Informatics

Community blockchain provisioning

Untersuchung der Eignung von XMPP für permissionierte private Blockchains

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Tomáš Šedivý, BSc
Matrikelnummer 1128710

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: A.o. Univ. Prof. Dr. Dipl.-Ing. Eva Maria Kühn
Mitwirkung: Dr. Dipl.-Ing. Gerson Joskowicz

Wien, 15. April 2021

Tomáš Šedivý   Eva Maria Kühn

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Informatics

Community blockchain provisioning

Studying the suitability of XMPP for permissioned private blockchain systems

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Tomáš Šedivý, BSc
Registration Number 1128710

to the Faculty of Informatics

at the TU Wien

Advisor: A.o. Univ. Prof. Dr. Dipl.-Ing. Eva Maria Kühn
Assistance: Dr. Dipl.-Ing. Gerson Joskowicz

Vienna, 15th April, 2021

Tomáš Šedivý   Eva Maria Kühn

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Erklärung zur Verfassung der Arbeit

Tomáš Šedivý, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 15. April 2021

Tomáš Šedivý

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Danksagung

Ich möchte einer Reihe von Menschen, die dazu beigetragen haben, diese Masterarbeit zu ermöglichen, meinen herzlichen Dank aussprechen. Zunächst möchte ich meinen Betreuern A.o. Univ. Prof. Dr. Dipl.-Ing. Eva Maria Kühn und Dr. Dipl.-Ing. Gerson Joskowicz für Ihre Geduld, hilfreichen Ratschläge und Anleitungen danken. Ohne Ihre Hilfe wäre diese Masterarbeit nicht möglich gewesen. Als nächstes möchte ich meinem Arbeitgeber Ing. Robert Siegel, MBA dafür danken, dass er für mich zeitliche Bedingungen geschaffen hat, die mir ermöglicht haben, diese Arbeit zu schreiben. Ich möchte auch meiner Verlobten Lucia für ihre Liebe und Unterstützung in den letzten Jahren meines Studiums danken. Abschließend möchte ich meiner Familie, meinen Freunden und Kollegen für ihre Geduld, ihr Interesse und ihr Verständnis für diese Masterarbeit danken.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Acknowledgements

I would like to express my thanks to a number of people who contributed to make this thesis possible. First of all I would like to express my deep gratitude to my supervisors A.o. Univ. Prof. Dr. Dipl.-Ing. Eva Maria Kühn and Dr. Dipl.-Ing. Gerson Joskowicz. I thank you for your patience, helpful advice and guidance, without which this thesis would not have been possible. Next I like to thank my employer Ing. Robert Siegel, MBA for creating the time conditions that allowed me to write this thesis. I also would like to thank my fiance Lucia for her love and support during the final years of my studies. Finally I would like to thank my family, friends and colleagues for their patience, interest, and understanding with regards to this thesis.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Kurzfassung

Blockchain Technologien gewinnen immer mehr Interesse der Industrie und das nicht nur im Finanzsektor aber auch in anderen Disziplinen. Dies gilt insbesondere für permissionierte private Blockchains, welche über ein Zugriffskontrollmechanismus verfügen und die Datenprivatsphäre schützen. Permissionierte private Blockchains brauchen zusätzliche Off-chain Dienste die Funktionalität bereitstellen die man nicht mit einer Blockchain implementieren kann. Dazu gehört die Verteilung von privaten Daten und Nachrichtenübertragung.

XMPP ist ein ausgereiftes Kommunikationsprotokoll das sich nicht nur als Chat Kommunikationssystem sondern auch in vielen anderen Anwendungen als zuverlässig und hoch erweiterbar erwiesen hat.

Diese Arbeit analysiert die Eignung von XMPP für Off-chain Dienste und zeigt, dass XMPP alle Funktionen bietet, die für ein Off-chain Kommunikationsprotokoll erforderlich sind. Leider führt die Zentralisierte Architektur von XMPP zu einem zentralen Fehlerpunkt in einem stark verteilten System wie einer Blockchain.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Abstract

Blockchains are receiving an increased amount of attention not only in the financial sector but also in other disciplines. This applies specially to permissioned private blockchains, which provide a fine graded access control mechanism and ensure data privacy. Permissioned private blockchains rely on additional Off-chain services, which provide functionality, which cant be implemented on a blockchain. This includes private data distribution and messaging.

XMPP is a mature communication protocol, which has proven to be reliable and highly extensible not only as a chat communication system but also in many other different applications.

This thesis analyzes the suitability of XMPP for Off-chain services and shows that XMPP provides all features required by a Off-chain communication protocol. Unfortunately the centralized nature of XMPP creates a central point of failure in a highly distributed system like a blockchain.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Contents

Kurzfassung xi

Abstract xiii

Contents xv

1 Introduction 1
1.1 Aim of the Work . . . 2
1.2 Methodological Approach . . . 2
1.3 Structure of the Work . . . 2

2 Blockchain Basics 5
2.1 Blockchain Basics . . . 5
2.2 Transactions . . . 6
2.3 Consensus . . . 6
2.4 Data Structures . . . 7
2.5 Smart Contract . . . 8
2.6 Taxonomy . . . 8

3 Permissioned Blockchain Systems 11
3.1 Corda . . . 11
3.2 Hyperledger Fabric . . . 13
3.3 MultiChain . . . 15
3.4 Tendermint . . . 15
3.5 Quorum . . . 16

4 Problem Statement 19

5 Related Work 21
5.1 Corda . . . 21
5.2 Hyperledger Fabric . . . 22
5.3 MultiChain . . . 22
5.4 Tendermint . . . 23
5.5 Quorum . . . 24

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

6 Requirement Analysis 25
6.1 Functional Requirements 25
6.2 Non-Functional Requirements 26

7 XMPP 29
7.1 XMPP Architecture 29
7.2 Jabber Id 30
7.3 Network and XML Streams 31
7.4 Namespaces 32
7.5 Stanza 32
7.6 XMPP Security 37
7.7 XMPP Features and Extensions 38

8 Smart-Toolbox - Blockchain Provisioning 43
8.1 Smart-Toolbox Concept 43
8.2 Membership and Messaging Service 49
8.3 Smart-Toolbox Use Cases 50
8.4 Architecture and Use Cases 50
8.5 Key Management 67

9 Evaluation 71
9.1 Private Data Distribution 71
9.2 Offline Capability 72
9.3 Direct Messages 72
9.4 Message Broadcast 73
9.5 Synchronous Messages 73
9.6 Asynchronous Messages 74
9.7 Group Support and Dynamic Group Creation 74
9.8 User Query 75
9.9 User Groups 75
9.10 Cryptography Storage 75
9.11 Non-Functional Requirements 76
9.12 Disadvantages of XMPP 76
9.13 Summary 76

10 Conclusion and Future Work 79
10.1 Conclusion 79
10.2 Future Work 80

List of Figures 81

List of Tables 83

Acronyms 85

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 1

Introduction

Since the crypto-currency Bitcoin was proposed by the paper [Nak19] in 2008 and firstly solved the double spending problem without a central authority [CPV+16], blockchain systems have received enormous attention [DXM+19]. They are employed in many financial (equity exchange, stock market, diamond certification) and non-financial (documents authenticity verification, email certification, notary services) applications [CPV+16]. A blockchain is a distributed database that maintains an immutable ledger typically deployed in a peer-to-peer (P2P) network [DXM+19]. The ledger stores a certain and verifiable record of every transaction that has been executed in the blockchain, while maintaining the anonymity and privacy of all involved users [CPV+16].

This thesis deals with permissioned private blockchains, which utilize one or more authorities to decide if a given user can participate in the network [XWS+17]. This specific form of a blockchain requires access control mechanisms in which each participant has access only to a subset of all data stored on the ledger. In contradiction to these privacy requirements, trusted interactions between participants of a blockchain based system depend on full access for all participants to the complete ledger. Private blockchains therefore rely on additional communication mechanisms for dynamic group creation, negotiation of smart contracts, cryptographic material sharing and protection of private data [Hyp19].

The eXtensible Messaging and Presence Protocol (XMPP) is a communication standard for presence and instant messaging, which is based on a distributed client-server architecture and has a wide range of applications in industry. It features group management, multi user chat, direct messaging and a highly extensible architecture, which can be used to underpin additional peer-to-peer mechanisms like the exchange of cryptographic material and encrypted store-and-forward [SA11a].

The features and extensibility of XMPP suggest that this protocol is capable of replacing all Off-chain communication employed in various permissioned private blockchain systems,

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

while maintaining security and stability. This thesis analyzes the suitability of XMPP using an example program, which will demonstrate the suitability of XMPP in comparison with existing permissioned blockchain and distributed ledger systems.

1.1 Aim of the Work

This thesis analyzes the suitability of a standardized messaging system based on XMPP, as its features can be used to solve the problems which arise from the limited data visibility in a permissioned private blockchain, and to facilitate member management, dynamic group creation and Off-chain communication.

During the course of this thesis an example program was created to demonstrate an XMPP based implementation of the Off-chain features required by permissioned private blockchains. It is possible to invite participants (a subset of the entire community) for a given workflow or use case. Our implementation handles signature verification and the creation, signing and distribution of certificates. Example use cases include dynamic group creation through XMPP-based invitations of identified and pseudonymous users required by voting and auction applications.

The suitability of XMPP is analyzed by comparing the XMPP based implementation to equivalent ones found in other permissioned private blockchains based on the requirements for Off-chain communication defined in chapter 6.

1.2 Methodological Approach

In the first step, existing Off-chain communication mechanisms employed in various blockchain system have been analyzed, and based on the result of this analysis a list of requirements for an Off-chain communication system has been defined.

The next step was to design a suitable software architecture and to choose the technological stack for the implementation. Since this thesis is part of the Smart-toolbox project [sma], which is still being actively developed, the initial architecture evolved during the course of this thesis.

Finally, the suitability of XMPP has been evaluated through a comparison with existing Off-chain communication mechanisms employed in various private blockchain systems in terms of capabilities, performance and complexity, based on the requirements defined in the first step.

1.3 Structure of the Work

The thesis is structured as follows: the chapter 2 introduces the basics of blockchain technology in an implementation independent manner. The chapter 3 summarizes several popular permissioned private blockchain implementations. The chapter 4 defines the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

problem which is addressed in this thesis. The chapter 5 evaluates existing Off-chain communications systems employed in various permissioned private blockchains, which is then used as the basis for the chapter 6, which defines all functional and non-functional requirements for an Off-chain communications system.

XMPP is introduced in the chapter 7, which provides detailed information about the internal mechanisms of this protocol, and the chapter 8 introduces the Smart-Toolbox project.

The chapter 9 compares the features of our implementation with features of existing Off-chain communication systems based on the requirements defined in chapter 6, and finally the chapter 10 concludes this thesis and presents opportunities for future research.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 2

Blockchain Basics

Recently, blockchain systems have received enormous attention [DXM+19]. They are employed in many financial (equity exchange, stock market, diamond certification) and non-financial (documents authenticity verification, email certification, notary services) applications [CPV+16]. A blockchain is a distributed database that maintains an immutable ledger typically deployed in a peer-to-peer (P2P) network without relying on a central intermediary [DXM+19]. The ledger stores a certain and verifiable record of every transaction that has been executed in the blockchain, while maintaining the anonymity and privacy of all involved users. Each transaction is verified by a majority consensus of the blockchain participants and it cannot be erased or modified [CPV+16]. This chapter outlines the basic principles behind blockchain systems in a implementation independent fashion, while linking described technologies with concrete systems.

