University of Zurich[UZH]

MASTER THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

Design and Development of a Blockchain Interoperability API

Timo Hegnauer
Zürich, Switzerland
Student ID: 11-722-766

Supervisor: Eder Scheid, Bruno Rodrigues
Date of Submission: February 17, 2019

University of Zürich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Master Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

Abstract

In den letzten Jahren entstand eine Vielzahl verschiedener Blockchains. Zielkonflikte in den Bereichen Funktionalität, Anonymität und Effizienz verhindern dabei die Dominanz einer Blockchain in allen Anwendungsfällen. Interoperabilität ist deshalb zentral, um die auf verschiedene Anwendungsfälle zugeschnittenen Blockchains zu vernetzen. Der vielversprechendste Ansatz ist dabei Sidechains. Sidechains können Informationen von anderen Blockchains lesen und verarbeiten. Allerdings sind sie nur schwer in existierende Blockchains integrierbar. Im Kontext der Interoperabilität ist es zudem entscheidend, dass Entwickler mit verschiedenen Blockchains interagieren können, ohne die Implementation jeder Blockchain zu kennen. Diese Masterarbeit stellt deshalb eine API vor, die Speichern und Lesen von Daten auf sieben verschiedenen Blockchains ermöglicht. Dafür wird ein sogenanntes Notary-Schema verwendet. So kann die Lösung potenziell mit allen existierenden Blockchains verwendet werden. Der Nachteil dieses Ansatzes ist das nötige Vertrauen in den Notary.

A large number of blockchains emerged in the past years. Due to trade-offs regarding features, privacy and efficiency, no single blockchain will be able to cover and dominate all use cases. Instead, interoperability is needed to link different blockchains by exchanging information and assets. The most promising approach for interoperability is the use of sidechains. Sidechains can read and process information about the state of other blockchains. However, they are difficult to implement into existing blockchains. To further support interoperability, developers must be able to interact with multiple blockchains without knowing the details of each implementation. This thesis presents an API solution to store and retrieve data on seven different blockchains. A notary scheme is used, which allows it to connect to existing blockchains but requires trust in the notary.

Acknowledgments

I want to thank my supervisors Eder Scheid and Bruno Rodrigues for supporting me during the process of writing the thesis with excellent feedback and suggestions. Furthermore, thanks to Patrick Widmer which provided useful previous work and Isabel Stiefel for the review.

Contents

Abstract **i**

Acknowledgments **iii**

1 Introduction **1**

1.1 Description of Work 2

1.2 Thesis Outline 2

2 Theoretical Background **5**

2.1 Building Blocks of Blockchains 5

2.1.1 Ledger 5

2.1.2 Node Application 5

2.1.3 Consensus Mechanisms 6

2.2 Blockchains Overview 11

2.2.1 Bitcoin 11

2.2.2 Ethereum 12

2.2.3 Stellar 12

2.2.4 EOS 12

2.2.5 IOTA 13

2.2.6 Hyperledger Sawtooth 13

2.2.7 Multichain 14

2.2.8 Comparison 14

2.3 Interoperability Use Cases 14

2.3.1 Asset portability 14
2.3.2 Atomic Swap 15
2.3.3 Cross-chain oracles 15
2.3.4 Asset encumbrance 15
2.3.5 Cross-chain contracts 15
2.4 Interoperability Techniques 15
2.4.1 Notary Schemes 16
2.4.2 Sidechains 16
2.4.3 Hash-locking 17
2.4.4 Overview 17

3 Related Work **19**
3.1 Notary Schemes 19
3.1.1 Herdius 19
3.2 Sidechain/Relay 19
3.2.1 BTC-Relay 19
3.2.2 Aion 20
3.2.3 Aelf 20
3.2.4 Cosmos 20
3.2.5 Polkadot 21
3.2.6 Ark 21
3.2.7 VirtualChain 21
3.2.8 ICON 22
3.3 Hash Locking 22
3.3.1 Wanchain 22
3.3.2 Interledger 22
3.4 Other projects 23
3.4.1 Crowd Machine 23
3.4.2 Origintrail.io 23
3.5 Discussion 23

4 Interoperability API Solution **25**

4.1 Objective . . . 25

4.2 Solution Design . . . 26

4.2.1 Application Programming Interface (API) . . . 26

4.2.2 Blockchain Adapters and Nodes . . . 27

4.2.3 Database . . . 29

4.2.4 Environment and Dependencies . . . 30

5 Evaluation and Discussion **31**

5.1 Performance Analysis . . . 31

5.2 Security Analysis . . . 32

5.2.1 Private Key Management . . . 32

5.2.2 Local and Remote Nodes . . . 33

5.2.3 Centralization . . . 33

5.3 Data size analysis . . . 34

6 Conclusion and Future Work **35**

6.1 Future work . . . 36

Abbreviations **43**

Glossary **45**

List of Figures **45**

List of Tables **47**

A Installation Guidelines **51**

A.1 Usage 51

A.2 Setup 51

A.2.1 Prerequisites 51

A.2.2 Install Dependencies 52

A.2.3 Database Setup 52

A.2.4 Setup for each blockchain 52

B Contents of the CD **53**

Chapter 1

Introduction

The blockchain concept was introduced in 2009 with the release of the Bitcoin white-paper [1]. Bitcoin is considered to be the first blockchain-based cryptocurrency, achieving a market capitalization of more than 109 billion US dollars [2] in 2018. After its release, several cryptocurrencies were created taking advantage of the blockchain technology and the success of Bitcoin. As of 2018, there exist around 2000 different cryptocurrencies, some following the Bitcoin blockchain protocol, but also a great number of them using their own blockchain protocol and implementation [2]. In contrast to conventional databases, a blockchain is distributed and immutable employing cryptography to remove the need of a Trusted Third Party to secure and validate transactions within such systems. The use cases of the blockchain technology range from cross-border transactions to complex smart contract handling insurance claims [1, 3, 4].

Several blockchain implementations were proposed, each focusing on solving specific shortcomings of the original blockchain proposal, *i.e.,* Bitcoin. On the one hand, some implementations focus on increasing the number of transactions per second that the blockchain can handle by optimizing transaction-relevant properties. However, this might lead to an increase in the blockchain's centralization [5]. On the other hand, some implementations focus on special features, such as smart contract support [3], privacy [6] or digital identities [7]. Others allow enterprises to create private blockchains, where only a defined set of entities can participate [8].

However, there are conflicts between those properties. For example, a high degree of decentralization leads to increased confirmation times and a focus on privacy limits the identity features. Thus, blockchain proposals need to position themselves in the market by defining their vision and targeting a set of use cases. Considering the wide spectrum of use cases found in the literature [9], it is unlikely that there will exist one dominant blockchain which perfectly fits all the needs for every use case.

One problem that arises with the increasing number of blockchain implementations is the isolation of such systems. Because of their different protocols and technologies, native cryptocurrencies cannot be exchanged among each other and it is not possible to exchange information about their state or events directly. Thus, the research on blockchain interoperability is a promising field to address such a problem, being referred to as the "holy

grail" of blockchains [10]. Interoperability, in general terms, is defined as "*The ability of computer systems or software to exchange and make use of information.*" [11].

In the blockchain context, interoperability means connecting multiple blockchains to access information and act on it by changing the state of the own or another blockchain. Optimally, this would be achieved without breaching the blockchain's promise of trustlessness.

In a more practical and user focused context, interoperability also relates to the goal that a user (*e.g.,* application developer) can work with any blockchain without restrictions. Currently, one major restriction is that users need to possess detailed knowledge about the technical details to be able to interact with different blockchains. However, if blockchains and cryptocurrencies become widely used and present in the daily life, this complexity needs to be hidden from the user and replaced with simple and transparent interfaces. Hence, this thesis proposes an approach to deliver an easy-to-use interface for the interaction with multiple blockchains.

The main driver of interoperability is the linking of blockchains which are specialized for their use cases. For example when a user buys a car using a public blockchain (*e.g.,* PaymentChain) and wants to automatically register it on a private blockchain (*e.g.,* InsuranceChain). Nevertheless, there are other reasons where users can benefit from blockchain interoperability. One is to increase performance of blockchains by linking multiple smaller blockchains. Another one is to decrease the risk of unavailability or attacks of smaller blockchains by providing a kind of backup on another blockchain [12].

1.1 Description of Work

This thesis addresses interoperability between blockchains and presents a solution to easily interact with multiple blockchains. It reviews the current state-of-art in blockchain interoperability considering both theoretical work and existing projects in the field. The main contribution of this thesis is the prototype of a interoperability Application Programming Interface (API). This prototype delivers a simple interface for users to interact with multiple private and public blockchains in the form of a Python API. It enables the user to store and retrieve any arbitrary data on a given blockchain, without the need to understand the underlying blockchain libraries or technical details. From a theoretical point of view, it provides this API based on an interoperability technique called "notary scheme", which is explained in Section 2.4. In such a scheme, the prototype runs on a trusted server providing the interface to application developers or end-users. Furthermore, an evaluation of the interoperability API is performed, presenting measurements of the different blockchains adapters and discussing properties such as security and centralization.

1.2 Thesis Outline

This thesis starts by providing theoretical background information to the reader in Chapter 2. This chapter covers the basics about blockchains with an emphasis on consensus

mechanisms. It also introduces the set of blockchains present in the interoperability API solution and their main properties. It continues by presenting a classification of interoperability use cases and techniques. Next, Chapter 3 presents related work by discussing existing interoperability solutions. Further, in Chapter 4, the Interoperability API solution is presented and the reasoning behind the architecture is described. This is followed by Chapter 5, which analyzes the prototype in terms of performance, security and centralization and discusses the implication of the design decisions. Finally, Chapter 6 concludes the thesis and presents an outlook regarding future work.

Chapter 2

Theoretical Background

As introduced in Chapter 1, blockchains need to focus on a subset of the various use cases and set their properties accordingly. The need for interoperability is therefore rooted in the variety of blockchains architectures and their underlying technologies. To explain the fundamentals of those differences, this chapter starts by describing the building blocks of the blockchain technology. The subsequent section introduces each blockchain used in the implementation part of this work. The chapter continues by discussing the theoretical aspects of interoperability starting with the possible use cases. Furthermore, it explains techniques to achieve interoperability and finally maps those techniques to the use cases.

2.1 Building Blocks of Blockchains

This section explains the building blocks of blockchains: the ledger, the node application and the consensus mechanism. A special emphasis is put on the consensus mechanisms as they are often the main difference between blockchains and have an impact on the use cases they are able to support.

2.1.1 Ledger

The ledger is the data storage of a blockchain. It is consensually shared and synchronized with the nodes. Depending on the blockchain, it holds the records of the transactions, the state of accounts or smart contracts [13].

2.1.2 Node Application

The node application is required to let nodes interact with the overlay that defines the network. It allows the operation of a blockchain by letting nodes process transactions. This application usually consists of a code library, an API description or a software wallet.

