.html`, 2005. Accessed: February 2, 2018.

[17] Microsoft. PE Format. `https://msdn.microsoft.com/en-us/library/windows/desktop/ms680547(v=vs.85).aspx`. Accessed: February 8, 2018.

[18] Zahra Bazrafshan, Hashem Hashemi, Seyed Mehdi Hazrati Fard, and Ali Hamzeh. A survey on heuristic malware detection techniques. *The 5th Conference on Information and Knowledge Technology, Information and Knowledge Technology (IKT), 2013 5th Conference on*, page 113, 2013.

[19] Andreas Moser, Christopher Kruegel, and Engin Kirda. Limits of Static Analysis for Malware Detection. *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007), Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*, page 421, 2007.

[20] Waqas Aman. A Framework for Analysis and Comparison of Dynamic Malware Analysis Tools. 2014.

[21] David De Lille, Bart Coppens, Daan Raman, and Bjorn De Stutter. Automatically combining static malware detection techniques. *2015 10th International Conference on Malicious and Unwanted Software (MALWARE), Malicious and Unwanted Software (MALWARE), 2015 10th International Conference on*, page 48, 2015.

[22] Vitalik Buterin. Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform. `https://github.com/ethereum/wiki/wiki/White-Paper`, 2013. Accessed: January 26, 2018.

[23] Marko Vukolić. The Quest for Scalable Blockchain Fabric: Proof-of-Work vs. BFT Replication. *Open Problems in Network Security*, page 112, 2016.

[24] Adam Back et al. Hashcash-a denial of service counter-measure, 2002.

[25] Digiconomist. Bitcoin Energy Consumption Index. `https://digiconomist.net/bitcoin-energy-consumption`, 2018. Accessed: February 7, 2018.

[26] Power Compare. Bitcoin Mining Now Consuming More Electricity Than 159 Countries Including Ireland & Most Countries In Africa. https://powercompare.co.uk/bitcoin/, 2017. Accessed: February 7, 2018.

[27] Lin Chen, Lei Xu, Nolan Shah, Zhimin Gao, Yang Lu, and Weidong Shi. On Security Analysis of Proof-of-Elapsed-Time (PoET). In Paul Spirakis and Philippas Tsigas, editors, *Stabilization, Safety, and Security of Distributed Systems*, pages 282–297, Cham, 2017. Springer International Publishing.

[28] Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov, and Krzysztof Pietrzak. Proofs of Space. *Advances in Cryptology, Crypto 2015: 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, page 585, 2015.

[29] Sunny King and Scott Nadal. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake, 2012.

[30] Jonathan Jogenfors. Quantum bitcoin: An anonymous and distributed currency secured by the no-cloning theorem of quantum mechanics. 2016.

[31] David Siegel. Understanding The DAO Attack. https://www.coindesk.com/understanding-dao-hack-journalists/, 2016. Accessed: February 7, 2018.

[32] Matthew Leising. The ether thief. https://www.bloomberg.com/features/2017-the-ether-thief/, 2017. Accessed: February 7, 2018.

[33] Frank Hofmann, Simone Wurster, Eyal Ron, and Moritz Böhmecke-Schwafert. THE IMMUTABILITY CONCEPT OF BLOCKCHAINS AND BENEFITS OF EARLY STANDARDIZATION, 2017.

[34] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine Generals Problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, July 1982.

[35] Miguel Castro and Barbara Liskov. Practical Byzantine Fault Tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, OSDI '99, pages 173–186, Berkeley, CA, USA, 1999. USENIX Association.

[36] Linux Foundation. Linux foundation unites industry leaders to advance blockchain technology. http://www.linuxfoundation.org/press-release/linux-foundation-unites-industry-leaders-to-advance-blockchain-technology/, 2015. Accessed: February 12, 2018.

[37] Hyperledger. Members. https://hyperledger.org/members, 2017. Accessed: February 12, 2018.

[38] Hyperledger. Hyperledger Fabric Model. http://hyperledger-fabric.readthedocs.io/en/release/fabric_model.html, 2017. Accessed: February 12, 2018.

[39] Hyperledger. Transaction Flow. http://hyperledger-fabric.readthedocs.io/en/release/txflow.html, 2017. Accessed: February 12, 2018.

[40] Nick Szabo. The idea of smart contracts. *Nick Szabo's Papers and Concise Tutorials*, 6, 1997.

[41] Tien Tuan Anh Dinh, Rui Liu, Meihui Zhang, Gang Chen, Beng Chin Ooi, and Ji Wang. Untangling Blockchain: A Data Processing View of Blockchain Systems. 2017.