2.1 Blockchain Basics

This section introduces the basic principles behind the blockchain technology. Figure-2.1 illustrates a chain of blocks, which is a list of chronologically ordered blocks containing the hash of the previous block and a list of transactions [Vuk15]. Based on the consensus mechanism, the blockchain might also contain a nonce containing a solution for the

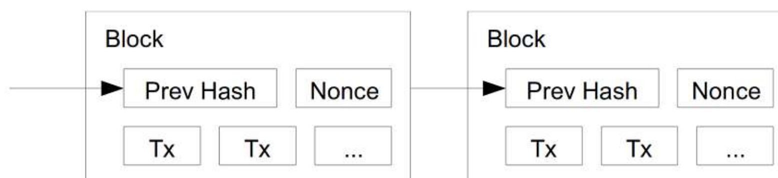

Figure 2.1: A chain of blocks. This figure is a reprint from [Nak19].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

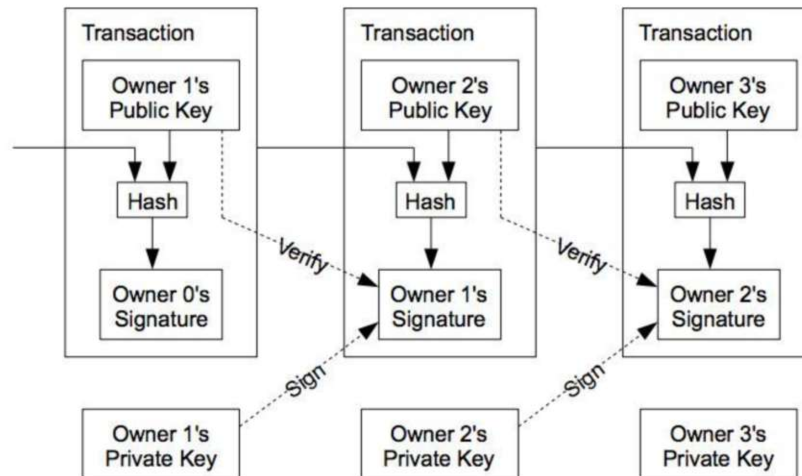

Figure 2.2: Transaction workflow of a blockchain. This figure is a reprint from [Nak19].

proof of work puzzle (see Section 2.3 for more details). The chain of blocks is known by each node participating in the blockchain system and new blocks are broadcasted automatically. The immutability of data is ensured by the fact that a transaction cannot be changed without the modification of hashes contained in previous blocks in every node included in the blockchain. If multiple new blocks are being added to the blockchain the system has to decide which block is valid [XWS+17]. This is decided by a consensus mechanism, which is discussed in section 2.3.

2.2 Transactions

One of the key challenges for transactions in a p2p based system which does not rely on a central intermediary is the double spending problem. The transaction mechanism in a blockchain system solves this issue by choosing the first transaction. As this has to be done without a central authority, each transaction has to be publicly announced and the majority of all participants have to agree using a consensus mechanism on the historical order in which the transactions have been received [Nak19]. The consensus mechanism is described in section 2.3. Ownership in a blockchain system is defined using a chain of digital signatures. As shown in figure-2.2, each transaction signs the hash of the previous transaction with the public key of the next owner and adds these to the end of the signature chain. The entire chain of ownership can then be validated using signature verification [Nak19].

2.3 Consensus

The consensus protocol has a major impact on the scalability and security of a blockchain system [XWS+17]. The role of a consensus protocol is to ensure unambiguous ordering

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

of transactions and blocks across all participants and to maintain the integrity and consistency of a blockchain. A consensus protocol has to ensure that the entire blockchain network eventually converges to a single chain without long term forks and it has to prevent Sybil attacks, where a single group of entities dominates and controls the consensus process, for example, by using fake identities [Bal17]. The following summarizes the basic principles behind various consensus systems.

The proof of work approach is based on a difficult computer problem, designed such that the correctness of a solution is easily verifiable, although the time needed to find such a solution is effectively random. A block can only be created if a particular instance of this computer problem is solved. This and the randomness of the required computation time creates a race for the block creation. This process is called mining and requires a large amount of computing power and electricity. The blockchain system also has to ensure that all participants consider the longest observed chain to be valid, which means that an attacker has to outrun all other miners if he wants to create faulty blocks [XWS+17]. Proof of work is used by bitcoin and multichain [DGG20].

The proof of stake approach is based on the idea that the owner of a digital currency is interested in the correctness of a transaction and thus selects a miner based on his stakes usually combined with a random factor. Proof of stake is computationally less expensive as proof of work and has smaller latency[XWS+17]. This approach can be optionally used in Tendermit [ten20b].

The proof of retrievability approach repurposes the energy costly proof of work mining puzzle so that the solution includes the storage of a part of a public dataset [MJS+14].

All blockchain systems rely on consensus mechanisms which are tolerant to Byzantine faults, since no two participants trust each other and there is no central authority. A Byzantine fault in a distributed system happens when a subset of its components share conflicting or wrong information [LSP19]. However, each participant is able to verify the correctness of the whole chain by inspecting the local copy of the replicated data. This approach has been adopted in Hyperledger fabric, which is a permissioned blockchain [Bal17], but also in core Tendermint in combination with proof of stake to prevent Sybil attacks [XWS+17], Corda [Hea16], and Quorum [Cha20].

2.4 Data Structures

As mentioned in section 2.1, the basic data structure used in blockchain systems is a chain (or list) of cryptographically linked blocks [Vuk15]. This section summarizes other data structures commonly used in blokchain implementations.

GHOST or the Greedy Heaviest-Observed Sub-Tree is a modification of the chain of blocks algorithm, where the conflict resolution of concurrent forks is based on the selection of the heaviest subtree [SZ15]. This allows shorter inter-block times and better throughput [XWS+17]. A variant of this mechanism is used in Ethereum [SZ15].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

The Block DAG approach uses a directed acyclic graph rather than the chain of blocks list structure, which makes it possible to include non-conflicting transactions from independently mined concurrent blocks in the main chain [XWS+17].

The Segregated Witness approach separates the signature from the data in the transaction so that the signature does decrease the block size limit. This can further decrease the size of all transactions, which are replicated on all nodes [XWS+17].

Another common data structure used in blockchains systems is a merkle tree, where each non leaf node is labelled with the hash of its child labels and each leaf is labelled with a value. In the context of blockchains this is usually a hash [Szy04]. Bitcoin utilizes merkle trees to save storage [Nak19]. Corda structures each transaction into a merkle tree, which allows oracle sigatures with limited data visibility [Hea16].

2.5 Smart Contract

A ledger in a blockchain system can be seen as a state transition system in which the modification of ownership is defined by a state transition function, also known as a smart contract [B+14]. Smart contracts are both automated and self enforcing, and are implemented as part of a computer protocol [CPV+16]. Early blockchain implementations, like bitcoin, had a very limited capability for auxiliary data and programmable transactions. Newer blockchains provide a programmable Turing complete infrastructure for smart contracts, which are executed within a transaction and are able to store computational results in the ledger [XWS+17]. Smart contracts can be based on one of the following computational models [Hea16]:

- The Unspent Transaction Output (UTXO) model, where each payment takes the previous unspent transaction as input and generates a new unspent transaction. This model is used by Bitcoin and Corda [ZXD+20].

- The virtual computer model, which is described as a single-threaded execution on the memory state of a global computer [Hea16], and the new state is recorded directly instead of calculation of unspent transactions. This model is used by Etherum and Hyperledger Fabric [Hea16].

2.6 Taxonomy

This section summarizes the taxonomy proposed in the paper[XWS+17], which analyses architectural characteristics of blockchains and defines a blockchain taxonomy. We will discuss the taxonomy based on level of decentralisation and blockchain infrastructure configuration.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

2.6.1 Level of centralization

A centralized system has one central trusted authority, which validates all transactions. This authority can manipulate the entire system and is the single point of failure. In contrast, a fully decentralized system cannot rely on mutual trust between users or a central entity. This includes permissionless blockchains such as Bitcoin or Ethereum, where anyone can join the network, validate transactions and mine blocks [XWS+17].

The paper [XWS+17] also defines partially decentralized (and partially centralized) systems based on two aspects permission and verification:

- A permissioned blockchain controls the access of users using one or more authorities, which may decide if a given user can join the network, mine or to initiate a transaction. A permission management mechanism may become a single point of failure. There are two kinds of permissioned blockchains:
  - Permissioned blockchains with fine graded access control.
  - Permissioned blockchains with privileged miners(write) and unprivileged normal nodes (read).
- Verification on the blockchain can only access data that is part of a transaction and no external system can be accessed directly [XWS+17]. This is because a transaction has to be deterministic and cannot depend on external data [Hea16]. This problem can be solved using a verifier, which is a trusted entity that can verify conditions that depend on external state. A verifier can be implemented in the following ways [XWS+17]:
  - A centralized verifier, which can be implemented as an external server that can sign a transaction with its own keypair. Such a server is a potential single point of failure [XWS+17].
  - A distributed verifier consisting of multiple verifiers, with the system relying on a multi-signature schema (M-of-N) that requires keys from M out on N verifiers [XWS+17].
  - An ad hoc verifier that is a trusted arbitrator that may be human or automated and is able to resolve disputes and sight transactions [XWS+17].

  The following blockchain systems have support for verifiers: Corda [Hea16] and Hyperledger fabric [LD20].

2.6.2 Blockchain scope

This subsection introduces the taxonomy of blockchains based on their access control mechanisms [XWS+17].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

- Public blockchains can be accessed by anyone on the internet. This results in better information transparency and auditability, but sacrifices performance for this increased security. Most digital currencies are public [XWS+17].
- Private permissioned blockchains control access and permissions using one or more central authorities. [XWS+17].
- Consortium blockchain is used by multiple organizations. Read access may be public or private and the consensus is controlled by a privileged node [XWS+17].

2.6.3 Data storage and computation

Data storage and computation on a blockchain is not only limited by storage capacity and computational power, but may also cost real money. Therefore, it is common practice to store raw data outside the blockchain i.e Off-chain . However, data stored on a blockchain may not only be for integration with external systems, but may also be used as On-chain auxiliary data, for example "coloured coins", which may be used a representation of real world assets on the bitcoin blockchain. An item collection is commonly stored directly on a blockchain, but a separate chain might also be used, which could increase flexibility as it is possible to optimize the auxiliary blockchain configuration for the stored data. Computation in a blockchain based system can be achieved On-chain using a smart contract or Off-chain i.e using the blockchain just as a data layer [XWS+17].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

CHAPTER 3

Permissioned Blockchain Systems

In contrast to public blockchains a private permissioned blockchain uses a central authority to control access to the network and provides a fine graded permissions system for data access, transactions and asset creation. This allows for better performance as the security does not rely on a costly proof of work consensus mechanism (see section 2.3 for more information) [XWS+17]. The permissioned private blockchain systems introduced in this chapter have been selected based on a literature review of recent articles and surveys addressing blockchain systems and their application in the industry. The table 3.1 summarizes the most commonly referenced blockchain systems.

3.1 Corda

Corda is developed by R3 and was first released in 2016. It aims to create a decentralized database, which can be used as a global ledger. Corda provides an application platform, which allows the implementation of new capabilities bundled in CorDapps typically stored in jar format. This makes it possible to define new datatypes, inter-node protocol flows and smart contracts. Corda uses a similar structure as the email network. A node requires an identity (which might be pseudonymous), but only a valid key pair is necessary to participate in the network if a node grants an authorized account. The communication in corda is abstracted using a programming model called flow, which is based on quasar fibers and uses a message broker based on Advanced Message Queuing Protocol (AMQP). Flow

	[AA20]	[Wan21]	[SIHC21]	[NRP+21]	[HKG+21]	[BVGC21]	[Alr21]
Multichain	•	•	•	•	•	•	•
Tendermint	•					•	•
Quorum	•	•	•	•		•	•
Hyperledger Fabric	•	•	•		•	•	•
Corda	•	•	•	•	•		•

Figure 3.1: Permissioned blockchain systems referenced in recent literature

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

allows interruptible transactions, off-line operation and re-synchronization of participating nodes [Hea16].