Additionally, interfaces can be provided to interact with the blockchain programmatically using the Hypertext Transfer Protocol (HTTP) or a Remote Procedure Call (RPC) [13].

There are three different types of node applications: full nodes, miner nodes, and light nodes.

Full nodes store every block of the chain and check transaction against the consensus rules. For example, they assure that there is no double spending inside of a block or validate that the signatures are matching the transaction's originator [1, 13].

New transactions are collected and included in a new block by the second type, so-called **miner nodes**[1]. In addition to being full nodes, miners contribute actively to the extension of the ledger by including new transactions, usually by solving a cryptographic hash puzzle [1].

The third type of nodes are **light nodes**. Light nodes are less resource-demanding because they do not store or verify every block or transaction, they only verify the "block headers" of the longest chain. Light nodes rely on full nodes to provide them with missing details or particular functionality. The implementation and features of light nodes can differ between different blockchains. Often, a technique called Simplified Payment Verification (SPV) is used.

Simplified Payment Verification (SPV)

SPV uses block headers and "Merkle trees" to verify if a transaction occurred without having to download the whole ledger. SPV is used by light clients but also by sidechains, as Section 2.4.2 explains.

The "Merkle root" is taken from the block header and consists of a hash of all the transaction hashes in the block. Together with the Merkle Tree of a transaction it thus can be validated if a transaction is located in a certain block. Furthermore, the block header is used to ensure that a transaction is considered valid in the network. Depending on the consensus mechanism (Section 2.1.3), either enough blocks must to be built upon the transaction or the block header needs to be signed by two thirds of the validators [1, 14, 15].

2.1.3 Consensus Mechanisms

At the core of every blockchain is the consensus mechanism. It ensures that there is only one recognized ledger in the network and thus prevents double spending. The main challenge is the open and distributed nature of blockchains, making it impossible to prevent malicious actors from joining the network and to prevent node failures [13].

There are three desirable properties of consensus protocols:

[1]Miner is the name in the case of Proof-of-Work (PoW). Also known as validators or minters in Proof-of-Stake (PoS) blockchains.

- Safety: All the outputs (of non-faulty/malicious nodes) have the same value (*i.e.,* came to an agreement) and all the outputs equal at least one of the agents inputs.
- Liveness: Ensure that eventually all non-faulty nodes output a value.
- Fault tolerance: It can recover from the failure of an agent. The relevant fault tolerance in the blockchain environment is "Byzantine-fault-tolerance", which is able to deal with arbitrary behavior and is explained in more detail below.

The Fischer Lynch Paterson (FLP) impossibility result says that only two out of these three properties can be fulfilled in an asynchronous system[2]. Thus, every blockchain can chose two out of these three properties [17].

Block Finality

The safety property is of particular importance for interoperability trough the concept of "block finality". A block which reached block finality can not be reverted by the network. This is a key aspect regarding interoperability, as Daniel Larimer from EOS explains: "Objective and unambiguous 100% finality is a critical property for all blockchains that wish to support inter-blockchain communication. Absent 100% finality, a reversion on one chain could have irreconcilable ripple effects across all interconnected chains [18]." This is why block finality is considered during the discussion of each consensus mechanisms below.

The Problem: Byzantine General Problem

The consensus problem is represented in the literature by the Byzantine General Problem: A general an his lieutenants have encircled the city of Rome with their armies; to successfully take Rome the generals and the lieutenants must come to a consensus about attacking or retreating in unison.

One problem is that messages from the general to the lieutenant or between lieutenants could be lost. Furthermore, the general or some of the lieutenants could be traitors and communicate the wrong decision. This corresponds to the situation in a network, where it is not clear if nodes are trustworthy (*i.e.,* lie) or if they are faulty (*i.e.,* message does not arrive). As the only goal is to get an agreement, it is not important if the consensus consists of the general's initial message as long as consensus is reached.

Byzantine Fault Tolerance (BFT)

Lamport, Pease, and Shostak [19] proposed the following solution to the problem:

[2]Meaning there are no guaranteed bounds on network latency even between correctly functioning nodes[16]

1. The commander sends his value to every lieutenant.

2. For each i, let v_i be the value Lieutenant i receives from the commander, or else be RETREAT if he receives no value. Lieutenant i acts as the commander in algorithm $OM(m-1)$ to send the value v_i to each of the $n-2$ other lieutenants.

3. For each i and each j, let v_j be the value Lieutenant i received from Lieutenant j in (2) or else RETREAT if he received no such value. Lieutenant i uses the majority value from $v_1 ... v_{n-1}$.

The algorithm works as long as their are only m traitors out of $3 \times m$ generals (*i.e.,* 33% faulty nodes in network). Each lieutenant's final choice comes from the majority of all lieutenant's choices. Even if the commander sends a different message to all his lieutenant, they will reach a consensus (using a majority or a default option) to retreat or attack. This solution does fulfill fault-tolerance, safety and liveness as long as two thirds of the nodes are honest and the system is synchronous.

There are different solutions to tackle the resulting scalability problem, as the protocol is very message intensive. Those solutions are either extending the aforementioned solution or taking a different approach to reach fault tolerance, mostly by using communication restrictions or digital signatures. The following sections explain the most widely used consensus protocols [19].

Proof-of-Work (PoW)

Proof-of-Work solves the consensus problem using a concept called mining. By finding the solution to a computing power intensive hashing puzzles, a miner earns the right to create a new block. If the created block is considered to be valid by other nodes, the miner is rewarded in the form of cryptocurrency. As the longest chain is considered the valid one, there is an economic incentives to create a block on the longest chain, having the highest probability to be considered valid and build upon by other nodes.

The concept of mining creates two issues. First, the possibility of 51% attacks. An attacker with a majority of the total mining power can alter past blocks and subsequently overtake the currently longest and previously valid chain. To prevent such an attack, mining is very energy and hardware intensive. This means in turn, that PoW blockchains cause environmental issues through their energy consumption [20]. The second issue is centralization. PoW is completely decentralized in theory, because anyone can be a miner. In practice however, mining pools control a big portion of the mining power [21].

Among other issues *e.g.,* scalability, those two issues lead to research and implementation of alternative consensus mechanisms described in the following sections.

PoW favors liveness over safety, as there can be different states of the ledger (called a fork) possible, even though there are incentive to prevent this scenario in the long run. By their nature, a block in PoW does never reach finality. The probability of finality only increases as blocks are built upon each other [1].

Proof-of-Stake (PoS)

Proof-of-Stake was proposed to solve the energy-consumption problem of PoW blockchains and was first implemented in Peercoin [22]. Instead of choosing block creators by a computational competition, minters (equivalent to a miner in PoW) are randomly chosen out of all nodes. However, the probability to be chosen depends on the amount of coins a node is holding (his stake). In the absence of this additional property, a malicious actor could operate a large number of nodes and would thus be able to create a large portion of the new blocks, enabling double spending. One problem with this approach is the so called "nothing-at-stake" problems. As there is no incentive for the minters not to sign every block on every fork, forks would never be abandoned, thus a consensus could not be reached [22]. Thus, newer PoS approaches introduce a penalty for signing two conflicting block headers by destroying a stake which the minter has to lock up beforehand. This creates an analog mechanism as in PoW, where resources are wasted if mining is done on an abandoned block [16].

Beside the advantage in terms of energy consumption, a 51% attack is more expensive as a majority of all coins need to be hold. Furthermore a successful attack would presumably cause the price of the connected cryptocurrency to fall, decreasing the value of the attacker's assets.

According to the original definition PoS favors liveness over safety. By penalizing the signing of conflicting headers, there is a weak form of block finality called "economic finality" as it is exceedingly expensive to revert a block [23]. However, PoS approaches are often combined with BFT consensus. In this case safety is favoured over liveness. Thus, if a partition in the network happens, every partition could lack the majority of votes needed for consensus and the network would not be able to continue to approve transactions [16].

Delegated Proof-of-Stake (dPoS)

Delegated Proof-of-Stake seeks to improve scalability compared to the classic PoW and PoS mechanisms by introducing a voting and delegation system. Coin holders can vote for a set of "witnesses", who will take turns verifying and creating new blocks.

Performance is improved by increasing the hardware specification for witnesses and by limiting the confirmation process for new blocks the small set of witnesses. Compared to PoS, even nodes with little assets but adequate hardware can become witnesses. At the same time nodes with large assets have power though voting, as the voting power is proportional to the value of coins they are willing to stake.

Although decentralization is increased by giving every node the chance to vote, this could be outweighed by the centralization introduced by the small (*e.g.,* 21 at EOS) group of witnesses. Each malicious witnesses could censor transactions and together they could double spend, even though nodes could detect it and vote for other witnesses. Additionally there is an incentive for validators to bribe nodes for their vote, as witnesses are financially rewarded for their work [24].

Forks in dPoS are possible *e.g.,* if the propagation of a block to the other witnesses fails until a new block has to be built. Similarly to PoW, the longest chain is therefore the "legit" one because most of the block producers agree with this version. Thus PoS favours liveness over safety and has the aforementioned economic form of finality. However the stakes of the witnesses are higher in comparison [5, 25, 26].

Delegated Byzantine Fault Tolerance (dBFT)

Delegated Byzantine Fault Tolerance follows the initial solution from Lamport, but addresses the scalability issues by having a small number of nodes which have to come to a consensus before the block is created. Those nodes are voted on by all the coin holders similar to dPoS. The whole process is divided into three phase: pre-prepared,prepared and commit. In each phase, a block would enter next phase if it has received votes from over two thirds of all nodes.

Due to the multiple rounds system, dBFT has the advantage of block finality, but sacrifices liveness. If one third of the producers *e.g.,* fail to contribute to a consensus, block generation stops. It also sacrifices decentralization by having only a small number of block creators, which is addressed by the voting scheme [27].

Stellar Consensus Protocol (SCP)

The Stellar Consensus Protocol uses an adapted version of BFT. Instead of coming to a consensus with all other nodes (BFT) or voting for a set of block producers (dBFT), each node chooses its own set of trusted nodes, called quorum slice. A quorum slice is the subset of a quorum able to convince one particular node of agreement, meaning it will take the same output as their quorum slice nodes. SCP uses a three-round voting (vote, accept, confirm). Nodes which vote for a specific outcome can change their opinion after every round to agree with their quorum slice.

Because of the use of quorum slices and the fact that they are highly overlapping a node does not need to communicate with every other node to reach consensus. This is how the scalability issues of BFT are handled by SCP.

SCP favours safety over liveness and thus has the advantage of block finality. The system could get stuck if the nodes cannot agree on a vote. In this case, an additional vote to discard one options of the previous vote, so that a consensus can be found in a subsequent round [28].

Proof-of-Elapsed-Time (PoET)

PoET uses time to randomly select the block creators. Each node is required to wait for a randomly defined period of time. The first one who finishes the waiting period, is able to write the next block.