[42] Ethereum. go-ethereum/COPYING. `https://github.com/ethereum/go-ethereum/blob/master/COPYING`, 2015. Accessed: February 14, 2018.

[43] Hyperledger. HYPERLEDGER SAWTOOTH. `https://hyperledger.org/projects/sawtooth`, 2018. Accessed: February 14, 2018.

[44] Christian Cachin and Marko Vukolić. Blockchain Consensus Protocols in the Wild. 2017.

[45] Marko Vukolić. Rethinking Permissioned Blockchains. In *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts*, BCC '17, pages 3–7, New York, NY, USA, 2017. ACM.

[46] jcs47. Byzantine fault-tolerant ordering service for Hyperledger Fabric. `https://github.com/jcs47/hyperledger-bftsmart`. Accessed: March 27, 2018.

[47] kichik. . `https://github.com/kichik/pecoff4j`. Accessed: April 17, 2018.

[48] Jakob Nielsen. Website Response Times. `https://www.nngroup.com/articles/website-response-times/`, 2010. Accessed: May 18, 2018.

[49] Yacov Manevich. Minimize transaction size by saving identities to stateDB and referencing them. `https://jira.hyperledger.org/browse/FAB-8007`, 2018. Accessed: May 8, 2018.

[50] Nir Kshetri. Blockchain's roles in strengthening cybersecurity and protecting privacy. *Telecommunications Policy*, 41(Celebrating 40 Years of Telecommunications Policy - A Retrospective and Prospective View):1027 – 1038, 2017.

[51] Pierluigi Cuccuru. Beyond bitcoin: an early overview on smart contracts. *International Journal of Law & Information Technology*, 25(3):179 – 195, 2017.

[52] Jeremy M. Sklaroff. SMART CONTRACTS AND THE COST OF INFLEXIBILITY. *University of Pennsylvania Law Review*, 166(1):263 – 303, 2017.

[53] Marco Iansiti and Karim R. Lakhani. The Truth About Blockchain. `https://hbr.org/2017/01/the-truth-about-blockchain`, 2017. Accessed: April 5, 2018.

[54] Vitalik Buterin. CRITICAL UPDATE Re: DAO Vulnerability. `https://blog.ethereum.org/2016/06/17/critical-update-re-dao-vulnerability/`, 2016. Accessed: February 7, 2018.

[55] Bill Marino and Ari Juels. Setting Standards for Altering and Undoing Smart Contracts. *Rule Technologies. Research, Tools & Applications*, pages 151 – 166, 2016.

[56] FinTech Network. Smart Contracts – From Ethereum to Potential Banking Use Cases. https://blockchainapac.fintecnet.com/uploads/2/4/3/8/24384857/smart_contracts.pdf. Accessed: April 23, 2018.

[57] Hyperledger. Channels. http://hyperledger-fabric.readthedocs.io/en/release-1.1/channels.html, 2017. Accessed: April 23, 2018.

[58] Maher Alharby and Aad van Moorsel. Blockchain-based Smart Contracts: A Systematic Mapping Study. 2017.

[59] CoinMarketCap. Top 100 Cryptocurrencies by Market Capitalization. https://coinmarketcap.com/, 2018. Accessed: May 08, 2018.

[60] Visa Europe. https://www.visaeurope.com/about-us/, 2018. Accessed: March 13, 2018.

[61] Tony Kontzer. Inside Visa's Data Center. https://www.networkcomputing.com/networking/inside-visas-data-center/1599285558, 2013. Accessed: May 3, 2018.

[62] Carlos Domingo. The Bitcoin vs Visa Electricity Consumption Fallacy. https://hackernoon.com/the-bitcoin-vs-visa-electricity-consumption-fallacy-8cf194987a50, 2017. Accessed: March 29, 2018.

[63] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukolić, Sharon Weed Cocco, and Jason Yellick. Hyperledger fabric: A distributed operating system for permissioned blockchains. 2018.

[64] Joao Sousa, Alysson Bessani, and Marko Vukolić. A byzantine fault-tolerant ordering service for the hyperledger fabric blockchain platform. *arXiv preprint arXiv:1709.06921*, 2017.

[65] Jennifer Xu. Are blockchains immune to all malicious attacks? *Financial Innovation*, 2(1):1, 2016.

[66] Parity. https://www.parity.io/, 2018. Accessed: April 20, 2018.

[67] João Sousa and Alysson Bessani. Separating the WHEAT from the chaff: An empirical design for geo-replicated state machines. In *Reliable Distributed Systems (SRDS), 2015 IEEE 34th Symposium on*, pages 146–155. IEEE, 2015.