3.1.1 Network

Corda uses a peer to peer network for communication and all messages are transported using AMQP secured by Transport Layer Security (TLS) (see section 5.1 for more information). The Corda network consists of the following parts [Hea16] :

- Nodes, which act as an application server for CorDapps. Each node provides access to the peer to peer network, cryptographic key signing a relational database and access to the vault, which stores ledger data relevant to the node owner. [Hea16].
- Identity service provided by an X509 signature authority [Hea16].
- Network map service, which provides node specific network information such as ip addresses and identity certificates. [Hea16].
- Notary services, which are responsible for transaction ordering and timestamping. [Hea16].
- Oracle services, which act as a verifier for external states (see section 2.6.1 for more information) [Hea16].

3.1.2 Consensus

The consensus system employed in Corda is not based on time organized into blocks. Instead the transaction ordering is handled by a notary service, which should be maintained by each distrusting party. Notaries process each transaction and return either a rejection error, in case of double spending, or a signature, which indicates the transaction finality. Corda does not enforce a concrete consensus algorithm and even allows the employment of different notary implementations in the same network. This allows for a fine graded performance - trust trade off [Hea16].

Corda splits the consensus mechanism in two parts: Transaction validity and transaction uniqueness. This separation increases privacy as the transaction validity can be verified solely by the participants using smart contracts and signature verification. And the transaction uniqueness, which prevents double spending, can be achieved without private data containend in the transaction [BCGH16].

3.1.3 Transactions

Corda defines states as atomic unmodifiable units of information, which are either current (unspent) or consumed (spent). Each Transaction consumes one or more states as input and create zero or more new states. Every state has an identifier (StateRef), which consists of the creating transaction and its output index. A transaction definition contains

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

a list of input state references, which will be consumed during execution, a list of non-consuming input references, a list of output states (including the notary, contracts and data), which will be created during the execution, a list of commands, which are used as parameters for the executed contract (this includes data from oracles), a transaction type (normal, notary-changing or explicit upgrades), signatures, timestamps, and network parameters [Hea16].

A transaction in Corda becomes valid when it receives signatures from all required signers, this always includes notaries, which are responsible for transaction ordering and validation but may include oracles and owners depending on the executed contract [Hea16].

Each transaction in Corda is structured into a merkle tree (see section 2.4 for more details), where the label of the root identifies the transaction. This data-structure allows transactions to be signed by an oracle or notary without complete data visibility as entire branches containing sensitive data can be removed [Hea16].

The recorded transactions are not globally visible and the read access is limited to relevant participants. This is achieved using cryptographic hashes used for data and participant identification [BCGH16].

3.2 Hyperledger Fabric

Hyperledger Fabric is developed by the linux foundation under the Hyperledger project, which was established in 2016 [C+16]. It is an open-source distributed operating system for permissioned blockchains with an extensible architecture. It allows for modular deployment of the ordering service, membership service, gossip service and the ledger. The ordering service is used for the consensus mechanism. The membership service maintains all identities and can act as Certification Authority (CA). The gossip service disseminates blocks and distributes private data (for more information see 5.2). The ledger acts as a value store and is located on each peer [ABB+18].

Fabric was the first blockchain system, which supports the execution of applications written in a standard programming language like Java or Go. One fabric network can run multiple blockchains called channels. Each channel has an independent consensus and configuration [ABB+18].

3.2.1 Network

As you can see in figure 3.2, the Fabric network consists of clients, peers, ordering service nodes and the Membership Service Provider (MSP). Clients create transaction proposals. Peers validate and execute transactions. The ordering service nodes handle the ordering of new blocks and the MSP maintains all identities [ABB+18].

Each participating node has an identity with specific permissions, which is assigned by the MSP. The communication between nodes is handled using a peer-to-peer gossip

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

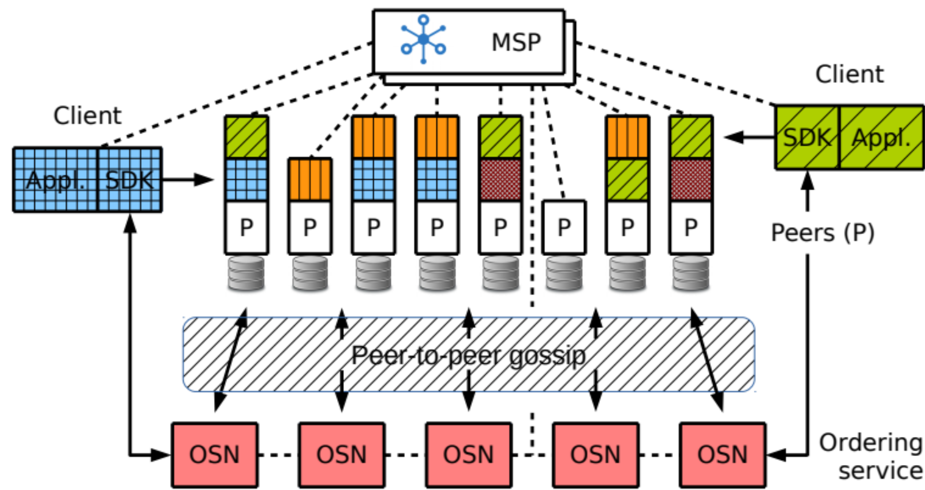

Figure 3.2: Fabric network, this figure is a reprint from [ABB+18].

protocol, which synchronizes the nodes [ABB+18] and distributes private data based on access permissions (for more information see 5.2) [Hyp19].

3.2.2 Consensus

A transaction in Hyperledger Fabric consists of the header, signature, proposal, response and endorsement. The header contains all metadata related to the transaction. The proposal contains the parameters for the smart contract execution. The response holds a read/write set containing the before and after values of the modified states. The endorsement holds all required signatures based on the endorsement policy [Hyp20].

Hyperledger fabric is based on the execute-order-validate paradigm, which splits the transactions mechanism into three parts [ABB+18] :

- Transaction execution, where the clients create and sign a transaction proposal, which is sent to one or more endorsers depending on the smart contract [ABB+18].
- Transaction ordering, where the client submits a transaction based on the proposal signed by all endorsers to the ordering service. This service guarantees the total order of all submitted transactions and atomically broadcasts endorsements ensuring consensus. As the ordering service does not verify any transaction there is no need for the blockchain state to be stored on the ordering service. The separation of this step allows for high modularity of the consensus mechanism [ABB+18].
- Transaction validation phase is executed after the ordering service or the gossip protocol distribute the new blocks. This phase validates all signatures, checks for read/write conflicts and updates the local ledger [ABB+18].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Each of the steps may be executed on different nodes [ABB+18].

3.3 MultiChain

MultiChain is an easy to use platform for creation and deployment of private blockchains supporting Windows, Linux and Mac. It aims to create a smooth transition between the bitcoin blockchain, private blockchains and vice versa. All permissions are managed on the blockchain and can be granted or revoked using special transactions. The genesis block defines an administrator with all permissions. MultiChain was initially forked from the bitcoin core and has the same architecture [DGG20].

3.3.1 Network

A MultiChain deployment can run multiple blockchain instances, where each blockchain consists of multiple nodes. Each node has a node address consisting of the blockchain name and node ip address. Each new blockchain instance consists initially only of one node, which mines the genesis block. New nodes can join the blockchain only after a permission transaction, which authorizes the new node to be executed [DGG20].

3.3.2 Consensus

The consensus in MultiChain is achieved using a mining process similar to bitcoins proof of work (see section 2.3 for more information). But in contrast to bitcoin the security of MultiChain does not rely on proof of work, it is only used to ensure mining diversity as all mining entities are identifiable and authorized to mine. The mining process additionally prevents monopolization using a round robin approach, where only a given number of blocks can be created by one node, within a given time window [DGG20].

3.4 Tendermint

Tendermint is an open source Byzantine Fault Tolerant (BFT) state machine replication blockchain platform, consisting of Tendermint core, which provides a BFT consensus, high level interface for deterministic applications (Application BlockChain Interface) and management tools [BKM18].

3.4.1 Network

The Tendermint network consist of the following node types [ten20b]:

- Validator Nodes, which are responsible for the consensus [ten20b]. These nodes should use a sentry node for communication as they are the core of the consensus mechanism securing the Tendermint network.
- Full Nodes, which maintain a local ledger and validate transaction [ten20b].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

- Seed Nodes, which are responsible for p2p node discovery and maintain a list of known peers in a local address book [ten20d].
- Sentry Nodes, which act as a network proxy for validator nodes, ensuring their safety [ten20b].

The nodes in the Tendermint network can communicate using a direct TCP connection using the p2p package or using Remote Procedure Call (RPC) over HTTP. Both communication systems are discussed in section 5.4 [ten20b].

3.4.2 Consensus

Tendermint is a distributed DLS based BFT consensus protocol, which is byzantine fault tolerant if at least 2/3 of nodes are not compromised [Kwo14]. The Consensus is facilitated by authorized participants called validators, which propose blocks of transactions and vote on block validity. A block is considered committed if more then 2/3 of the validators vote for validity. This guarantees that the safety of the system will not be violated if less then 1/3 of validators are Byzantine. The security of consensus can be enhanced with a proof of stake mechanism if the deployed application uses a currency (see section 2.3 for more information) [ten20b]. Transactions in Tendermint are encoded as arbitrary byte arrays and the application, which is built on the Tendermint core, decides the content of these arrays. This increases the flexibility of Tendermint for various use cases [ten20a].

3.5 Quorum

Quorum is a private permissioned blockchain system based on the official etherum protocol implementation. As seen in figure 3.3, privacy ensuring changes are implemented in a separate layer above the standard implementation [Cha20]. This includes the transaction manager, crypto enclave, consensus implementations and the network manager. The Transaction Manager handles access to private data, manages local data and communicates with other transaction managers. The Crypto Enclave handles the cryptography and private key management. The Network Manager is responsible for network access control [Cha20].

Quorum introduces a new private transaction type, which uses cryptography to prevent unauthorized access to sensitive data. All data is stored on a shared blockchain. This requires some changes in the smart contract architecture and the modification of the proposal and validation system. The contract execution during the verification of private transactions is skipped in nodes, which are not part of the transaction [Cha20].

These modifications enable the segmentation of the state database separating the public and private states. The network is in perfect consensus on the public states, but the private states may differ [Cha20].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

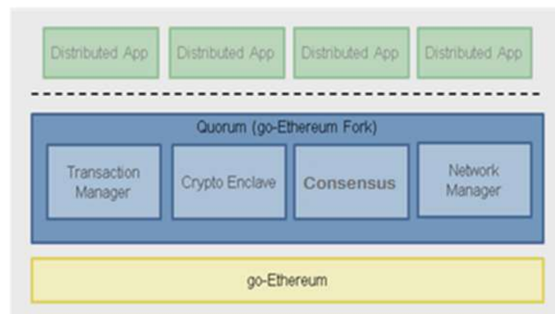

Figure 3.3: Quorum architecture, this figure is a reprint from [Cha20].

3.5.1 Consensus

The consensus in Quorum can be achieved using the Raft or Istanbul BFT protocols [Cha20].