It must be ensured that a node selects a truly random wait time and has completed this waiting period before creating a block. This is why PoET makes use of special hardware *e.g.,* Intels Software Guard Extensions (SGX) technology, which includes a so called "enclave". Enclaves are physically isolated part of the processor and thus protected form processes running at higher privilege levels. Therefore one can run signed code without the possibility of interference or manipulation and have proof that this exact code was executed [29]

PoET can lead to a very high transaction throughput. The downside to this approach is the specific hardware needed. Furthermore an attacker could run a lot of nodes and thus increase his changes to be selected as block producer. Thus, PoET is mostly used in private blockchains [29, 30].

Proof-of-Authority (PoA)

Proof of Authority is often used in private blockchains and testnets. It is optimized for performance and low hardware requirements. In PoA blockchains, a fixed set of block creators and validators are defined. The disadvantage of this approach is the centralization. The chosen block creators must be trusted or malicious behaviour deterred by using some form of reputation system. This approach favours safety over liveness as a failure of the authorized nodes could stop block generation [31].

2.2 Blockchains Overview

This section provides an overview of the blockchains chosen for the prototype and their properties.

The blockchains were selected considering factors such as current market capitalization, capabilities (*e.g.,* smart contracts) and possible use cases. Even though it is uncertain which blockchains will be the most important in the future, those are some indications of their current and future success.

2.2.1 Bitcoin

Bitcoin was released in 2009 as the first decentralized cryptocurrency. The underlying Bitcoin blockchain was the first to solve the double spending problem by introducing the Proof-of-Work consensus mechanism. Being the first cryptocurrency, it attracted a lot of attention and investors resulting in a market capitalization of 60 billion dollars, by far the highest of all cryptocurrencies [2].

Disadvantages are the PoW mechanism, causing issues in terms of power consumption, scalability and centralization. As the implementation is very focused on supporting the Bitcoin cryptocurrency, Bitcoin lacks smart contract capabilities [1].

Many blockchains based on the Bitcoin source code emerged. Two notable examples are Bitcoin Cash and Litecoin. Both of them are very similar to Bitcoin but made changes to increase scalability by increasing the block size (Bitcoin Cash) or decreasing the block time (Litecoin) [32, 33].

Bitcoin has a block time of around 10 minutes and probabilistic block finality. A transaction is often considered final after six confirmations[3] [34, 35].

2.2.2 Ethereum

Ethereum was introduced in 2014 to address Bitcoin's lack of a Turing-complete scripting language. Therefore Ethereum's biggest advantage is the possibilities to develop distributed applications using "smart contracts". This also lead to a high developer base working on use cases.

However, as it uses PoW, Ethereum has the same efficiency issues as Bitcoin. However, it is planning to change the consensus mechanism to Proof of Stake [16].

Due to the smart contract capabilities, Ethereum allows the creation of additional cryptocurrencies. ERC20 is a definition of rules and functionalities which can be implemented to create a token on Ethereum. An example of this is the Binance coin, developed by and used for the cryptocurrency trading platform Binance [3, 36].

Ethereum has a block time of around 15 seconds and probabilistic block finality. The white paper states 7 confirmations as a reasonable time to wait [3, 37, 38].

2.2.3 Stellar

Stellar focuses on being a cross-boarder payment solution. The main vision is to provide an efficient and secure payment system to developing countries.

Stellar uses its own consensus mechanism called Stellar Consensus Protocol. This allows for a high transaction throughput (3000+ transaction per second) and low transaction fees while keeping a high degree of decentralization. Stellar has smart contract capabilities but no touring complete language, which makes it useful for simple applications only.

Stellar has block finality due to their SCP consensus mechanism and a block time of 2-5 seconds [28].

2.2.4 EOS

EOS calls itself the "infrastructure for decentralized applications" [39] and therefore has a similar vision to Ethereum. EOS solves the issues of PoW by using a combination

[3]Six blocks built on top of the respective block

of dPoS and BFT. Thus, EOS has a high transaction throughput, no transaction costs for simple token transfers and consumes less energy. The main criticism of EOS is their centralization introduced by dPoS with only 21 block creators and the possibility to bribe voters[40].

EOS has block finality and a block time of 0,5 seconds [5].

2.2.5 IOTA

IOTA was built to enable a machine to machine economy in the internet of things. In theory, IOTA is not a blockchain, but its properties are very similar. It uses an acyclic directed graph called "tangle" instead of blocks.

Instead of miners, which externally verify transactions and therefore need to be rewarded, confirmation happens directly within the network by the transaction issuers. Every transaction references two past transactions, which are randomly chosen by the Markov Chain Monte Carlo algorithm. Thus every transaction attest the conformity to the protocol rules of those two transactions building a graph of transactions. A transaction is considered confirmed if it is directly or indirectly referenced by all subsequent transactions.

As every transaction needs to confirm two previous transactions, the amount of transactions IOTA can handle increases with the amount of users. This also implies that as long as there are only few users, it is easy to attack the network. This is why IOTA currently uses a "coordinator" to verify each transactions. As this coordinator can allow double spends or censor transactions, IOTA is very centralized until the coordinator is removed [41].

The main advantage of IOTA is the lack of transaction fees and its scalability. However, it does not support smart contracts [42].

IOTA currently has a fixed block time of one minute and complete block finality. After removing the coordinator, its finality would become probabilistic [43].

2.2.6 Hyperledger Sawtooth

Hyperledger Sawtooth was developed by Intel as a tool to build and deploy private blockchains. A strong focus on modularity allows developers to choose parts like the consensus mechanism according to the concrete use case. By default it uses Proof-of-Elapsed-Time (PoET) as a consensus mechanism. Another strength of Sawtooth is a multi-language support to write smart contracts [29].

Sawtooth has block finality. The block time is dependant on the randomized waiting time. However, a target waiting time can be set with a default of 20 seconds [29].

2.2.7 Multichain

Multichain is another platform for building blockchains. Its advantages are a simple deployment of private blockchains and the configurations options *e.g.,* consensus, block time and permissions for nodes. It also provides a "Bitcoin-like" blockchain preconfigured, using same API as Bitcoin Core [8].

Multichain is not a smart contract platform but supports some limited functionality through "Smart Filters" [44]. As a PoA blockchain, Multichain has block finality and a default block time of 15 seconds [45].

2.2.8 Comparison

The following table provides a summary over the blockchains used in the prototype and some of their properties. Values of one in the "Confirmation After" signify block finality. Else a best practice estimate is provided [46].

Blockchain	Type	Consensus	Finality	Blocktime (s)	Confirmation After
Bitcoin	Public	PoW	No	600	6 blocks
Ethereum	Public	PoW	No	15	7 blocks
Stellar	Public	SCP	Yes	5	1 block
EOS	Public	dPoS	Yes	0.5	1 block
IOTA	Public	IOTA	Yes	60	1 block
Hyperledger	Private	PoET	Yes	20	1 block
Multichain	Private	PoA	Yes	15	1 block

Table 2.1: Blockchain Overview

2.3 Interoperability Use Cases

This section explains the different use cases of blockchain interoperability. Asset portability, atomic swaps, cross-chain oracles, asset encumbrance and cross-chain contracts are five categories of important applications which are not possible without interoperability. [47].

2.3.1 Asset portability

Asset portability enables a user to transfer his own tokens from one blockchain to another. This connection should be two-way, also allowing to transfer the asset back at any time [47].

2.3.2 Atomic Swap

Atomic swaps[4] allow two parties to trade two tokens from different blockchains with each other. Both parties are required to have an account or an address on each blockchain. The trades are then executed simultaneously on both blockchains. It needs to be guaranteed that either both transfers happen or neither of them happen. This property is called "atomic". Atomic swaps are often used in the context of decentralized exchanges [47].

2.3.3 Cross-chain oracles

An oracle provides a blockchain with outside data. In the special case of a cross-chain oracle, the data originates from another blockchain. For example, an oracle extracts information about a transaction from an external blockchain and subsequently triggers an action on a smart contract [47, 48].

2.3.4 Asset encumbrance

Asset encumbrance is the ability to lock up an asset on one blockchain and unlocks it if a certain condition on another chain is met. For example, this could be used for security deposits [47].

2.3.5 Cross-chain contracts

Cross-chain contracts enable seamless dependent actions between blockchains with multiple dependencies in more then one chain. For example, a smart contract could pay out dividends on one chain by checking if this user is registered as a stakeholder on another chain. This is the most general use case and thus would enable some of the other use cases as well [47, 49].

2.4 Interoperability Techniques

There are three main techniques to enable the aforementioned use cases: notary schemes, sidechains and hash-locking [47].

[4] Atomic Swaps are also known as payment-for-payment or payment-for-delivery trades.

2.4.1 Notary Schemes

Notary schemes use a trusted entity as an intermediary between two blockchains. The role of the notary is therefore to verify that an event took place at one blockchain and to feed this information to a second blockchain.

The main advantage of the notary scheme is the simplicity, as no changes are required in the underlying blockchains. The downside is the trust needed in the notary. As one possible solution, both involved parties could chose a set of notaries they trust. The notaries output could then be created by using a consensus algorithms, such as BFT. Thus, there would be no need to trust every individual notary, but only two thirds of the group [47].

2.4.2 Sidechains

A sidechain is a blockchain which has the capability to validate and gather information about the state of other blockchains[5].

Simplified Payment Verification (explained in Section 2.1.2) is used to attest that a blockchain is in a given state. Although the data needs to be externally fed from one blockchain to the other, this process does not require trust due to the cryptographic properties of blockchains. It would be easy to create proof that the headers were tampered with.

Sidechains allow for a range of use cases by being able to access the state from other blockchains. However, smart contract capabilities are required to create a sidechain. Furthermore, to achieve full interoperability, every blockchain would need a sidechain which in turn needs to support every other blockchain. Thus, the maintenance of this growing system would become a major issue [47].

Pegged Sidechains

Pegged sidechains are a specific implementation of sidechains to trade asset between blockchains. It works by locking up assets on the target chain and creating a cryptographic proof that it is locked up. This proof can then be relayed and verified by the sidechain.

This approach allows to make transaction from one blockchain to another, but only works if the blockchain supports locking up assets. Furthermore there need to be waiting periods to make sure the action of locking up the asset is final. Furthermore, to enable full asset portability it would be necessary to be able to create and destroy coins on the sidechain. However, most of today's major blockchains do not have this feature. "Federated Pegged Sidechains" address this problem with a notary who locks the asset and provides a confirmation. [47].

[5]More precisely, the sidechain is only the part of the blockchain which enables this functionality. However in the context of interoperability the whole blockchain is often called a sidechain, because it was specifically created for the sidechain functionality.