[68] End Point. Benchmarking Top NoSQL Databases. https://www.datastax.com/wp-content/themes/datastax-2014-08/files/NoSQL_Benchmarks_EndPoint.pdf, 2015. Accessed: May 17, 2018.

[69] Steve McConnell. *Code Complete, Second Edition*. Microsoft Press, Redmond, WA, USA, 2004.

[70] Cryptocurrency statistics. https://bitinfocharts.com/, 2018. Accessed: March 16, 2018.

[71] S. Haber and W.S. Stornetta. How to time-stamp a digital document. *Journal of Cryptology*, 3(2):99 – 111, 1991.

[72] Xiaoqi Li, Peng Jiang, Ting Chen, Xiapu Luo, and Qiaoyan Wen. A survey on the security of blockchain systems. *Future Generation Computer Systems*, 2017.

[73] Mauro Conti, Sandeep Kumar E, Chhagan Lal, and Sushmita Ruj. A Survey on Security and Privacy Issues of Bitcoin. 2017.

[74] Council of European Union. Council regulation (EU) no 2016/679. `https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=uriserv:OJ.L_.2016.119.01.0001.01.ENG`, 2016.

[75] Trunomi. GDPR Key Changes. `https://www.eugdpr.org/key-changes.html`. Accessed: May 15, 2018.

[76] Matthias Berberich and Malgorzata Steiner. Blockchain Technology and the GDPR - How to Reconcile Privacy and Distributed Ledgers? *European Data Protection Law Review (EDPL)*, (3):422, 2016.

[77] Dharavath Ramesh and Chiranjeev Kumar. An optimal novel byzantine agreement protocol (onbap) for heterogeneous distributed database processing systems. *Procedia Technology*, 6:57–66, 2012. 2nd International Conference on Communication, Computing & Security [ICCCS-2012].

[78] Chen Shiping, A. Ng, and P. Greenfield. A performance evaluation of distributed database architectures. *Concurrency and Computation: Practice and Experience*, 25(11):1524 – 1546, 2013.

[79] Harish Balasubramanian. Performance analysis of scalable sql and nosql databases: A quantitative approach, 2014.

[80] Shehar Bano, Alberto Sonnino, Mustafa Al-Bassam, Sarah Azouvi, and Patrick McCorry. SoK: Consensus in the Age of Blockchains. 2017.

[81] Morgen E. Peck. Blockchain world - do you need a blockchain? this chart will tell you if the technology can solve your problem. *IEEE Spectrum, Spectrum, IEEE, IEEE Spectr*, (10):38, 2017.

[82] Roger Maull, Phil Godsiff, Alan Brown, Beth Kewell, and Catherine Mulligan. Distributed ledger technology: Applications and implications. *Strategic Change*, 26(5):481–489, 2017.

[83] Vitalik Buterin. Ethereum: Platform review; opportunities and challenges for private and consortium blockchains, 2016.

[84] Hyperledger. Frequently Asked Questions. `https://www.hyperledger.org/wp-content/uploads/2018/01/Hyperledger_Sawtooth_FAQ.pdf`, 2018. Accessed: May 08, 2018.

[85] Hyperledger Fabric. Architecture explained. `http://hyperledger-fabric.readthedocs.io/en/release-1.1/arch-deep-dive.html#post-v1-validated-ledger-and-peerledger-checkpointing-pruning`, 2018. Accessed: May 22, 2018.

[86] Hyperledger. Hyperledger Fabric 1.0 is Released! https://www.hyperledger.org/blog/2017/07/11/hyperledger-fabric-1-0-is-released, 2017. Accessed: April 24, 2018.

[87] Hyperledger. https://wiki.hyperledger.org/projects/fabric/roadmap, 2018. Accessed: April 24, 2018.

[88] Hyperledger Fabric. Getting started. https://hyperledger-fabric.readthedocs.io/en/release-1.1/getting_started.html, 2018. Accessed: May 15, 2018.

[89] Flavio Ramalho and Augusto Neto. Virtualization at the network edge: A performance comparison. *2016 IEEE 17th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM), World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2016 IEEE 17th International Symposium on A*, page 1, 2016.

[90] Kasey Panetta. 3 Trends Appear in the Gartner Hype Cycle for Emerging Technologies, 2016. https://www.gartner.com/smarterwithgartner/3-trends-appear-in-the-gartner-hype-cycle-for-emerging-technologies-2016/, 2016. Accessed: May 15, 2018.

[91] Ittay Eyal. Blockchain Technology: Transforming Libertarian Cryptocurrency Dreams to Finance and Banking Realities. *Computer*, (9):38, 2017.