The Raft based consensus can be used in a closed membership environment, where each Quourum (Etherum) node corresponds to one Raft node. The entire cluster has only one leader, which is chosen using a voting process. The leader orders transactions into blocks without the need of proof of work. A Block is accepted if it has been verified by a majority of raft nodes [Cha20].

The Istanbul BFT protocol can tolerate up to one third of all validator nodes to be faulty. It uses a three-phase consensus mechanism. Block validators, which are chosen using a voting process, pick a proposer node in each consensus round. The proposer node creates a new block, which is broadcast in the first step. This block is then validated by all validators, which then broadcast a prepare message. The last step commits the state and is executed after a validator node receives a prepare message from at least two thirds of all validator nodes [Cha20].

3.5.2 Network

The nodes in Quorum network can have the following roles [Tee17]:

- A Maker node is responsible for the creation of new blocks. Each Maker has a randomly set timer, which starts the creation of a new block. The timer is reset after a new block from a different maker has been received [Tee17].
- A Voter node participates in the block selection process depending on the defined consensus mechanism [Tee17].
- An Observer node is a node which has neither a vote role nor a maker role assigned. Such a node can only receive and validate blocks [Tee17].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

These roles are defined in the genesis block but can be assigned during runtime post blockchain creation as the network changes [Tee17]. A node can have assigned both voter and maker roles at once.

3.5.3 Transactions

Transactions in quorum consist of the recipient, signature of the sender, optional ether amount, list of participants allowed to see the transaction and a payload (a hash value in case of a private transaction) [Bas18].

Contract code in a public transaction is executed on each node, resulting in a state change. Private transaction contracts cannot be executed on each node as the private payload is replaced with a hash value before the transaction is propagated to other nodes. Only authorized participants are able to access the original payload using Tessera or the Constellation peer to peer protocol (see section 5.5 for more information) [Tee17].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

CHAPTER 4

Problem Statement

This thesis deals with permissioned private blockchains, which use a central authority to control access to the network and provide a fine graded permissions system for data access, transactions and asset creation (see chapter 3 for more information) [XWS+17]. The verification system in a blockchain relies on data visibility by every authorized participant. This makes the data stored in a permissioned private blockchain public to all authorized users. This problem can be mitigated by the following approaches [sma20]:

- The separation of data and blockchains, which can be achieved using cryptographic hashes of the actual data. These approaches requires an additional Off-chain mechanism for data distribution [sma20]. For example hyperledger fabric uses the gossip protocol for private data distribution (see section 5.2 for more information) [ABB+18].
- A separate blockchain, where each participant is authorized to access the private data. For example, hyperledger fabric uses this concept in so called channels, which are immutable and cannot be altered or dynamically created [sma20].

Each of these mitigation approaches require a mechanism for user authorization or group management to ensure data visibility for authorized users. In case of a separate blockchain an asset synchronization system is also required to prevent double spending in separate chains.

XMPP is a mature and extensible communication standard, which is adopted in a wide range of applications in different industries. XMPP provides dynamic group creation, store and forward messaging and an invitation system using standard extensions. These features suggest that XMPP is capable of replacing all Off-chain communication employed in various permissioned private blockchain systems, while maintaining security and stability.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

This thesis aims to analyze the suitability of XMPP especially for Off-chain services which include key management, private data exchange and dynamic mutable group creation, where the invitation system is used to dynamically group users and the messaging functionality of XMPP is used to exchange cryptographic material and distribute private data.

The suitability of XMPP for asset synchronization is outside the scope of this thesis.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

CHAPTER 5

Related Work

This chapter evaluates existing Off-chain communication systems employed in various permissioned private blockchains.

5.1 Corda

Corda is a private permissioned blockchain system developed by R3, which supports smart contracts, observer nodes and different consensus mechanisms (see section 3.1 for more information) [Hea16]. Corda uses a peer to peer network for communication and all messages are transported using AMQP, secured by TLS. The reference implementation uses the Apache Artemis message broker, which supports journalling, load balancing, clustering and streaming of messages too large to fit in RAM. Similar to the email network, nodes are expected to be long lived, with a globally unique identity, which ensures delivery messages are sent using the store and forward technique, where each message is stored on the hard drive until the recipient confirms delivery. This prevents data loss on unstable networks and in the case of node downtime [Hea16].

The AMQP is an open binary protocol for message exchange. It provides java compatible primitive types but is extensible and allows natural java mapping to business types. Each message has a unique id, which prevents redelivery [Hea16]. A AMQP network consists of named entities called nodes, which can send, receive or relay messages. These nodes exist in a container, which may contain multiple different nodes. Each node may act as a producer, consumer, or a queue, which stores and forwards messages [Sta12]. Corda additionally defines a session-id, which allows long lived sessions even across server restarts and ensures that all messages are self describing. The deserialization system in Corda is implemented with versioning in mind. Each data stream is deserialized to a named class. A process called data evolution is triggered in cases where the name does not exactly match. This process ensures values are mapped with existing classes, allowing for default values in new code and discarding of future fields where possible. If

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

no matching class is found, the framework synthesizes a new class, which can be accessed using reflection. Corda nodes also provide a basic RPC mechanism, which is also based on AMQP and allows direct interaction with a node [Hea16].

5.2 Hyperledger Fabric

Hyperledger Fabric has been developed by the linux foundation under the Hyperledger project since 2016. It is an open-source distributed operating system for permissioned blockchains with an extensible architecture (see section 3.2 for more information). Peer communication in Fabric is based on the gossip protocol, which is responsible for the distribution of new execution results to all nodes and the synchronization of newly joined nodes, which were offline and therefore have outdated state. The gossip protocol utilizes epidemic multicast, which enables fast detection of offline peers, while maintaining a list of active peers (membership view). The dissemination of new blocks using the gossip protocol is achieved in two steps. First, in the push phase, each node randomly selects target peers from the membership view and transfers the message. Second, during the pull step, each node randomly selects a list of peers and requests missing messages. This process synchronizes all nodes, which can then reconstruct the entire blockchain, while effectively utilizing the network resources [ABB+18]. Additionally, the gossip protocol transfers private data to authorized peers [Hyp19]. The message exchange in the gossip protocol is facilitated using gRPC in combination with TLS for security [ABB+18].

The gRPC library is a general-purpose RPC system, initially created by google, and is commonly used in microservice architectures. As usual, in a RPC system a gRPC client is able to call a server method transparently to the client side developer as if the server were a local object. It has support for many common deployment environments and programming languages such as java, c#, c++, go, php, javascript and others. All messages in gPRC are transported in binary format using the HTTP/2 protocol. The encoding of messages is facilitated using protocol buffers, which is an open source serialization framework developed by google. The gRPC api supports bidirectional streaming and synchronous and asynchronous calls [Fou20].

5.3 MultiChain

MultiChain is an easy to use platform for creation and deployment of private blockchains and it aims to create a smooth transition between the bitcoin blockchain, private blockchains and vice versa. MultiChain was initially forked from the bitcoin core and has the same architecture (see section 3.3 for more information) [DGG20].

MultiChain streams provide an abstraction layer, which hides the complexity of the underlying blockchain. A multichain stream can be used as a key value database (document store), a time series database, or an identity driven database. Streams can be created and closed dynamically with different access rule configurations. Each stream represents an append only collection of items, where each item is stored as a separate

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

blockchain transaction, including an id, a timestamp, data and one or more publishers which have signed the item. The id is used for later retrieval in the case of a key value database. The data has dynamic length up to many megabytes. The timestamp is equal to the block header timestamp [Ltd20a].

Private data access in Multichain can be realized using a combination of three streams. One is used for public key distribution, another for data distribution and contains items with encrypted data. The final one is used for decryption key distribution and is limited to authorized participants. This allows blockchain data storage with limited data visibility [Ltd20a].

5.4 Tendermint

Tendermint is an open source BFT state machine replication blockchain platform designed to be integrated into various applications [BKM18]. The communication between nodes can be facilitated using a RPC mechanism or a native p2p protocol [ten20b].

5.4.1 RPC

Tendermint core includes an RPC system, which allows for remote invocation of methods using the JSONRPC over web sockets, JSONRPC over Hypertext Transfer Protocol (HTTP) or the Uniform Resource Identifier (URI) over HTTP protocol, which is a custom REST like interface [ten20c].

JSONRPC is a public communication specification for RPC communication. It is stateless and transport agnostic supporting http and web sockets. The protocol encodes all request and response messages using json [Gro13].

5.4.2 Peer to Peer protocol

Each peer in a Tendermint peer to peer network has an id consisting of the port, ip address and an identifier corresponding to the private key of the peer. All connections use TCP with an Station To Station (STS) protocol, secured using X25519 and chacha20poly1305. The messages sent over this protocol are grouped into channels and each transferred message has a channel identifier. The protocol supports synchronous and asynchronous messaging and defines three types of packets [ten20b]:

- The ping packet consists of only one byte and is used as a keep alive mechanism, which also checks if a given connection is still alive. A pong packet is expected within a timeout after a ping packet is sent [ten20b].
- A pong packet is sent after a ping packet was received [ten20b].
- The msg is used to deliver messages between peers and, besides data, also contains a channel id and End Of File (EOF) flag, which is used to mark the end of a message consisting of multiple msg packets [ten20b].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Data transferred using msg packets is encoded using go-amino [ten20b]. Amino is a binary encoding format for complex objects, which is based on Proto3 [ten19].

5.5 Quorum

Quorum is a private permissioned blockchain system based on the official etherum protocol implementation (see section 3.5 for more information). The peer to peer communication and the distribution of private data is handled by the transaction manager [Con20c]. There are currently two transaction manager implementations in Quorum:

- Constellation, which is a general-purpose message exchange system allowing encrypted communication between peers [Bas18].
- Tessera, which is an open source private transaction manager developed for the Quorum blockchain [Con20d].

5.5.1 Constellation

Constellation is a haskel based peer to peer system, which was developed as a privacy engine for the Quorum blockchain. Constellation is able to transfer encrypted data between automatically managed nodes. It also acts as a distributed key server, where each node holds a list of encryption keys. This list, and the list of known hosts, is updated during the startup process, where the new node contacts all known nodes provided during startup. Each contacted node shares its know encryption keys and node list, which is then used for further key exchange and the process is repeated until the node is synchronized [Con20b].

5.5.2 Tessera

Tessera is a REST based peer to peer system written in java and provides services for encryption, decryption and distribution of private data in Quorum. Similarly to Constellation, Tessera also maintans a list of encryption keys and manages the discovery and synchronization of peer to peer nodes [Con20d].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

CHAPTER 6

Requirement Analysis

This chapter defines requirements for an Off-chain messaging system capable of key management, group management, and data synchronization. The definition is based on existing communication mechanisms employed in different permissioned blockchain implementations as described in chapter 5. The requirements are categorized into functional requirements, which describe concrete features, and non-functional requirements, which describe system properties.

6.1 Functional Requirements

6.1.1 Private Data Distribution

The system shall be able to distribute private data to authorized users.

6.1.2 Offline Capability

All messages transferred within the communication system shall be delivered even if the recipient is offline or not reachable due to network outages. This is usually achieved using a store and forward mechanism, where the message is persisted to the hard drive until delivery is possible. This will allow for asynchronous invites, where the invitee receives an invite after reconnection.

6.1.3 Direct Messages

The message system shall facilitate direct message exchange between participants. This will allow human to human interaction and might be used for private data exchange.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

6.1.4 Message Broadcast

The message system shall be able to send the same message to a defined set of recipients at once. This allows direct communication between a dynamic group of participants.