2.4.3 Hash-locking

Hash-locking triggers an action on two blockchains simultaneously in an atomic manner *i.e.,* with both actions or none of them happening. Hash-locking can be used to make trades between two or more parties without using an intermediary. As hash-locking schemes can be chained after each other, can enable trades even if there is no direct connection between the trading parties. This technique is used by most decentralized exchanges today.

For example, two parties A and B want to make a trade but have their assets on different blockchains. They can use the following scheme to enable the trade:

1. A generates a secret s and computes the hash $h(s)$, which is sent to B.
2. Both A and B lock their assets into a smart contract, which can verify if an inputted s belongs to $h(s)$.
3. A has to provide s to the contract holding B's funds within $2 * X$ seconds to trigger a transfer to A, otherwise the asset goes back to B.
4. B has to provide s to the contract holding A's funds within X seconds to trigger a transfer to B, otherwise the asset goes back to A.
5. A reveals the secret within X seconds. Thus B learns the secret and can claim the asset from A.

The scheme works and is atomic as long as both parties act according to their own financial interest. One obvious drawback of this scheme is that the blockchain needs to support this special type of smart contracts, called Hash-TimeLock Contracts (HTLC). Furthermore the waiting period X could be exploited by speculating on falling or increasing prices and thus only revealing the secret if the price fluctuation favours the party.

Hash locking can be used only for the use cases atomic swap and the similar case of asset encumbrance. Cross-chain oracle are not possible because this would require a passive (read) operation where hash-locking needs active participation from both sides. Asset portability is also not possible (without an exchange as intermediary) because hash-locking preserves the assets at both chains [47].

2.4.4 Overview

Table 2.2 summarizes which techniques enable which use case. Both notary schemes and sidechains could potentially enable all use cases. However sidechains are hard to implement because they require smart contract and locking capabilities. Furthermore, every blockchain would need to implement a sidechain for every other blockchain. In comparison, the notary scheme is easy to implement but requires trust in the notary [47].

Use Case	**Notaries**	**Sidechain/Relay**	**Hash-locking**
Asset portability	Yes	Yes	No
Atomic Swap	Yes	Yes	Yes
Cross-chain oracles	Yes	Yes	Not directly
Asset encumbrance	Yes	Yes	Yes
Cross-chain contracts	Yes	Yes	No

Table 2.2: Interoperability schemes overview

Chapter 3

Related Work

The rapid increase in the number of different blockchains raised awareness about the importance of interoperability. Thus, several projects tackling this problem were created. This chapter presents an overview of the current interoperability projects. Furthermore, it provides a classification according to the techniques described in section 2.4. Finally, a discussion summarizes the current state and main challenges of interoperability projects.

3.1 Notary Schemes

3.1.1 Herdius

In its essence, Herdius is a decentralized exchange platform which focuses on the common linking point among all blockchains, the private keys. Thus, Herdius enables exchanges between different blockchains through sharing them. To decentralize this process, Herdius encrypts the users private keys (*e.g.,* an Ethereum private key) with a private key generated within Herdius. This key is then sliced and distributed to notaries called "assembler nodes", using a threshold multisignature mechanism.

Multiple assembler nodes are able to sign a transaction on behalf of the user by combining their parts of the private key. To add another layer of security, no assembler node can completely decrypt the native private key. Instead, the transaction is signed by making use of homomorphic cryptography computations. By using this structure, Herdius is able to decentralize the notary-scheme [50, 51].

3.2 Sidechain/Relay

3.2.1 BTC-Relay

BTC-Relay enables one way interoperability between Bitcoin and Ethereum. It is a sidechain for Bitcoin in the Ethereum blockchain. Thus, the Ethereum contract uses

SPV to verify and process transactions from the Bitcoin blockchain [52].

3.2.2 Aion

Aion is built as a layer on top of blockchains with the vision of full interoperability. It provides a communication protocol which a blockchain can implement to become part of the Aion network. However, the following requirements need to be met to be able to support the protocol:

- Be decentralized in some fashion and support procedures commonly found in blockchains such as atomic broadcast and transactions.
- Be able to recognize transactions between blockchains as distinct from regular transactions.
- Be aware of the consensus protocol used by the bridge and store a transaction deemed valid.
- Implement locktime or a similar feature that allows tokens to be held by the network for a period of time.

Although these are common requirements for the sidechain approach, most of today's major blockchains do not fulfill them. As a workaround for such blockchains, Aion is developing "token bridges". Those are blockchain specific solutions to support the requirements above even though they are not implemented natively. For example, they developed a pegged sidechains in Ethereum (see Section 2.4.2) [53, 54].

3.2.3 Aelf

Aelf describe themselves as a "branched eco-system", emphasizing their focus on modularity. The Aelf main chain is the backbone for multiple sidechains, connecting it to different blockchains.

The idea behind this modularity is that each sidechain will be dedicated to one type of transaction. Thus, each sidechain will only solve one type of business problem or use case. This means that the complexity of the overall system can be decreased and the modules can be tailored to fit the specific needs. [1, 55].

3.2.4 Cosmos

Cosmos Network works on interoperability since 2014, when releasing its own consensus mechanism called *Tendermint*, a dPoS approach. Cosmos uses their "Inter Blockchain Communication Protocol". This protocol consists of sidechains using SPV to validate transaction from other blockchains *e.g.,* Ethereum.

Furthermore, Cosmos is providing tools for users to create their own blockchain with *Tendermint* as consensus. Hence, interoperability between the resulting blockchains can be enabled more directly compared to using SPV. Additional interoperability is achieved by using the Ethereum Virtual Machine and Ethereums programming language Solidity for smart contracts [12, 56] .

3.2.5 Polkadot

Polkadot has a very similar approach to Cosmos and is even using a slightly adapted version of their consensus mechanism *Tendermint*.

The main difference lies in their governance approach. Cosmos does not prevent any blockchains from joining, as long as they support their protocol. In contrast, to attach a blockchain to the Polkadot network, a large number of their cryptocurrency "Dots" must be staked. Therefore, Polkadot takes more control over their partner chains by being able to punish malicious behaviour [57].

3.2.6 Ark

Ark describes itself as an open source framework to create blockchains with one click. The ARK Platform provides a so called "Smart Bridge". The Smart Bridge is a sidechains in the Ark blockchain. Instead of supporting different blockchains by implementing sidechains compatible with each other blockchain, Ark only provides a generic protocol, thus leaving it up to the blockchain to implement it. However, this means that either a blockchain needs to be created by using the tools from Ark or it has to change its implementation to be able to send data to the Ark blockchain.

A second interoperability feature is their planned integration of cryptocurrency exchanges (*e.g.,* Shapeshift) to convert between different cryptocurrencies automatically, thus additionally implementing a notary scheme [58].

3.2.7 VirtualChain

VirtualChain is a layer sitting on top of blockchains. This layer enhances the functionality of the underlying blockchain without requiring changes to it. The nodes of the underlying blockchain are unaware of the top layer, because the new operations are defined in the virtual blockchain layer and encoded in the metadata of blockchain transactions, *e.g.,* using smart contracts. Blockchain nodes do see the raw transactions, but the logic for processing the virtual blockchain operations only exists at the virtual blockchain level. Interoperability is therefore enabled by holding information about multiple blockchains in the VirtualChain. The information is inputted using sidechains [59, 60].

3.2.8 ICON

ICON is "connecting crypto to real world". As part of the "Blockchain Interoperability Alliance", they are pursuing and supporting research in the area of blockchain interoperability and are trying to establish industry standards for blockchain architecture and communication. In practice, Nexus is their name for their hub which connects to SPV clients to enable interoperability [61, 62].

3.3 Hash Locking

3.3.1 Wanchain

As one of the more mature projects in this section, Wanchain has developed a hash lock based solution to enable token transfers between their own chain and Ethereum. Furthermore, they are working on supporting other blockchains.

Instead of the more common Hashed Time Lock contract they use what they call "Locked Account" scheme. This scheme is used to create an account which will be locked while the two transfers are happening. It manages that by splitting up the private key of the account and distributing it to multiple nodes in the Wanchain. The nodes can only jointly make transfer from the locked accounts and will make sure that the transactions are atomic. Another interoperability feature is the ability to run Ethereums solidity contracts on the Wanchain [63].

3.3.2 Interledger

Interledger calls themself "the protocol for connecting ledgers" and their protocol "ILP" (Interledger Protocol), a reference to their inspiration, the IP protocol [64].

Different from many projects above, Interledger is not a blockchain, token or central service. Instead, Interledger is a network of connectors, connecting hashed-timelock contracts in different blockchains to enable decentralized exchanges [65].

If one of the two ledgers does not support hashed-timelock contracts, time-based multisignature is used. Thus, locked funds can only be withdrawn if both parties sign a withdrawal claim. To make a transaction, the sender first sends a message including the hashlock and timeout to the receiver. If the receiver presents the secret in time, the sender provides him a signed claim, which the recipient can use to withdraw funds. Because the sender could decide not to sign the claim, the recipient must trust the sender for the total amount that is not yet covered by payment channel claims [66].

Furthermore, if no multisignature accounts are available, so called "trustlines" can be used. In simple terms, trustlines are balance sheets which keep track of the debt between two users. This balance is resolved off- or on-chain manually. Obviously, this scheme requires full trust in the counterpart [65, 67, 68].

3.4 Other projects

3.4.1 Crowd Machine

Crowd Machine focuses on the interoperability in terms of smart contracts. Thus, they provide an engine, which can auto-generate smart contract for multiple blockchains. Hence, developers can implement the application logic without knowing the underlying smart contract technology [69].

3.4.2 Origintrail.io

Origintrail is a blockchain specifically build for supply chain tracking. It establishes interoperablility by using the "GS1-standards" for master data (descriptive attributes for products), transaction data (related to business relations), and visibility data (related to tracing and tracking). It is therefore not a general purpose solution to enable interoperability, but enhances interoperability in the supply chain context [70].

3.5 Discussion

The works presented in this section show that there is a number of efforts addressing interoperability. Although they differ in the architectures, most of them follow the sidechain approach. As described in Section 2.4, this is generally the most promising approach regarding the spectrum of use cases it can cover. However, it often requires changes in the target blockchains, as most blockchains do not support locking or destroying tokens.

To overcome this restrictions some projects combine different techniques. For example, Interledger uses sidechains as well as notary-scheme components [67]. Most of the projects focus on the interoperability in a separate ecosystem that they are building instead of enabling interoperability between existing blockchains. One notable exception is BTC-Relay, which establishes an existing blockchain as a sidechain of another, namely Ethereum as a sidechain of Bitcoin [52].

As most of the projects described above are still in their infancy, it is likely that their approach and technique will adapt as interoperability research progresses [68].

Chapter 4

Interoperability API Solution

This section introduces a solution for storing and retrieving data on seven different blockchains. It starts by defining the goals and scope of the solution. Furthermore, it presents an in-depth explanation of the implementation and the reasoning behind the architecture decisions.