[92] Nikolaos Alexopoulos, Jörg Daubert, Max Mühlhäuser, and Sheikh Mahbub Habib. Beyond the Hype: On Using Blockchains in Trust Management for Authentication. 2017.

[93] Benhut1. Portable executable 32 bit structure in svg fixed. https://commons.wikimedia.org/wiki/File:Portable_Executable_32_bit_Structure_in_SVG_fixed.svg, 2016. Accessed: February 2, 2018.

# Appendix A
# Figures

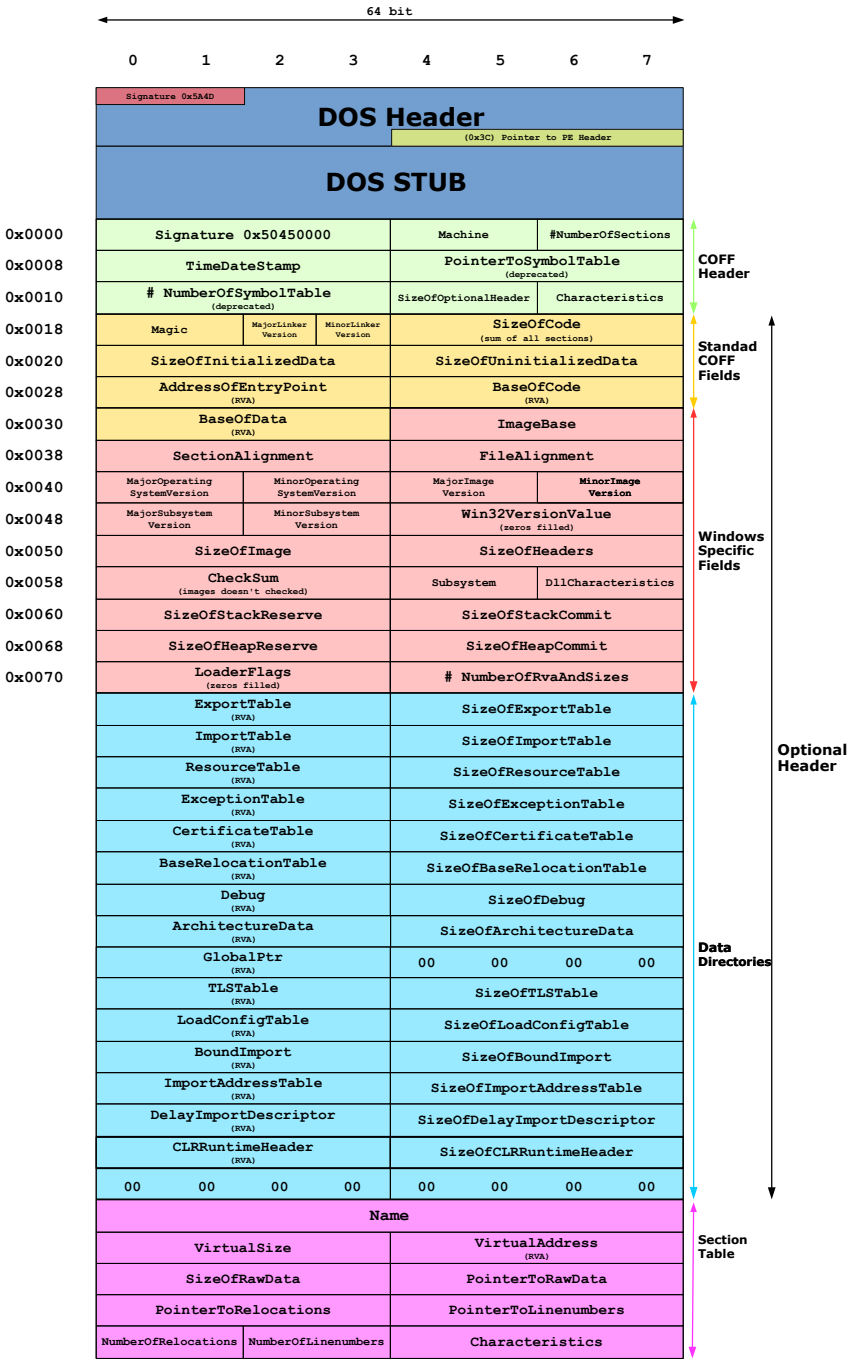


**Figure A.1:** PE File format. By Benhut1 [93]. Distributed under a CC-BY-4.0 license

Printed by Tryckeriet i E-huset, Lund 2018

LUND
UNIVERSITY

Series of Master's theses
Department of Electrical and Information Technology
LU/LTH-EIT 2018-643
http://www.eit.lth.se