6.1.5 Synchronous Messages

The system shall support synchronous message delivery. This approach allows the synchronous execution of processes like public key publishing.

6.1.6 Asynchronous Messages

The system shall support asynchronous message delivery. This approach allows the asynchronous execution of processes like public invites.

6.1.7 Group Support

The system shall be able to create a group for private data exchange.

6.1.8 Dynamic Group Creation

The system shall be able to dynamically create a group for private data exchange.

6.1.9 User Query

The system shall be able to list all registered users, which can then be invited.

6.1.10 User Groups

The system shall be able to group users into logical groups (for example departments). This will allow the invitation of multiple users at once.

6.1.11 Cryptography Storage

The system shall be able to store cryptographic material.

6.2 Non-Functional Requirements

6.2.1 Performance and Scalability

The system should run with reasonable performance and be highly scalable.

6.2.2 Availability

The system should allow for a high availability deployment. This is usually achieved using redundancy.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

6.2.3 Encryption

Each connection which is used to deliver messages shall be encrypted. The security of every communication system is dependent on encryption as this prevents network sniffing and men in the middle attacks.

6.2.4 Security

The system has to protect the stored keys from unauthorized access.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

CHAPTER 7

XMPP

The XMPP is a communication standard, for presence and instant messaging, which is built on a distributed client-server architecture. The core specification of XMPP was published in 2004 based on an instant messaging system called Jabber and has since been adopted in a wide range of applications through the industry, not only for instant communication but also in games, geolocation, cloud computing, disaster management, public transport and smart cities [IVBA+14]. The protocol is asynchronous, XML based and works across multiple servers with different domains similarly to the email network. It features a highly extensible architecture, which is not only employed in various XMPP Extensions Protocol (XEP) extensions but can also be used for custom payloads [SA11a]. Public XEP specifications are managed by the XMPP Standards Foundation (XSF), which oversees the proposals, documentation and standardization of new XMPP extensions [SAC10]. This chapter outlines the architecture and basic principles defined in the XMPP specification. The section 7.1 introduces the distributed server architecture, which is the basis for XMPP. XML streams and network communication are described in section 7.3. The section 7.5 deals with stanzas and the last section 7.7 then summarizes various extensions, which are relevant to this thesis.

7.1 XMPP Architecture

XMPP enables close to realtime asynchronous end to end exchange of short structured xml encoded data called XML stanzas (handled in section 7.5), using continuous streams between globally addressable presence-aware clients and servers. The typical XMPP implementation is based on a distributed client-server architecture, which can be seen in figure-7.1. Each user authenticates with its server using a globally unique address called Jabber ID (JID), which is based on the domain name system. XMPP allows multiple authorized connections with the same JID as each connection is identified using a unique resource id. For more information about JID see section-7.2. An authenticated

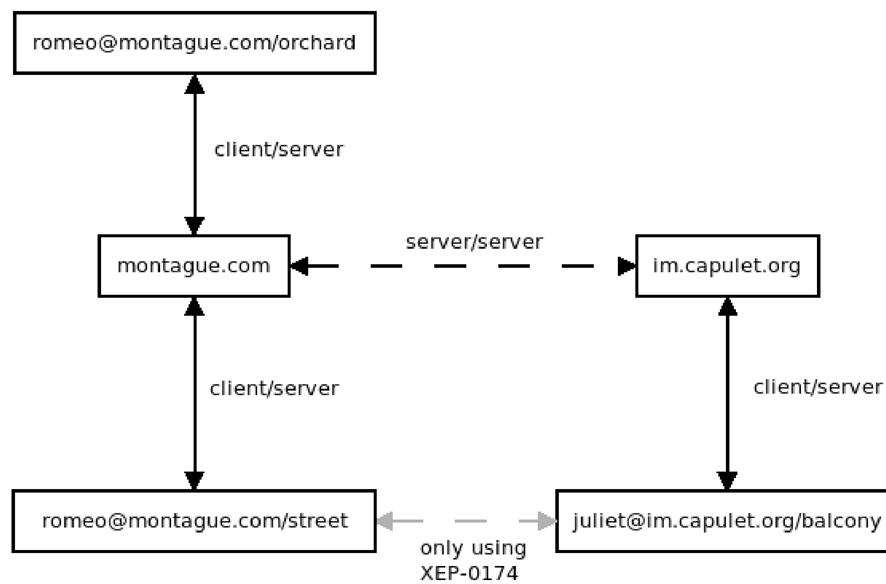

Figure 7.1: Distributed client server architecture. This figure is based on [SA11a].

client can exchange XML stanzas with other participants of the network, while the other participants do not have to be connected to the same server as the XMPP protocol defines an inter domain or inter server relay mechanism, where one server connects to another server and exchanges XML stanzas on behalf of its associated entities. Beside authentication a XMPP server has to manage XML streams (handled in section 7.3) with local clients and remote servers, deliver stanzas between connected clients, forward stanzas to other servers for further routing, store client data (for example contact lists) and host extensions [SA11a].

7.2 Jabber Id

The routing mechanism of XMPP requires each entity connected to the network to be globally addressable. This not only applies to servers and clients, but also to various additional services, and is achieved using a globally unique identifier called JID. JIDs are based on the domain name system similarly to the email network and consist of three parts [SA11a]:

- The domain part is a fully qualified domain name, which typically identifies the server responsible for the routing and authentication for a given network participant, but may also identify other entities, like for example a multi user chat service [SA11b].
- The local part is an optional identifier, which is typically used to identify entities associated with a server, like for example a user account, but it may also identify a chatroom associated with a muti user chat service [SA11b].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

- The resource part is an optional identifier, which is used to distinguish connections authenticated using the same JID. It is unique among all connected resources and is defined during connection initialization either by the client or by the server [SA11a].

The XMPP standard defines 2 types of JIDs [SA11a]:

- Bare JID, which is defined in the following format: localpart@domain and is used to identify entities in the XMPP network without relation to a specific connection [SA11a].
- Full JID, which is defined in the following format: localpart@domainpart/resource and is used to identify entities in the context of different connections [SA11a].

7.3 Network and XML Streams

The network communication in XMPP is facilitated through client to server and server to server TCP connections [SA11a]. Each client connects to a server, which handles the routing based on the recipients JID. Direct client to client communication is not supported by the core specification, but is possible using an extension called serverless messaging (XEP-0174) [SA08b], which is described in section 7.7.10. A connection life cycle consist of the following steps [SA11a]:

1. Determine the server ip address and port. This step is usually done using domain name resolution.
2. Open a TCP connection.
3. Open XML stream.
4. Optionally negotiate TLS encryption using XML commands.
5. Authenticate using Simple Authentication and Security Layer (SASL).
6. Bind resource to stream. This step is executed only in the case of a client to server connection, and its purpose is to define a unique resource id for the full JID associated with this connection [SA11a]. For more information see section 7.2.
7. Exchange XML stanzas.
8. Close XML Stream.
9. Close TCP connection.

An XML stream is the backbone of the XMPP protocol. As shown in figure-7.2 it is basicaly an XML container, enclosing all XML encoded data transferred over the life cycle of the connection and begins with a stream header, which is just a <stream> tag

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

containing appropriate namespaces and attributes like from, to and version. The end of the stream is marked by a closing XML element </stream> tag. An XML stream is used to transfer stanzas (see section 7.5) and other xml elements, which are used for XMPP signaling and negotiations , such as stream errors, TLS or SASL metadata. Each stream transfers data only in one direction, which means that bidirectional communication requires at least two streams. This can be achieved in one of the following ways [SA11a]:

- Two streams over a single TCP connection, which is typical for a client to server session. This approach uses the same security context for both streams [SA11a].
- Two streams over two TCP connections securing each stream separately. This approach is typical for server to server communication [SA11a].
- Multiple streams over multiple TCP connections with separate security contexts. This is sometimes used for server to server communication for large XMPP service providers.

The figure-7.2 shows a simplified example of bidirectional communication consisting of a initial and response XML stream. The initiating stream contains a presence, message and iq stanza but only the iq stanza results in a result stanza in the response stream [SA11a]. For more information about stanzas see section 7.5.

7.4 Namespaces

The role of XML namespaces is to prevent clashes between names in different markup vocabularies [BHLC09]. XMPP makes extensive use of this mechanism to create strict boundaries of data ownership and to facilitate the extensibility of the basic stanza syntax for additional functionality. Message and presence stanzas may contain one or more child elements, with a specific namespace, which defines the semantics of they content. The core specification of XMPP defines the jabber:client namespace for client to server communication and jabber:server namespace for server to server communication. The only difference between them is that the jabber:server namespace requires the definition of the to and from attributes. Each XMPP extension is free to define its own set of qualifying namespaces, which are used to create and identify stanzas used by the extension [SA11a].

7.5 Stanza

The basic minimal unit employed in the XMPP protocol is called an XML stanza. It is defined as the first level element of the XMl stream whose name is message, presence or iq and its namespace is either jabber:client or jabber:server. There are three types of stanzas [SA11a]:

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

(a) Initial stream

(b) Response stream

Figure 7.2: Bidirectional communication using two streams. This figure is based on [SA11a].

- A message stanza is encoded with a <message> tag and is used for push messages [SA11a].
- A presence stanza is encoded with a <presence> tag and is used for publish-subscribe messages and broadcasting [SA11a].
- An iq stanza (Info/Query) is encoded with an <iq> tag and is used for request-response communication [SA11a].

An XML stanza usually contains one or more XML child elements and attributes, which encode the transferred information. The following attributes are common for all stanzas [SA11a]:

- The to attribute contains the recipients JID. The server uses it for routing and delivers the stanza if possible. If the to attribute is not defined the server processes the stanzas depending on the type. Message stanzas are handled as if the to

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

attribute would contain the JID of the sender. Presence stanzas are broadcast to subscribers and iq stanzas are handled directly by the server [SA11a].

- The from attribute contains the full JID of the sender in the case of message and iq stanzas, and the bare JID in case of subscription-related presence stanzas. The server has to make sure that the from value is correct and can override it based on the originating XML input stream if necessary [SA11a].
- The id attribute identifies the stanza in the context of an XML stream. The sender utilizes the id to track response or error stanzas based on the rule that the id of the answer has to match the id of the send stanza. This attribute is mandatory only for iq stanzas and its value has to be unique for the XML stream [SA11a].
- The type attribute defines the purpose of a stanza and the only value which is common to message, presence and iq stanzas is error. Each stanza type defines its own list of possible values, which are described in section-7.5.1 (iq stanza), section-7.5.2 (message stanza) and 7.5.3 (presence stanza) [SA11a].
- The xml:lang attribute defines the language of human readable characters contained in the xml stanza. Child tags can override this attribute for their content. The server may not modify or delete this attribute but should add it if it is missing. The default value is determined based on the value defined in the output XML stream of the client [SA11a].

.

7.5.1 Iq Stanza

The purpose of an iq stanza is to facilitate request-response communication between authenticated entities. As shown in figure-7.3 the requesting entity sends an iq stanza with the type 'get'. The recipient then processes the query and responds with an iq stanza of the type 'result'. The id attribute is used to map the send stanza to the received response. The XMPP standard defines the get, set, result and error types for iq stanzas. which can be identified by the value of type attribute. The get stanza type is used for data queries, where the recipient returns the requested value. The set stanza type is used to define or to replace existing values on the recipient entity. The result type marks a response iq stanza, which is returned after successful processing of a get or set iq stanza. The error type is returned when the processing of a set or get iq stanza fails [SA11a].