4.1 Objective

The objective of the prototype is a simple interface to interact with different blockchains. It allows a user to store, retrieve, and migrate data in the form of a string.

In the context of this solution, the "user" might be a software developer of a blockchain application. Thus, this interface allows a developer to support a variety of blockchains without having to know the specific implementations. Additionally, this allows further abstractions by letting an algorithm decide on the blockchain to use depending on certain parameters, for example transaction costs or performance.

As a prototype, a special emphasis was placed on **flexibility**, **ease of use**, and **modularity**. **Flexibility** is given by allowing the user to store a string in the blockchain, which could represent any arbitrary data, *e.g.,* an id or a hash. **Ease of use** is achieved by using docker whenever possible. Docker is a software that isolates applications using virtualization with containers [71]. This allows a user to start a node with the simple command `docker-compose -f [name].yaml up`. Furthermore, this minimizes compatibility problems as the nodes run in an isolated and replicable environment. To further improve usability, the API is the same for all blockchains and only requires minimal input from the user. **Modularity** is provided by implementing the adapter of each blockchain with a common interface. This simplifies adding or removing adapters to additional blockchains. Furthermore, a public testnet was used whenever possible, making it easy to check completed transactions using a block explorer website, *e.g.,* https://testnet.steexp.com/ for the Stellar blockchain. A local testnet was used if no stable public testnet was available or if it interfered with ease of use *i.e.,* if the process to get test tokens is cumbersome.

4.2 Solution Design

The proposed solution relies on a notary scheme to interact with multiple blockchains. Thus, it is intended to be hosted on a server, together with the corresponding blockchain application. The notary scheme was used because it is a feasible and straightforward manner to manage data stored on different blockchains.

The solution consists of three parts: the API, the blockchain adapters, and a database. Figure 4.1 presents an overview of the three parts and the data flow between them. The first layer is the API, which consists of the exposed python functions `store`, `retrieve` and `migrate`. The API is responsible for taking the user input and initiating the request.

Second, for each supported blockchain, an adapter was implemented. They convert the user input into transactions[1], which are subsequently transmitted to the blockchains nodes. The nodes forward the transactions to the whole network, where they will be processed by miners.

Third, there is a database storing the necessary credentials for the transactions. Furthermore, it stores the transaction hash after a successful transaction has been included in the blockchain. This hash can be later used to retrieve the stored data. It is worth mentioning that the data is only stored in the blockchain and not replicated in the database.

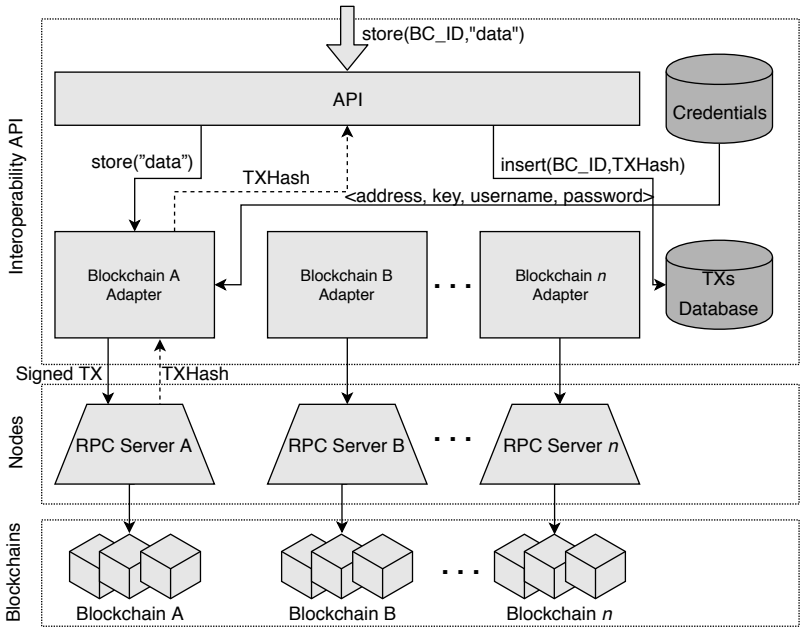

Figure 4.1: General architecture

4.2.1 Application Programming Interface (API)

The solution has three exposed functions. Their implementation is presented in Listing 4.1 and described in the following paragraphs.

[1]In the case of `retrieve` there are requests to read data from the blockchain instead of transactions

The `store(text, blockchain)` function receives a string and a blockchain ID as input. It then stores the string on the defined blockchain and returns the transaction hash.

The `retrieve(transaction_hash)` function takes a transaction hash as a parameter and returns the string previously stored on the blockchain. The corresponding blockchain is automatically recognized by a look up in the database. Therefore, the blockchain ID does not need to be provided.

The `migrate(transaction_hash, blockchain)` function retrieves a stored string from one blockchain, and stores it in the specified blockchain. Thus, the parameters are the transaction hash of the origin and the name of the target blockchain.

```
1  def store(text, blockchain):
2      adapter = Adapter[blockchain]
3      transaction_hash = adapter.store(text)
4      return transaction_hash
5
6  def retrieve(transaction_hash):
7      blockchain = database.find_blockchain(transaction_hash)
8      adapter = Adapter[blockchain]
9      text = adapter.retrieve(transaction_hash)
10     return text
11
12 def migrate(transaction_hash, blockchain):
13     value = retrieve(transaction_hash)
14     new_hash = store(value, blockchain)
15     return new_hash
```

Listing 4.1: Exposed functions

4.2.2 Blockchain Adapters and Nodes

Table 4.1 provides an overview of the different adapters. Seven adapters to different blockchains were implemented. Specifics about each blockchain can be found in Section 2.2. Additionally, a PostgreSQL adapter was implemented as a reference and to allow to store the data in a traditional database instead of a blockchain.

Apart from the blockchain and library name, the table also shows information about the node type, connection, and network type (*e.g.,* local or public testnet). For example, the Bitcoin implementation connects to a full node with access to the public Bitcoin testnet and uses the RPC protocol to communicate with the node.

Adapter Implementation

Internally, the store function first constructs a raw transaction using the provided string. This transaction is subsequently signed using the private key and sent using the RPC or HTTP protocol. After receiving the transaction hash, the function “confirmation check”

Blockchain	Library Name	Node type	Connection	Network
Bitcoin	python-bitcoinrpc	Full Node	RPC	Public Testnet
Ethereum	web3.py	Full Node	RPC	Local Testnet
Stellar	py-stellar-base	Remote Full Node	HTTP	Public SDF Testnet
IOTA	PyOTA	Remote Full Node	RPC	Public Testnet
EOS	eosjs_python	Remote Full Node	RPC	Public Jungle Testnet
Hyperledger	sawtooth_sdk	Full Node	HTTP	Local Testnet
Multichain	python-bitcoinrpc	Full Node	RPC	Local Testnet
PostgreSQL	psycopg2	PostgreSQL 9.6.8	Postgres	Local Postgres

Table 4.1: Blockchain Adapter Implementations

validates the existence of the transaction after waiting for a certain period (as discussed in the next section). Thus, it is confirmed that the block was included and finality is given. If the confirmation was successful, the hash is saved to a SQLite database, referencing the blockchain ID and including a timestamp.

The retrieve function requires the transaction hash as a parameter. The raw transaction is retrieved from the blockchain using RPC or HTTP. Afterwards, it extracts the string from the obtained raw transaction.

Listing 4.2 shows part of this implementation. The parameter "cls" reflects the chosen blockchain in the form of an adapter class.

```
1  // adapter.py
2  def store(cls, text):
3          transaction = cls.create_transaction(text)
4          signed_transaction = cls.sign_transaction(transaction)
5          transaction_hash = cls.send_raw_transaction(signed_transaction)
6          if(WAIT_FOR_CONFIRMATION):
7              if(cls.confirmation_check(transaction_hash)):
8                  cls.add_transaction_to_database(transaction_hash)
9                  return transaction_hash
10             else:
11                 raise LookupError(
12                     'Transaction not confirmed and not added to DB')
13         else:
14             cls.add_transaction_to_database(transaction_hash)
15             return transaction_hash
16
17 def retrieve(cls, transaction_hash):
18         transaction = cls.get_transaction(transaction_hash)
19         data = cls.extract_data(transaction)
20         return cls.to_text(data)
```

Listing 4.2: Adapter Implementation

Confirmation Times

Sending a transaction to a blockchain is generally no guarantee that the value is stored on the ledger. On the one hand the transaction could be refused by the network *e.g.,* because of a low transaction fee or if the validation failed. On the other hand, even if it was included, block finality needs to be ensured. Skipping this check could lead to transactions located on a fork which is abandoned later. However, finality depends on the consensus mechanism implemented in the blockchain.

In the prototype this issue is solved by using a waiting time before writing the transaction to the database. If the transaction is not found after this time, it is considered invalid and will be discarded. Considering table 4.2, Bitcoin and Ethereum need to have a longer waiting time of 3600 seconds respectively 105 seconds to have a high probability of finality.

Even if block finality is given, transactions do not happen instantly. Especially with PoS consensus, there can be multiple rounds of communication between the nodes until two thirds of them agree on one solution. As this communication takes some time, a minimum waiting time of 20 seconds was implemented [72].

Blockchain	Type	Consensus	Finality	Blocktime (s)	Confirmation After
Bitcoin	Public	PoW	No	600	6 blocks
Ethereum	Public	PoW	No	15	7 blocks
Stellar	Public	SCP	Yes	5	1 block
EOS	Public	dPoS	Yes	0.5	1 block
IOTA	Public	IOTA	Yes	60	1 block
Hyperledger	Private	PoET	Yes	20	1 block
Multichain	Private	PoA	Yes	15	1 block

Table 4.2: Blockchain Overview (extract from section 2.2.8)

The prototype was tested using the blockchain's public testnets or a local private networks. Even though the consensus mechanism of those networks often differ from the mainnet, the waiting times were set to conform with the confirmation times on the mainnets.

4.2.3 Database

The database is responsible for storing a list of supported blockchains, credentials for the transactions and the transaction hashes. For this prototype, a SQLite[2] database and the Python library *sqlite*3 was used. SQLite was chosen because of the simplicity of storing the whole database directly in a single file, without the need of a running a dedicated server [73]. The next paragraphs describe the tables which compose the database.

[2] `www.sqlite.org`

Blockchains Table

The blockchain table has two columns to match an *id* to the name of the blockchain *e.g.,* `id=1, name = 'Ethereum'`. This is used to link the blockchains to the transactions and credentials.

Credentials

The credentials table has six columns. The *id* is the primary key and can be used to map the identity of users to their credentials. The *blockchain_id* connects the credentials to the corresponding blockchain. Furthermore, *address* stores the public key and *key* the private key of the blockchain account. The *user* and *password* provide the credentials for the RPC or HTTP client. The amount of credentials needed depends on the blockchain. For example Stellar does not need *user* and *password*. IOTA does not even need *key*, as zero-value transactions do not require a sender and thus no signature.