7.5.2 Message Stanza

The figure-7.4 shows a message stanza, which facilitates the push mechanism of XMPP, where one entity can send data to another entity [SA11a]. They are used for single

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

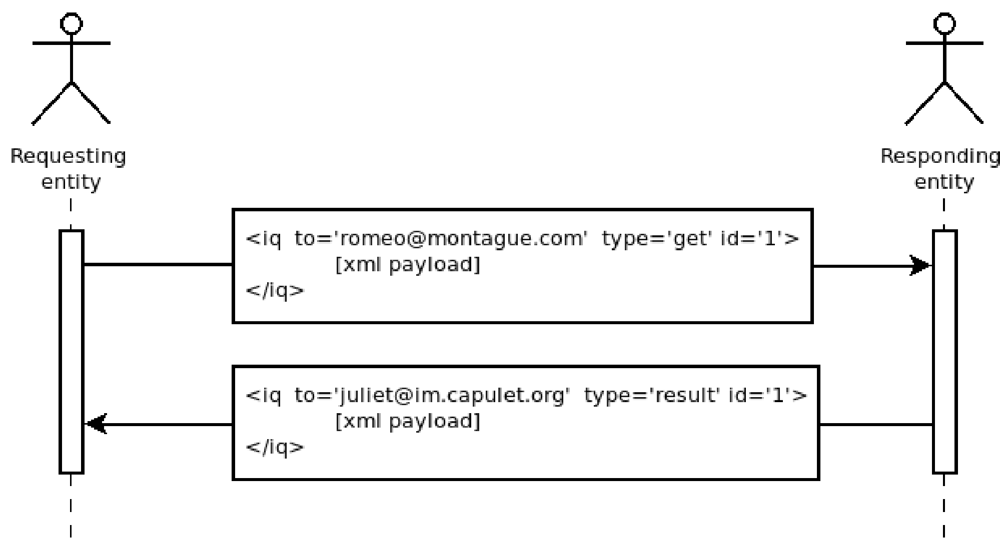

Figure 7.3: Iq stanza example. This figure is based on [SA11a].

```
<message from="local@domain/resource"
      to="romeo@montague.com"
      xml:lang="en" type="normal">
   <body>
      Art thou not Romeo, and a Montague?
   </body>
</message>
```

Figure 7.4: Example of a message stanza. This figure is based on [SA11a].

messages, one to one chat, alerts, notifications and other. The XMPP standard defines 5 types of message stanzas. The chat type is used in the context of one to one chat messages, which are usually displayed to the user with a chat history. The groupchat type is used in the context of many to many chats. The headline type is used for notifications and alerts. This message type does not expect that a receiver sends a response. The normal type is used in messages, which are sent outside of the context of one to one or many to many chats, and are usually shown to the user without a history. The error type is returned if the processing of a message fails [RFC11].

7.5.3 Presence Stanza

The figure-7.5 shows an example of a presence stanza, which facilitates the broadcast mechanism of XMPP, in which multiple subscribed entities receive information from one

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

```
<presence type="probe"
   from="local@domain/resource" xml:lang='en'>
      <show>dnd</show>
      <status>Wooing Juliet</status>
</presence>
```

Figure 7.5: Example of a presence stanza. This figure is based on [SA11a].

publishing entity. The publisher should send presence stanzas without the to attribute as it is sent to all subscribers automatically [SA11a]. The XMPP standard defines 7 types of presence stanzas. The probe type is used to determine the current presence of an entity. This presence type should be generated only by the server. The subscribe type is sent by the subscribing entity to permanently subscribe to a contact's presence information. The subscribed type is sent to the subscribing entity when the publisher allows the subscription. The unavailable type is used to inform the subscriber that the sender is no longer available. The unsubscribe type is sent by the subscribing entity to permanently unsubscribe from a contact's presence information. The error type is returned if the processing of a presence stanza failed [RFC11].

7.5.4 Stanza Error

An error stanza is usually sent back if the processing of a received stanza fails. It is of the same kind as the failing stanza but has the type set to error. The from and to attributes are usually swapped, but the id stays the same so that the error can be mapped by the receiving entity. As you can see in figure-7.6, each error stanza contains an error tag with a type attribute, which informs the sending entity under what conditions it should send the stanza again. XMPP defines 5 error values for the type attribute. The auth value signals that the sending entity should try again after providing credentials. The cancel value means that the sending entity should not try to send the stanza again as the error cannot be corrected. The continue value marks warnings and the sending entity can continue. The modify value signals that the request should be modified before the next attempt. The wait value means that the error is only temporary. [SA11a].

The figure-7.6 also contains an internal-server-error tag, which is one of the possible defined conditions specified in [SA11a]. They are mandatory and specify the cause of the error. The following summarizes some of the error conditions defined in [SA11a]:

- The bad-request condition signifies that the received stanza was malformed or had an unknown namespace. The error type should be modify [SA11a].
- The conflict condition means that a resource with the given name already exists. The error type should be cancel [SA11a].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

```
<message type="error" from="local@domain/resource"
     to="romeo@montague.com">
   <error type="cancel">
     <internal-server-error
        xmlns="urn:ietf:params:xml:ns:xmpp-stanzas"/>
   </error>
</message>
```

Figure 7.6: Example of a stanza error. This figure is based on [SA11a].

- The feature-not-implemented condition is returned if the received namespace is known but the requested feature is not implemented. The error type should be cancel or modify [SA11a].
- The forbidden condition informs the requesting entity that it does not have the necessary permissions. The error type should be auth [SA11a].
- The gone condition signifies that the recipient can no longer be found on the given address. The error type should be cancel and the new address should be included in the gone tag [SA11a].
- The internal-server-error condition is returned if the server-side processing of the stanza caused an internal error. The error type should be cancel [SA11a].
- The item-not-found condition means that the addressed JID or requested item cannot be found. The error type should be cancel [SA11a].
- The redirect condition is used to signify that the requesting entity should send the stanza to another entity, which is written in the redirect tag. This redirect is usually temporary and the error type should be modify [SA11a].

7.6 XMPP Security

The basis of XMPP security is channel encryption based on TLS with STARTTLS. This encyption mechanism has to be implemented by every XMPP client and server. Although its usage is optional, TLS should be preferred in production environments as it ensures the integrity and confidentiality of transferred data and can prevent sniffing, stanza replays, man-in-the-middle and other security attacks [SA11a]. TLS can be used to secure the client to server and server to server communication, but cannot be used to protect data during the routing process, which requires that the server can read the header of the transmitted stanza [IVBA+14]. This disadvantage can be overcome using end to

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

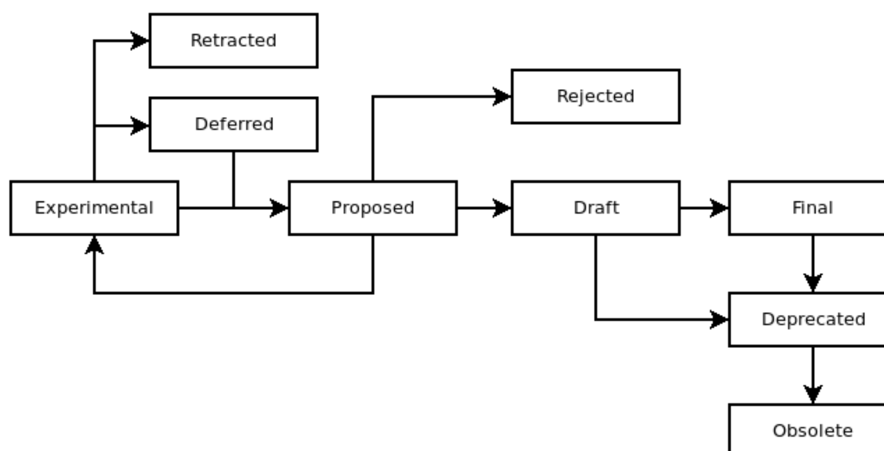

Figure 7.7: Xep life cycle. This figure is based on [SAC10].

end encryption proposed in the OpenPGP for XMPP extension (XEP-0373) [SSB16], or similar plugins. See section 7.7 for more information about this extension.

7.7 XMPP Features and Extensions

XMPP is highly extensible by design and many basic features, which are not part of the core specification, are specified as extensions[SA11a]. All public XMPP specifications are managed by the XSF, which oversees the proposals, documentation and standardization of new extensions called XEPs [SAC10]. This section summarizes the document life cycle of standards track XEPs and introduces some of the public extensions of XMPP, which are related to this thesis and might be relevant for blockchain provisioning.

7.7.1 Xsf Standardization Process

[SAC10] defines five types of XEP specifications. Informational XEPs usually describe best practices for implementation and deployment of existing protocols. Historical XEPs provide documentation for protocols, which were developed before XSF was instituted. Humorous XEPs specify protocols, which should never be used in a production system and are published as a joke. Procedural XEPs describe a process or activity of XSF, and finally the standards track XEPs define a protocol or a protocol suite intended to be an extension for XMPP.

The figure-7.7 shows the standardization life cycle of a standards track XEP and the following summarizes its states as defined in [SAC10]:

- The experimental state is assigned to a XEP specification, which has been accepted by the XMPP council and published by the XSF. It is not recommended to use experimental XEP specifications in a production system [SAC10].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

- The proposed state is assigned to a XEP specification, which is under consideration by the XMPP council to be advanced to draft [SAC10].

- The draft state is assigned to a XEP specification after being subjected to extensive discussion and it is expected that it will be used as a base for a production ready implementation. Usage in mission critical applications should be avoided as backwards compatibility breaking modifications to the standard might be present [SAC10].

- The final state is assigned to a XEP specification if it has been in the draft state for at least six months, has been adopted by at least two separate projects, and has been advanced by the XMPP council. A final XEP should not be subject to compatibility breaking modifications, but limited modifications are possible as long as they are optional extensions [SAC10].

- The deferred state is assigned to a XEP specification if it is in the experimental state and has not been updated for twelve months [SAC10].

- The retracted state is assigned to a XEP specification if its author requests its removal [SAC10].

- The rejected state is assigned to a XEP specification which has been rejected by the XMPP council [SAC10].

- The deprecated state is assigned to a XEP specification which should not be implemented in new projects as it is outdated or has been replaced with a different protocol [SAC10].

- The obsolete state is assigned to a XEP specification when the XMPP council decides that it should no longer be implemented or deployed [SAC10].

7.7.2 Jingle

The jingle is an xmpp extension, which provides a one to one, peer to peer data channel. The data is usually not transferred using XMPP, but using a modular dedicated transport [LBSA+09].

7.7.3 Jingle File Transfer

The jingle file transfer extension provides a modular framework, which makes it possible to exchange file information and to negotiate a peer to peer session for file transfer. The file can be transfered over any jingle transport mechanism [SAS11].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

7.7.4 Multi User Chat

The multi user chat extension (XEP-0045) facilitates message exchange between multiple users in the context of a room or channel. The current status of the specification is draft and it features topics, invitations and a room membership control system [SA08a].

This extension introduces an occupant JID (additionally to bare JID and full JID described in section 7.2), which has the following form: room@service/nick. The occupant JID is used to identify a user in a specific chat room and to facilitate anonymity in an anonymous chat room. It consists of the room part, which is the room id, the service part, which corresponds to the hostname of the chat service and the nick part, which is the users desired nickname within the chatroom. The multi user chat extension specifies multiple roles with different privileges. For example, not every room member can send invites or ban other users. Each chat room can have the following attributes [SA08a]:

- A room can be member only, which means that only users contained in the member list can join. The member list is automatically extended by the invite mechanism [SA08a].
- A room can be moderated, which means that only users with a voice are allowed to send messages. This is controlled by moderators [SA08a].
- A room can be open without access control or protection by a password [SA08a].
- A room can be temporary or persistent. A temporary room is automatically destroyed when the last user leaves and a persistent room is never automatically destroyed [SA08a].

A multi user chat room holds a message history, which is delivered automatically after login. The amount of historical entries that get sent depends on room configuration [SA08a].

7.7.5 Data Forms

The data form (XEP-0004) introduces a mechanism for form processing, which can be used for different workflows such as service configuration. For example, a multi user chat room service might allow a user to configure a room using a form based on this extension. The current status of this specification is final and it features form validation, request, response, submit and cancel semantics and different field types like boolean, hidden, list options and others [EHM+04].

7.7.6 Service Discovery

The Service discovery (XEP-0030) specification defines a flexible and extensible mechanism for information discovery, where each entity participating on the network can use

TU Bibliothek WIEN Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

this extension to request information from other entities. The current status of this specification is final and it defines three kinds of discoverable data [HMESA08]:

- The identity of the target entity, which consists of a category (server, client ...) and a type, which belongs to this category (IM server, phone vs. handheld client). It is possible to define multiple identities for each entity. This data can be used for classification and for better user interaction [HMESA08].
- The supported features and protocols of the target entity [HMESA08].
- The items or entities associated with an entity. For example, a list of hosted rooms in a multi user chat [HMESA08].

7.7.7 Roster

A contact list in a XMPP context is called a roster and it defines a list of user specific contacts called roster items. The roster system in XMPP is tightly bound with presence subscription, which allows access to presence information about other users, provided they have approved the subscription request. Each roster item defines a JID, name, ask and an approved attribute. The JID attribute identifies a concrete user, which is referenced by the roster item. It is additionally used to uniquely identify each roster item. The name is not used by the server and it is defined by the roster owner. Its purpose is to allow the use of user friendly names in the contact list. The ask attribute is used to signal various subscription states. The approved attribute is used to signal subscription pre-approval [RFC11].

The roster is stored on the server and can be accessed by any device. It is automatically retrieved at login and should be refreshed regularly [RFC11]. Additionally, some XMPP servers support shared groups. This allows the administrator to define a roster group, which is automatically included in the roster of all authorized users.

7.7.8 Anonymous Connections

SASL, which is used for authentication in XMPP, supports multiple authentication mechanisms. One of them is SASL ANONYMOUS, which makes it possible to connect to an XMPP server without credentials and so anonymously without a user account. This can be used in conbination with the multi user chat extension to ensure anonymity [SA09].

7.7.9 Ad-Hoc Commands

The Ad-hoc commands (XEP-0050) specification defines a XMPP protocol extension, which enables the execution and discovery of application specific commands without the need for custom stanzas. This extension relies on service discovery (XEP-0030) for command advertising and usually uses data forms XEP-0004 for information exchange. The current status of this specification is draft [Mil05].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

7.7.10 Serverless Messaging

The Serverless messaging (XEP-0174) specification extends the core XMPP standard with direct client to client communication in a Local Area Network (LAN). This extension uses Multicast DNS (mDNS) to discover local clients, which can then initiate a direct XML stream with optional TLS encryption for data exchange. The current state of this specification is final [SA08c].

7.7.11 OpenPGP for XMPP

The OpenPGP for XMPP (XEP-0373) specification defines a mechanism for end to end and multi end to multi end encryption, based on digital signatures using OpenPGP. This extension provides a standardized way for public key discovery and a synchronization system for private keys, which can manage secret keys across multiple devices. The current state of this specification is deferred [SSB16].

7.7.12 Private XML Storage

The historical private XML storage (XEP-0049) specification describes a private storage system, which was already part of jabber. Every client can store arbitrary XML encoded data on the server using an iq stanza with the jabber:iq:private namespace. The data privacy is ensured by the fact that only the user which stored the data has read access [SAD04].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

CHAPTER 8

Smart-Toolbox - Blockchain Provisioning

This chapter introduces the basic concepts and software architecture of the Smart-Toolbox project in section 8.1. The section 8.2 introduces the membership and messaging services, which were implemented using XMPP during the course of this thesis. The section 8.3 shows example use cases for Smart-Toolbox and the section 8.4 introduces our implementation, which shows how XMPP can be used to solve the issues which arise from the limited data visibility in a permissioned private blockchain.

Smart-Toolbox is an FFG BRIDGE Project, which is developed by the Complang research group at TU WIEN. It aims to create a simple to use development environment, which facilitates the creation and verification of smart contracts using a pattern based methodology [sma]. The Smart-Toolbox project provides an application environment running on top of a permissioned blockchain system, while hiding its complexity and providing support for different systems [sma20].

8.1 Smart-Toolbox Concept

The security of transactions between members in a distributed system usually relies on the shared trust of a central or a third party authority, but the same level of trust can also be achieved using a blockchain system without a single point of failure (see chapter 2 for more information). Unfortunately, the deployment of a blockchain system and the definition of a correct smart contract is a complex process, which prevents the wide adoption of blockchains. Smart-Toolbox aims to create an easy to use interface between collaborative software and permissioned blockchain systems, which can be used to prevent the execution of unauthorized actions, to ensure data consistency, and to automate contract rules. This approach also allows the review and verification of the global state

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

by each participant. Additionally, Smart-Toolbox provides a graphical user interface allowing state visualization, creation and deployment of distributed applications based on smart contracts, dynamic creation of participating groups, and general blockchain secured interaction between participants [sma20].

The figure-8.1 shows a diagram of the Smart-Toolbox application environment, which consists of the following parts [sma20]:

- The Smart-Toolbox application, which provides a Graphical User Interface (GUI) and the toolbox runtime used by the end user [sma20].
- The common ledger, which is implemented using a permissioned blockchain system, ensures that the necessary public information is synchronized, allowing verification of proposals [sma20].
- The community services, which take care of Off-chain functionality, including the membership service, messaging service, and the distribution and validation of keys [sma20].
- The Smart-Toolbox library, which holds all template and artifact prototypes [sma20].

Community Blockchain

The Smart-Toolbox project defines a community blockchain as a permissioned private blockchain (see chapter 3 for more information), which allows the dynamic creation of small distributed applications where each participant, which is usually a member of a community (club, party, association etc), may create an independent self organizing group, invite other participants, create a template based smart contract, and execute its interactions. A community blockchain enables the ad-hoc creation of collaborative applications in an inter-organizational environment, dynamic group creation using an invite mechanism, and complex end user defined interactions between participants. A community blockchain has the following end user requirements [sma20]:

- A user has to be able to define a business proccess based on a smart contract template [sma20].
- A user has to be able to verify a contract and to understand its meaning [sma20].
- A user has to be able to deploy a contract and to invite other community members to participate [sma20].
- A user has to be able to join and to leave a contract [sma20].
- A user has to be able to execute the process steps as defined in a contract [sma20].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

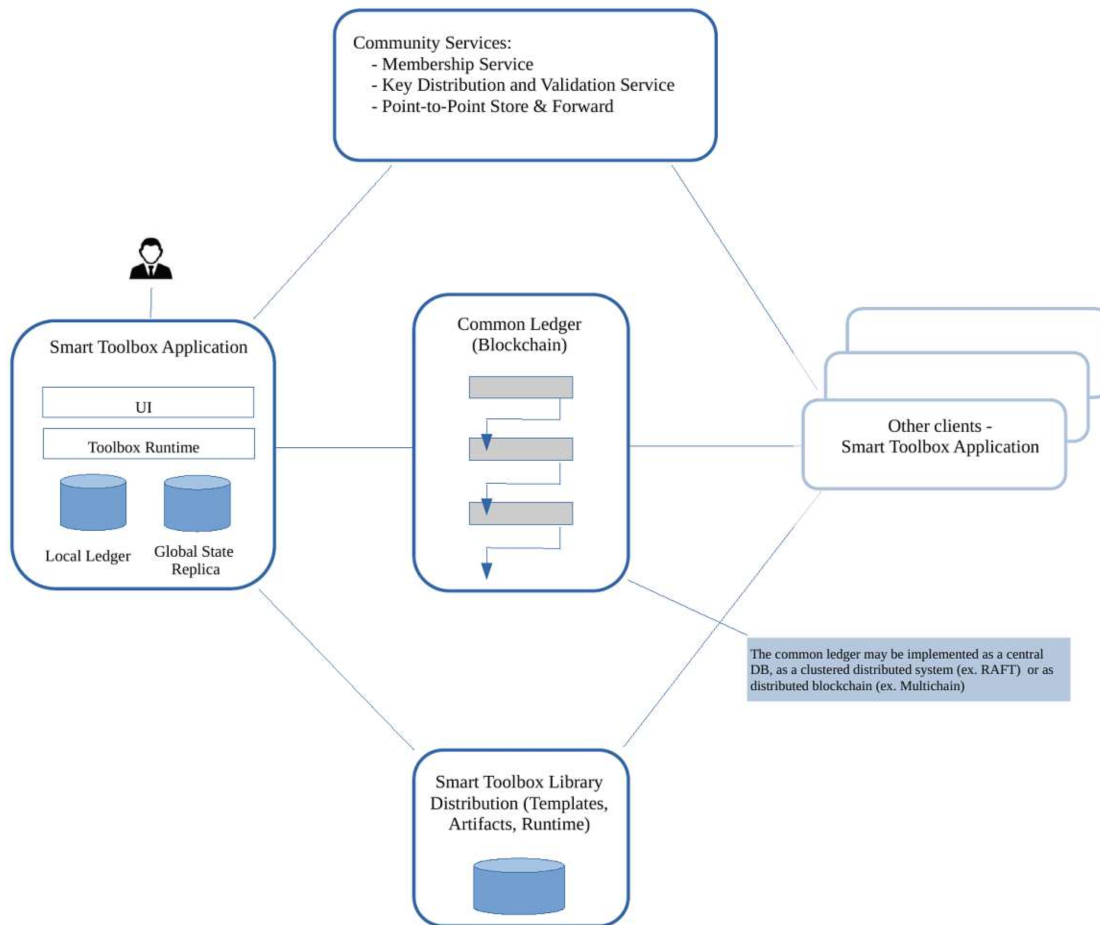

Figure 8.1: Smart-Toolbox application environment. This figure is a reprint from [sma20].

A contract in the context of community blockchains is defined as a well defined interaction between participants. As you can see in figure-8.2, each contact contains artifacts, which either describe a participant or a real world object reference called an asset. Each contract is based on a prototype defining its properties like name, description, business logic and rules, which specify which properties need to be fulfilled by related artifacts. A contract represents an interaction group within a community blockchain [sma20].

Abstract Blockchain Architecture

The Smart-Toolbox project aims to support multiple blockchain infrastructures like MultiChain, Hyperledger Fabric, Corda/R3, Tendermint, and others. This is achieved using the abstract blockchain architecture. As shown in figure-8.3, it consists of the following parts [sma20]:

- The wallet or application part handles the end user interaction, visualization of the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek, WIEN Your knowledge hub

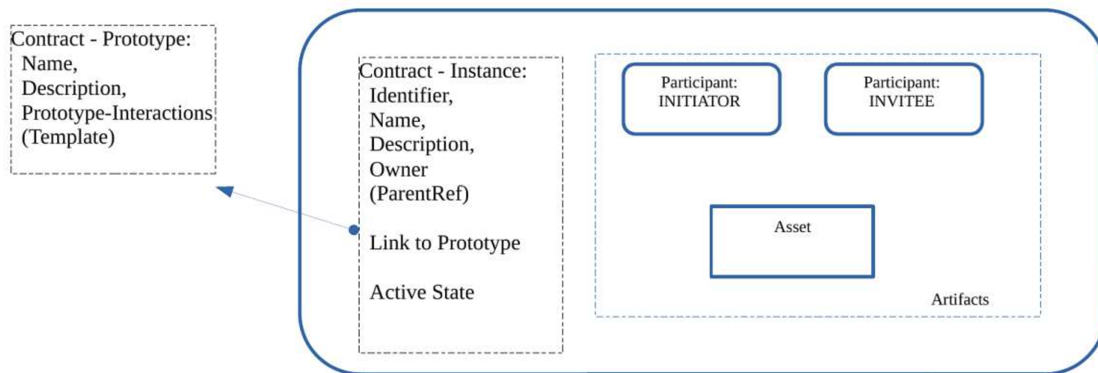

Figure 8.2: Contract layout. This figure is a reprint from [sma20].

local state, and contract status using an automatically generated GUI [sma20].