Transactions

After a transaction is completed, its hash is stored in the column *hash*. It is linked to a blockchain by *blockchain_id*. Furthermore there is the *timestamp* in the column *issued_at*.

4.2.4 Environment and Dependencies

The prototype is based on existing code by Patrick Widmer[3]. The prototype was developed using python. Python was chosen because its properties allow for rapid prototyping while keeping the code clean and comprehensible. Furthermore there is extensive support for libraries used by this project *e.g.,* for cryptographic operation, HTTP and RPC [74]. In terms of compatibility, the prototype was tested on both Linux Ubuntu 16 and MacOS Mojave using Python 3.6.6. Python's built-in virtual environment manager Venv was used to prevent environment issues.

[3]The initial version can be found at `https://gitlab.ifi.uzh.ch/scheid/bcio`

Chapter 5

Evaluation and Discussion

This chapter analyses the presented prototype with regard to performance, security and data transfer size. It starts by analyzing the performance of the prototype using a sample of 1000 transactions per blockchain. It continues by discussing the major security issues. Finally, it explains the data size limitations of the current implementation.

5.1 Performance Analysis

Listing 5.1 presents the adaptation of the store function for the performance analysis. It measures the time until a transaction is sent, the hash retrieved and the transaction information stored in the database. Note that the finality confirmation of the original store function was removed in order not to interfere with the measurements.

```
1 def store(cls, text):
2         start = int(round(time.time() * 1000))
3         transaction = cls.create_transaction(text)
4         signed_transaction = cls.sign_transaction(transaction)
5         transaction_hash = cls.send_raw_transaction(signed_transaction)
6         cls.add_transaction_to_database(transaction_hash)
7         cls.save_measurement(int(round(time.time() * 1000)) - start)
8         return transaction_hash
```

Listing 5.1: Adapted store function to measure performance

The measurements were performed on a Macbook Pro[1] with Python 3.6.6. At total, 1000 measurements were taken for each blockchain except Bitcoin[2].

As some remote nodes impose restrictions on how many transactions can be sent over a certain time, the measurements were completed in batches of 25 transactions each.

[1] Macbook Pro 2017, Dual Core i5 @ 2.3 GHz, 8GB RAM, MacOS Mojave 10.14.1

[2] In case of Bitcoin, only 100 samples were taken because of the limit of 25 unconfirmed transaction on the Bitcoin RPC server.

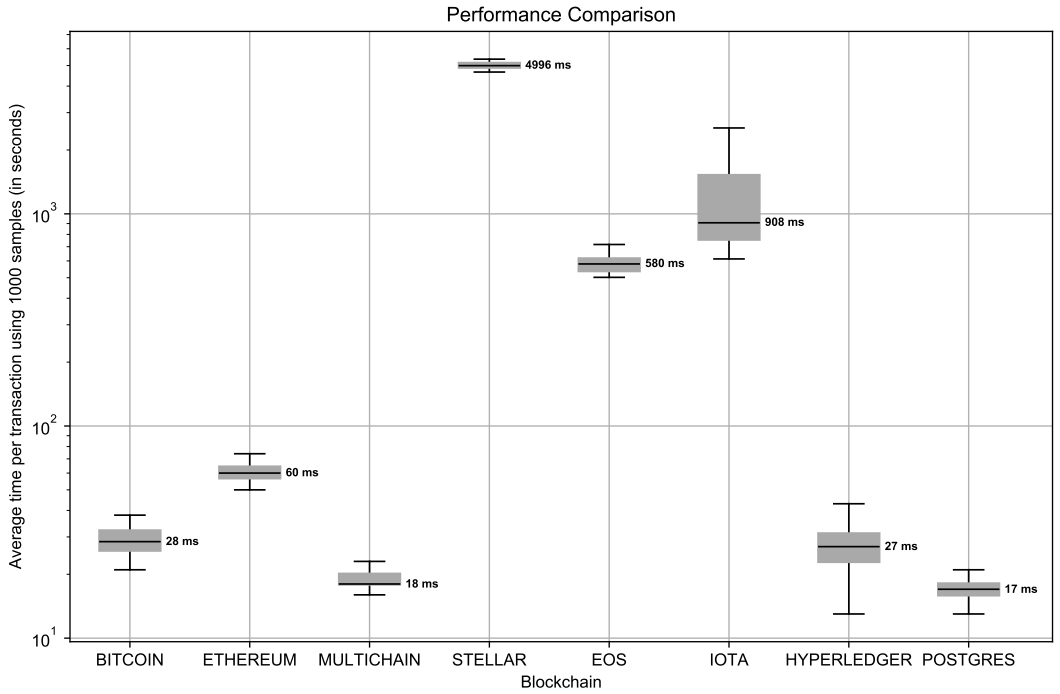

Figure 5.1: Performance measurements

Figure 5.1 presents the outcome of the measurements. There is a big speed difference between using local nodes and remote nodes. Remote nodes were used by Stellar, IOTA and EOS as listed in table 4.1. The Bitcoin client using a full local node with a public testnet was even faster than other blockchains which are using a local node with a private testnet.

It also has to be noted that with some adapters *e.g.,* Hyperledger, multiple transactions could be put into one batch. Thus, only one batch would need to be transmitted for multiple transactions which would result in a higher throughput.

5.2 Security Analysis

In the following sections some relevant security aspects are discussed. It starts by discussing the security issue of storing the private keys. It continues by considering the security of local and remote nodes. Finally, it explains the issue of centralization.

5.2.1 Private Key Management

A major concern is to secure the private keys from attackers. This would have two implications:

First, an attacker could spend the funds linked to the account. As this prototype is made for storing data on the blockchain, the funds would only need to cover the transaction

costs. Keeping only the necessary funds for the transactions would therefore mitigate this risk.

Second, an attacker could mess with the data by storing some arbitrary data which would be hard to distinguish from the genuine transaction. Depending on the use case, this could be an issue to consider. In most use cases, the motive for such actions is questionable.

In this prototype, the private key is stored in plain text on the SQLite database. This means if an attacker would get access to the server it would be trivial to access the private keys. One idea to circumvent this problem is to encrypt the private key with a symmetric key which is based on a password set by the user, thus only allowing transaction to happen if the user temporarily decrypts the private key. Another possible solution consists of a multisignature transaction scheme where the other keys needed for the transactions are stored in another, more secure place.

5.2.2 Local and Remote Nodes

A further risk factor is the exposure of the RPC or HTTP port by the blockchain node. If the node additionally holds the private keys in a local wallet, transactions could be initiated remotely. However, all of the blockchain adapters in this prototype sign the transactions locally, meaning there is no need for the node to hold the private keys. Nevertheless, the ports could be used to transmit unrelated, malicious transactions and denial of service attacks targeting the node could be executed.

In cases where public remote nodes are used, the owner of those nodes could potentially block a transaction from being processed. Furthermore, availability could become an issue.

In conclusion, those security issues could be circumvented by using full local nodes and exposing the RPC or HTTP port only to a local network or an application running on the same machine.

5.2.3 Centralization

One of the main benefits of Blockchains is the removal of trust by making use of decentralization and cryptographic properties. Thus, it would be advantageous if this property holds for this solution as well.

However, using a notary scheme implies that a user needs to trust the notary *i.e.,* the host of the application. First of all, the notary has access to the private keys which makes him vulnerable to the issues discussed in 5.2.1. Furthermore, the notary controls the application and the node and therefore is able to arbitrarily alter the original transactions *e.g.,* change the sender or the data or censor certain transactions.

Generally, blockchain applications are never trust-free and there are always layers which require a certain amount of trust. Using this prototype, the first layer could be the

blockchain's core code and cryptographic properties. The second layer could be the RPC-server and the RPC client. The notary could be the third layer of trust. After that, many other layers requiring trust follow *e.g.,* the operating system of the end-user.

From this perspective, the notary scheme adds a layer which requires trust but is still only one out of many. Nevertheless, the amount of trust needed can be minimized by open-sourcing the application, conducting peer reviews or running it in a decentralized environment.

5.3 Data size analysis

Transaction focused blockchains *e.g.,* Bitcoin are not designed to store arbitrary data in transactions. Thus, there is a limit on the amount of data which can be included.

Table 5.1 summarizes how much data can be stored in the different blockchains used by prototype. Note that on some blockchains this restriction could be circumvented by using smart contracts.

Blockchain	**Maximum string size**
Bitcoin	80 bytes [75]
Ethereum	46 kbytes[3] [76]
Stellar	28 bytes [77]
EOS	256 bytes [78]
IOTA	1300 bytes[4] [79]
Hyperledger	20 bytes [80]
Multichain	80 bytes [75]

Table 5.1: Data size

Generally, it is not very cost-efficient to store large amounts of data on blockchains. Instead a (decentralized) database can be used to store the data. This would mean only a link to the data and its hash would need to be stored on the blockchain. Thus, the validity of the data can be verified at any time while the data is stored more efficiently.

[3]Limited only by block gas limit

[4]IOTA uses trytes instead of bytes. 2187 trytes = 1300 bytes according to https://laurencetennant.com/iota-tools/

Chapter 6

Conclusion and Future Work

Due to trade-offs in terms of features, privacy and efficiency, there will be multiple, co-existing blockchains in the future, each covering a subset of the possible use cases. Thus, interoperability is needed to link different blockchains by exchanging information and assets.

On the one hand, there are different projects which are tackling the connection between blockchains. Most of them are using sidechains, a technique to read and validate data from other blockchains. However, this approach depends on changes in the blockchain's core code. Thus, most of the projects are not able to provide interoperability between today's major blockchains.

On the other hand, progress needs to be made in terms of simplicity and standardization. Developers and users need access to simple and comprehensible interfaces to interact with the variety of blockchains without knowing the specific implementations. This thesis approaches this problem by providing a common interface for storing, retrieving and migrating data on seven different blockchains using a notary scheme.

Notary schemes use an intermediary to validate or execute blockchain transactions. As they do not depend on changes in the blockchain's implementations, they work for all current and future blockchains and are comparably easy to implement. However, centralization is an issue. In the case of the presented prototype, the notary must be trusted not to censor or alter transactions. Furthermore, the increased centralization makes the notary a target for attacks. Thus, the usage of the notary scheme weakens the promises of blockchains to be a decentralized and trust-free system. That said, no blockchain application is currently able to deliver a completely trust-free system to the end-user, as there is always some layer where trust is involved.

Generally, a practical way to enable interoperability with today's major blockchains is to combine the sidechain technique with decentralized forms of notary schemes. However, as blockchain interoperability is still in its infancy, there is a probability of upcoming new techniques to solve the issue and make blockchain technology more accessible for the masses.

6.1 Future work

There are different aspects, which could improve the usefulness of this prototype, apart from extending it to support more blockchains.

The security could be increased by using multisignature transactions. Thus, an attacker who gains access to the database could not steal the funds connected to the private keys. In terms of usability, error handling could be improved. For example, failed transaction could be automatically repeated at a later point and the user could be notified. Furthermore, transactions to smart contracts would allow on-chain processing of the transmitted data, enabling more use cases. Furthermore, considerations could be made to run this application in a decentralized and trust-less manner.

Bibliography

[1] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. 2008. URL: https://bitcoin.org/bitcoin.pdf (visited on 11/23/2018).

[2] CoinMarketCap. *Kryptowaehrung Marktkapitalisierungen | CoinMarketCap*. 2018. URL: https://coinmarketcap.com/de/ (visited on 11/23/2018).

[3] Vitalik Buterin. *A next generation smart contract & decentralized application platform*. 2014. URL: https://cryptorating.eu/whitepapers/Ethereum/Ethereum_white_paper.pdf (visited on 11/27/2018).

[4] Etherisc.com. *Etherisc Flight Delay Insurance*. 2017. URL: https://fdd.etherisc.com/ (visited on 11/23/2018).

[5] EOS.IO. *EOS.IO Technical White Paper v2*. 2018. URL: https://github.com/EOSIO/Documentation/blob/master/TechnicalWhitePaper.md (visited on 11/23/2018).

[6] Alonso M. Kurt. *Zero to Monero - First Edition*. 2018. URL: https://pdfs.semanticscholar.org/16a8/c0aa45bd09830b8c5115c2c1e441f177fc82.pdf (visited on 11/23/2018).

[7] NEO. *NEO Documentation*. 2014. URL: http://docs.neo.org/en-us/index.html (visited on 11/23/2018).

[8] Gideon Greenspan. *MultiChain-White-Paper.pdf*. 2015. URL: https://www.multichain.com/download/MultiChain-White-Paper.pdf (visited on 11/23/2018).

[9] Zibin Zheng et al. "Blockchain challenges and opportunities: a survey". *Int. J. Web and Grid Services* 14 (2008), p. 24. URL: https://www.henrylab.net/pubs/ijwgs_blockchain_survey.pdf.

[10] Grayblock. *Interoperability â The Holy Grail of Blockchain*. 2018. URL: https://medium.com/coinmonks/interoperability-the-holy-grail-of-blockchain-eb078e1a29cc (visited on 11/23/2018).

[11] Oxford Dictionaries. *Definition of interoperability in English by Oxford Dictionaries*. 2018. URL: https://en.oxforddictionaries.com/definition/interoperability (visited on 11/23/2018).

[12] Dave Kajpust. *Blockchain Interoperability: Cosmos vs. Polkadot*. 2018. URL: https://medium.com/@davekaj/blockchain-interoperability-cosmos-vs-polkadot-48097d54d2e2 (visited on 11/23/2018).

[13] Neocapita. *The Logical Components of Blockchain*. 2017. URL: https://medium.com/@neocapita/the-logical-components-of-blockchain-870d781a4a3a (visited on 11/23/2018).

[14] Bitcoin Wiki. *Lightweight node - Bitcoin Wiki*. 2018. URL: https://en.bitcoin.it/wiki/Lightweight_node (visited on 11/23/2018).

[15] Keerthi Nelaturu. *Blockchain Interoperability â Sidechains*. 2018. URL: https://medium.com/coinmonks/blockchain-interoperability-sidechains-e8204b8c2a10 (visited on 12/21/2018).

[16] Ethereum Foundation. *Ethereum Proof of Stake FAQ*. 2018. URL: https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQs (visited on 11/23/2018).

[17] J Fischer and A Lynch. "Impossibility of Distributed Consensuswith One Faulty Process". *J. ACM* 32 (1985), p. 74.

[18] Grayblock. *Blockchain Finality In IoT*. 2018. URL: https://medium.com/coinmonks/blockchain-finality-in-iot-79e466406133 (visited on 11/23/2018).

[19] Leslie Lamport, Robert Shostak, and Marshall Pease. "The Byzantine Generals Problem". *ACM Transactions on Programming Languages and Systems* 4.3 (1982), pp. 382–401. ISSN: 01640925. DOI: 10.1145/357172.357176. URL: http://portal.acm.org/citation.cfm?doid=357172.357176 (visited on 11/23/2018).

[20] Ryan Smith. *What is the Environmental Impact of Bitcoin Mining?* 2018. URL: https://coincentral.com/what-is-the-environmental-impact-of-bitcoin-mining/ (visited on 12/20/2018).

[21] Bitrazzi.com. *These 5 Bitcoin Mining Pools Account for 70% of Bitcoin Hashrate Distribution*. 2018. URL: https://bitrazzi.com/these-5-bitcoin-mining-pools-account-for-70-of-bitcoin-hashrate-distribution/ (visited on 11/23/2018).

[22] Sunny King and Scott Nadal. *PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake*. 2012. URL: https://peercoin.net/assets/paper/peercoin-paper.pdf (visited on 11/23/2018).

[23] Alexis Gauba. *Finality in Blockchain Consensus*. 2018. URL: https://medium.com/mechanism-labs/finality-in-blockchain-consensus-d1f83c120a9a (visited on 11/23/2018).

[24] Ray Shaan. *The Difference Between Traditional and Delegated Proof of Stake*. 2018. URL: https://hackernoon.com/the-difference-between-traditional-and-delegated-proof-of-stake-36a3e3f25f7d (visited on 11/23/2018).

[25] Georgios Konstantopoulos. *Understanding Blockchain Fundamentals, Part 3: Delegated Proof of Stake*. 2018. URL: https://medium.com/loom-network/understanding-blockchain-fundamentals-part-3-delegated-proof-of-stake-b385a6b92ef (visited on 12/04/2018).

[26] Dantheman. *DPOS Consensus Algorithm - The Missing White Paper*. 2017. URL: https://steemit.com/dpos/@dantheman/dpos-consensus-algorithm-this-missing-white-paper (visited on 11/23/2018).

[27] Z. Zheng et al. "An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends". *2017 IEEE International Congress on Big Data (BigData Congress)*. 2017, pp. 557–564. DOI: 10.1109/BigDataCongress.2017.85.

[28] David Mazieres. *The Stellar Consensus Protocol: A Federated Model for Internet-level Consensus*. 2015. URL: https://www.stellar.org/papers/stellar-consensus-protocol.pdf (visited on 11/23/2018).

[29] Intel Corporation. *PoET 1.0 Specification Sawtooth v1.0.5 documentation*. 2018. URL: https://sawtooth.hyperledger.org/docs/core/releases/1.0/architecture/poet.html (visited on 11/23/2018).

[30] Intel Corporation. *Intel SGX Homepage*. 2018. URL: https://software.intel.com/en-us/sgx (visited on 11/23/2018).

[31] POA Network. *Proof of Authority: consensus model with Identity at Stake*. 2017. URL: https://medium.com/poa-network/proof-of-authority-consensus-model-with-identity-at-stake-d5bd15463256 (visited on 11/23/2018).

[32] Litecoin Project. *Litecoin - Quelloffene, digitale P2P-WÃ¤hrung*. 2018. URL: https://litecoin.org/de/ (visited on 11/23/2018).

[33] Nathaniel Popper. "Some Bitcoin Backers Are Defecting to Create a Rival Currency". *The New York Times* (2018). ISSN: 0362-4331. URL: https://www.nytimes.com/2017/07/25/business/dealbook/bitcoin-cash-split.html (visited on 11/23/2018).

[34] Meni Rosenfeld. "Analysis of hashrate-based double-spending". *eprint arXiv:1402.2009* (2014), p. 14. URL: http://adsabs.harvard.edu/cgi-bin/bib_query?arXiv:1402.2009.

[35] Bitinforcharts. *Bitcoin Block Time chart*. 2018. URL: https://bitinfocharts.com/ (visited on 11/23/2018).

[36] Ethereum Foundation. *A standard interface for tokens*. 2018. URL: https://github.com/ethereum/EIPs (visited on 11/23/2018).

[37] Ethereum Foundation. *On Slow and Fast Block Times*. 2015. URL: https://blog.ethereum.org/2015/09/14/on-slow-and-fast-block-times/ (visited on 11/23/2018).

[38] etherchain.org. *Average block time of the Ethereum Network*. 2018. URL: https://www.etherchain.org/charts/blockTime (visited on 11/23/2018).

[39] EOS.IO. *eosio | Blockchain software architecture*. 2018. URL: https://eos.io/ (visited on 11/23/2018).

[40] Interchain Foundation. *Consensus Compare: Tendermint BFT vs. EOS dPoS*. 2017. URL: https://blog.cosmos.network/consensus-compare-tendermint-bft-vs-eos-dpos-46c5bca7204b (visited on 11/23/2018).

[41] Eric Wall. *IOTA is centralized*. 2017. URL: https://medium.com/@ercwl/iota-is-centralized-6289246e7b4d (visited on 11/23/2018).

[42] Serguei Popov. *The Tangle*. 2018. URL: https://assets.ctfassets.net/r1dr6vzfxhev/2t4uxvsIqk0EUau6g2sw0g/45eae33637ca92f85dd9f4a3a218e1ec/iota1_4_3.pdf (visited on 11/23/2018).

[43] iota.org. *Consensus on the Tangle - IOTA Docs*. 2018. URL: https://iota.org (visited on 11/23/2018).

[44] MultiChain. *Second MultiChain 2.0 preview release*. 2018. URL: https://www.multichain.com/blog/2018/01/second-multichain-2-0-preview-release/ (visited on 11/23/2018).

[45] MultiChain. *Getting Started with MultiChain*. 2018. URL: https://www.multichain.com/getting-started/ (visited on 11/23/2018).

[46] Vitalik Buterin. *How should I handle blockchain forks in my DApp?* 2016. URL: https://ethereum.stackexchange.com/questions/183/how-should-i-handle-blockchain-forks-in-my-dapp/203 (visited on 11/23/2018).

[47] Vitalik Buterin. *Chain Interoperability*. 2016. URL: https://static1.squarespace.com/static/55f73743e4b051cfcc0b02cf/t/5886800ecd0f68de303349b1/1485209617040/Chain+Interoperability.pdf (visited on 11/23/2018).

[48] BlockchainHub. *Blockchain Oracles*. 2018. URL: https://blockchainhub.net/blockchain-oracles/ (visited on 11/23/2018).

[49] Maurice Herlihy. "Atomic Cross-Chain Swaps". *arXiv:1801.09515 [cs]* Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing (2018). URL: http://arxiv.org/abs/1801.09515 (visited on 11/23/2018).

[50] Herdius. *Herdius_Whitepaper_1.1.pdf*. 2017. URL: https://herdius.com/whitepaper/Herdius_Whitepaper_1.1.pdf (visited on 11/23/2018).

[51] Balazs Deme. *What is Herdius?* 2018. URL: https://medium.com/herdius/what-is-herdius-3831a47cfb6 (visited on 11/23/2018).

[52] btcrelay.org. *BTC Relay*. 2017. URL: http://btcrelay.org/ (visited on 11/27/2018).

[53] Blockgeeks. *Most Comprehensive AION Blockchain Guide*. 2018. URL: https://blockgeeks.com/guides/aion-blockchain/ (visited on 11/27/2018).

[54] Matthew Spoke. *AION White Paper*. 2018. URL: https://aion.network/media/en-aion-network-technical-introduction.pdf (visited on 11/27/2018).

[55] aelf.io. *aelf - A Multi-Chain Parallel Computing*. 2018. URL: https://aelf.io/gridcn/aelf_whitepaper_EN.pdf?v=1.6 (visited on 11/27/2018).

[56] Jae Kwon and Ethan Buchman. *Cosmos | Cosmos Documentation*. 2018. URL: https://cosmos.network/docs/resources/whitepaper.html (visited on 11/27/2018).

[57] Dr Gavin Wood. *Polkadot: Vision for a heterogeneous multi-chain framework*. 2017. URL: https://polkadot.network/PolkaDotPaper.pdf (visited on 11/23/2018).

[58] The ARK Crew. *ARK Whitepaper*. 2018. URL: https://ark.io/Whitepaper.pdf (visited on 11/27/2018).

[59] Jude Nelson et al. *Extending Existing Blockchains with Virtualchain*. 2017. URL: https://www.zurich.ibm.com/dccl/papers/nelson_dccl.pdf (visited on 11/23/2018).

[60] Blockstack. *Library for scanning blockchains and running Blockstack state engines: blockstack/virtualchain*. 2018. URL: https://github.com/blockstack/virtualchain (visited on 11/28/2018).

[61] ICON Foundation. *ICON*. 2018. URL: https://icon.foundation (visited on 11/28/2018).

[62] ICON Foundation. *Icon - Hyperconnect the world*. 2018. URL: https://icon.foundation/resources/whitepaper/ICON-Whitepaper-EN-Draft.pdf (visited on 11/28/2018).

[63] Wanchain. *Wanchain-Whitepaper-EN-version.pdf*. 2018. URL: https://wanchain.org/files/Wanchain-Whitepaper-EN-version.pdf (visited on 11/28/2018).

[64] Interledger W3C Community Group. *Interledger*. 2018. URL: https://interledger.org/ (visited on 11/28/2018).

[65] Interledger W3C Community Group. *Interledger Architecture*. 2018. URL: https://interledger.org/rfcs/0001-interledger-architecture/ (visited on 11/28/2018).

[66] Bisade Asolo. *Multisignature Technology Explained*. 2017. URL: https://www.mycryptopedia.com/multisignature-technology-explained/ (visited on 11/28/2018).

[67] Stefan Thomas and Evan Schwartz. *A Protocol for Interledger Payments*. 2016. URL: https://interledger.org/interledger.pdf (visited on 11/23/2018).

[68] Evan Schwartz. *Simplifying Interledger: The Graveyard of Possible Protocol Features*. 2018. URL: https://medium.com/interledger-blog/simplifying-interledger-the-graveyard-of-possible-protocol-features-b35bf67439be (visited on 09/26/2018).

[69] Craig Sproule. *Whitepaper*. 2018. URL: https://www.crowdmachine.com/wp-content/uploads/2017/11/Crowd-Machine-Whitepaper.pdf (visited on 11/27/2018).

[70] Branimir Rakic et al. *First purpose built protocol for supply chains based on blockchain*. 2017. URL: https://origintrail.io/storage/documents/OriginTrail-White-Paper.pdf (visited on 11/23/2018).

[71] Docker Inc. *Docker*. 2018. URL: https://www.docker.com/index.html (visited on 11/28/2018).

[72] Andrei Grigorean. *Latency and finality in different cryptocurrencies*. 2018. URL: https://hackernoon.com/latency-and-finality-in-different-cryptocurrencies-a7182a06d07a (visited on 11/28/2018).

[73] SQLite. *SQLite Is Serverless*. 2018. URL: https://www.sqlite.org/serverless.html (visited on 11/28/2018).

[74] Harpal Boparai. *Why Python is the best programming language for start-ups and why our developers love it*. 2018. URL: https://www.netsolutions.com/insights/why-python-is-the-best-programming-language-for-start-ups-and-why-our-developers-love-it/ (visited on 11/28/2018).

[75] Massimo Bartoletti and Livio Pompianu. "An analysis of Bitcoin OP_RETURN metadata". *arXiv:1702.01024 [cs]* (2017). URL: http://arxiv.org/abs/1702.01024 (visited on 11/27/2018).

[76] Afri. *Is there a limit for transaction size?* 2018. URL: https://ethereum.stackexchange.com/questions/1106/is-there-a-limit-for-transaction-size (visited on 11/27/2018).

[77] Stellar.org. *Transactions*. 2018. URL: https://www.stellar.org/developers/guides/concepts/transactions.html (visited on 11/27/2018).

[78] EOS.IO. *What is the character limit of MEMO?* 2018. URL: https://github.com/EOSIO/eos/issues/4296 (visited on 11/27/2018).

[79] iota.org. *The Anatomy of a Transaction*. 2018. URL: https://iota.readme.io/v1.2.0/docs/the-anatomy-of-a-transaction (visited on 11/27/2018).

[80] Intel Corporation. *IntegerKey Transaction Family*. 2018. URL: https://sawtooth.hyperledger.org/docs/core/releases/1.0/transaction_family_specifications/integerkey_transaction_family.html#state (visited on 11/27/2018).

[81] Technopedia. *What is an Application Programming Interface (API)? - Definition from Techopedia*. 2018. URL: https://www.techopedia.com/definition/24407/application-programming-interface-api (visited on 01/02/2019).

[82] Sheenam Khuttan. *What is Cryptocurrency Mainnet?* 2018. URL: https://kryptomoney.com/cryptocrrency-mainnet-testnet-explained/ (visited on 11/28/2018).

[83] Ralph C Merkle. "Protocols for Public Key Cryptosystems". *IEEE Symposium on Security and Privacy* (1980), p. 13. DOI: 10.1109/SP.1980.10006. URL: http://www.merkle.com/papers/Protocols.pdf (visited on 11/23/2018).

[84] Technopedia. *What is a Remote Procedure Call (RPC)? - Definition from Techopedia*. URL: https://www.techopedia.com/definition/24022/remote-procedure-call-rpc (visited on 11/28/2018).

Abbreviations

API	Application Programming Interface
BFT	Byzantine Fault Tolerance
dBFT	Delegated Byzantine Fault Tolerance
dPoS	Delegated Proof-of-Stake
FLP	Fischer Lynch Paterson
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
PoA	Proof-of-Authority
PoET	Proof-of-Elapsed-Time
PoS	Proof-of-Stake
PoW	Proof-of-Work
RPC	Remote Procedure Call
SCP	Stellar Consensus Protocol
SGX	Software Guard Extensions
SQL	Structured Query Language
SPV	Simplified Payment Verification

Glossary

API An application programming interface is a set of protocols, routines, functions or commands that programmers use to develop software or facilitate interaction between distinct systems [81].

Block finality Finality means that a block can be considered valid and persisted *i.e.,* is not located on a fork [18].

Double spending Double-spending is an attack to spend the same single digital token more then once [1].

Fork State of a chain where there are two parallel chains being worked on [1].

Mainnet Mainnet is the live and active network of a blockchain [82].

Merkle tree A data tree structure, where every leaf node contains the hash of the data and every non-leaf contains the hash of the child nodes data [83].

Merkle root The top of the Merkle tree, consists of one hash[83].

Mining Creating new blocks in a proof of work blockchain [1].

Minting Creating new blocks in a proof of stake blockchain [16].

Multisignature Multisignature a technique where transactions require two or more signatures before they can be executed [66].

Remote Procedure Call (RPC) RPC is a network programming model or interprocess communication technique that is used for point-to-point communications between software applications [84].

Remote Full Nodes A full node which is publicly accessible and exposes RPC or HTTP endpoints. Often used for testing.

Testnet Blochchain which provides the same functionality as the mainnet but is intended for testing only and therefore the tokens do not hold any value [82].

List of Figures

4.1 General architecture 26

5.1 Performance measurements 32

List of Tables

2.1 Blockchain Overview 14

2.2 Interoperability schemes overview 18

4.1 Blockchain Adapter Implementations 28

4.2 Blockchain Overview (extract from section 2.2.8) 29

5.1 Data size 34

Appendix A

Installation Guidelines

A.1 Usage

```
from api import store, retrieve, migrate, Blockchain
tx_hash = store('Some Data', Blockchain.STELLAR)
data = retrieve('[Transaction_Hash]')
tx_hash = migrate('[Transaction_Hash]', Blockchain.ETHEREUM)
```

Listing A.1: Usage

Or use the CLI:

```
source venv/bin/activate
python3 cli.py
```

Listing A.2: Usage

A.2 Setup

Python 3.6.6 (on Linux: 3.6.5) was used for this project. It was tested on MacOS and Ubuntu 18.

A.2.1 Prerequisites

- Pip
- Docker and Docker Compose
- Sqlite3
- Venv

Instructions are available in the repository.

A.2.2 Install Dependencies

On Linux, the following packages need to be installed:

```
#Upgrade pip:
pip install --upgrade pip
#Install packages:
sudo apt-get install build-essential libssl-dev libffi-dev python3-dev
```

Listing A.3: Install packages

Then you can install the dependencies:

```
#Create environment:
python3 -m venv venv
#Activate environment:
source venv/bin/activate
#Install dependencies:
venv/bin/pip install -r requirements.txt
```

Listing A.4: Install dependencies

A.2.3 Database Setup

Calling the setup function of the database module will:

- drop credentials and transactions tables if they already exist
- create tables for storing credentials and transactions
- seed the credentials and transaction tabletable with credentials

```
#Install sqlite:
pip install sqlite
#Run the database setup:
import db.database
db.database.setup()
```

Listing A.5: Setup database

A.2.4 Setup for each blockchain

Instruction on how to setup the local nodes for the adapters can be found here: `https://github.com/timohe/bc-interop/blob/master/SETUP.md` and on the CD.

Appendix B

Contents of the CD

The CD contains the following folders:

Code Contains the source code of the prototype, including a SETUP.md and README.md to explain the usage. The content is equivalent to the online repository at `https://github.com/timohe/bc-interop/blob/master`

Code-Documentation Contains the documentation of the code and the setup instructions as a PDF. The content is equivalent to the .md files in the repository.

Figures Contains the figures used in the thesis as JPG-files.

Presentations Contains the midterm and the final presentation in Powerpoint (.pptx) format.

Thesis Contains the Latex files for the final report and the report as a PDF and PS.