- The local state, which can be updated only by the chaincode based on the globally accepted proposals [sma20].
- The blockchain core, which synchronizes the embedded distributed ledger over all participants, maintaining consensus [sma20].
- The chaincode, which interprets operations defined in a proposal. Each proposal may originate on either the same or a different node and is either rejected or accepted. Accepted proposals are applied to the local state [sma20].

The figure-8.3 also shows interfaces defining the interaction between each part [sma20]:

- Interface A defines an abstraction between the application and the smart contract, allowing the creation of update proposals and access to the contract state information, including notifications about changes [sma20].
- Interface B defines an abstraction between the smart contract and the blockchain core, allowing access to the global state, and subscription to blockchain events [sma20].
- Interface C defines an abstraction between the blockchain core and chaincode, allowing the evaluation of proposals and modification or retrial of the global state [sma20].

Smart-Toolbox Application Environment

The Smart-Toolbox project provides an application environment, which aims to allow the end user to create, use and publish blockchain applications hiding the complexity of

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

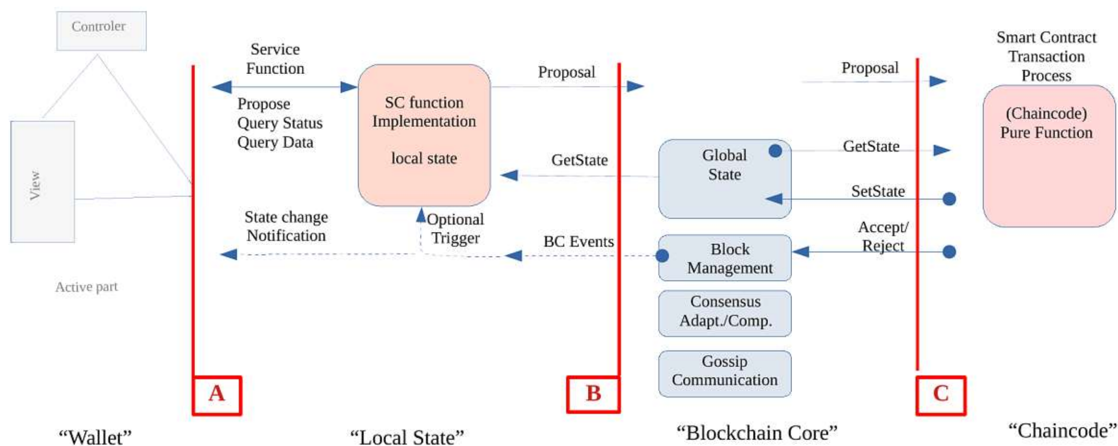

Figure 8.3: Abstract blockchain architecture, this figure is a reprint from [sma20].

smart contracts and different blockchain systems. The figure-8.4 shows a diagram, which summarizes all parts of the Smart-Toolbox application environment [sma20]:

- The template library holds predefined templates for business processes. For example, the transfer of an asset or voting on a topic. Each template defines a set of artifacts, which are linked to a real contract, participants or assets. Contracts represent the state of an interaction relating participants to assets. Each participant has a role defined by the template and each asset represents a real world object like money or a book [sma20].
- The artifact library holds concrete artifact clases, which define specific properties of real world objects. For example, a book can be transferred but cannot be divided like money [sma20].
- The application runtime visualizes the local state and contract status, providing a generated GUI, allowing the user to execute actions for the given template [sma20].
- The chaincode runtime interprets all actions submitted within a proposal and then executes the contract, resulting in the rejection or acceptance of proposals. Accepted proposals are applied to the local state [sma20].
- The blockchain core synchronizes the embedded distributed ledger over all participants and maintains consensus [sma20].
- The Off-chain services provide Off-chain communication, membership services and replicate private data [sma20].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

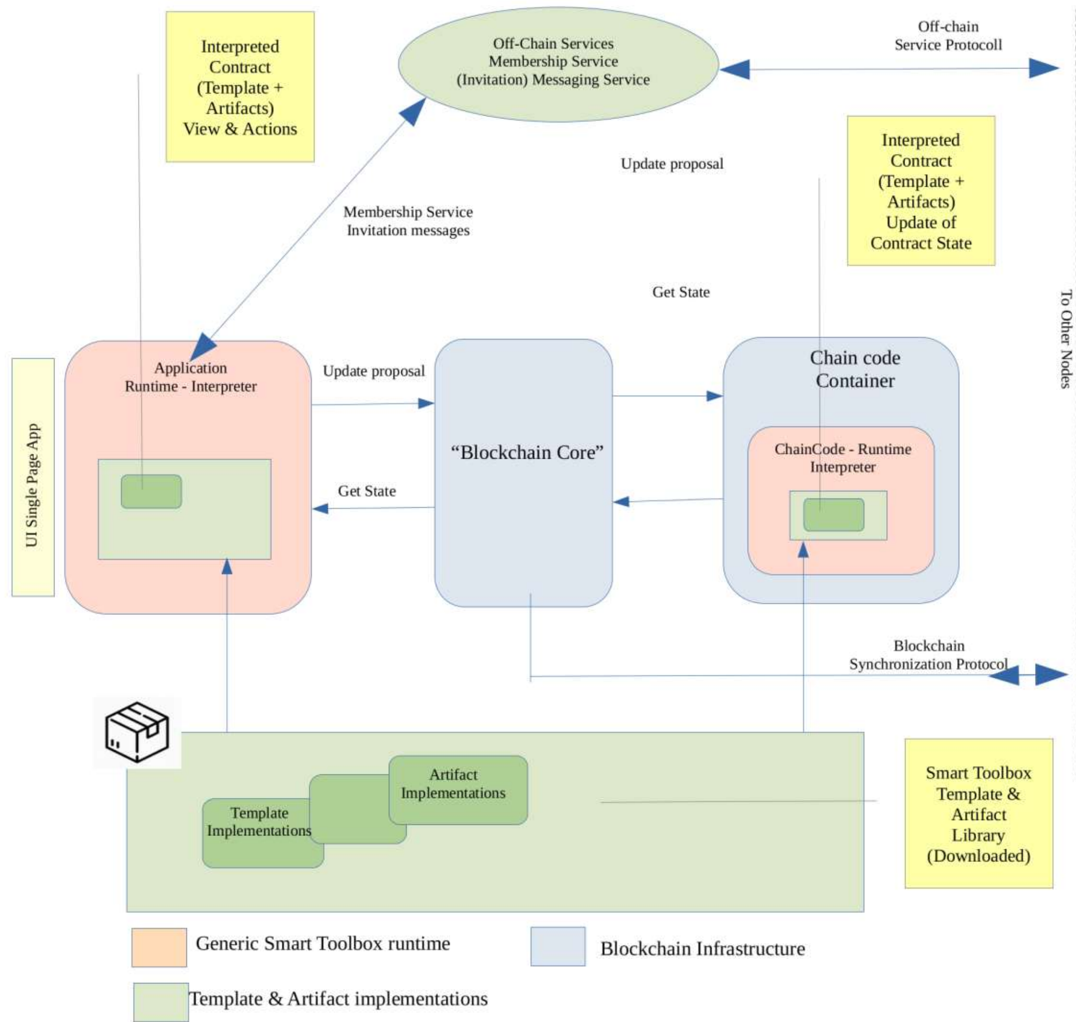

Figure 8.4: Smart-Toolbox application environment. This figure is a reprint from [sma20].

Off-Chain Services

Not all features of Smart-Toolbox are implemented using blockchain technology. The following summarizes Off-chain services used by the system [sma20]:

- The membership service manages user identities bound to X509 certificates. These identities are bound to participants during the execution of templates [sma20].
- The library distribution service ensures that all artifact and template libraries are replicated to each node, which in turn ensures they have the same version [sma20].
- The dynamic group formation and contract advertisement is facilitated using a messaging service. Each member can initiate a new contract and send invites to other users, which may accept and join dynamically, creating a new group, which then participates in the contract [sma20].
- Private data has to be distributed using an Off-chain mechanism, as each blockchain participant has access to the ledger and thus might access private data without authorization. This mechanism ensures consistency and privacy of data using a dedicated blockchain for every dynamically created group, which contains only hashes of the actual data, and point to point connections, which distribute private data only to authorized users [sma20].

This thesis deals with various Off-chain communication services in the context of Smart-Toolbox and analyzes the suitability of XMPP as a membership and messaging service in permissioned private blockchains.

8.2 Membership and Messaging Service

The membership service in Smart-Toolbox manages the user identities and binds them to X509 certificates [sma20]. Our XMPP based implementation provides a membership service, which utilizes the openfire user management system, which holds all registered users. We have also implemented an openfire plugin, which is responsible for the key management and stores certificates related to a given user. Because this is a proof of concept implementation, the plugin also acts as a trusted authority and signs all issued client certificates with the domain certificate. All signed certificates are then stored in a local LevelDB instance so that the keys can be retrieved even after the server is restarted. The plugin also provides signed public keys for signature verification to the client side. The key management is handled in section 8.5.

The messaging service in Smart-Toolbox handles the dynamic creation of self organizing groups using the publish-invite-join principle, which is an essential feature in a community blockchain [sma20]. Our XMPP based implementation provides this functionality using the multi user chat extensions (see section 7.7.4 for more information). This extension was used without any modification as it already provides the required privacy features

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

and an invite system, which can be seamlessly used to transfer metadata required by the underlying blockchain system (the invitation mechanism is discussed in section 8.4.4). This demonstrates that XMPP, in combination with the multi user chat extension, fulfills the following requirements: 6.1.2, 6.1.4, 6.1.3, 6.1.8.

8.3 Smart-Toolbox Use Cases

Smart-Toolbox use cases can be divided into the following categories[sma20]:

- Community-wide or group-wide decision making processes, where contract participants make a decision using votes. For example, voting, selection, auctioning, etc. [sma20].
- Transactions, where contract participants exchange or share digital assets according to contract rules in a verifiable way. This includes shared ownership, equipment use, etc. [sma20].

The following describes an example workflow for an auction use case. As described in section 8.1, the template library holds predefined templates for business processes and the artifacts library contains prototype artifacts defining the properties of assets. An auction template has a placeholder for a non-fungible asset, which gets sold, and placeholders for bids, which can be linked to fungible assets. The workflow can be divided into the following steps [sma20]:

- The initiating user (initiator) has to create assets using the prototypes defined in the artifacts library. As required by the auction template the new asset has to be non-fungible[sma20].
- The seller selects the auction template, specifies all other details required for an auction and invites buyers (invitees)[sma20].
- Each invitee can accept the invitation and place offers, which are only visible to the seller and the submitting buyer. Each offer has to be bound to a fungible asset as defined by the template [sma20].
- The auction is finished when the seller accepts an offer. This triggers an exchange transaction on the blockchain ensuring both parties receive the expected asset [sma20].

8.4 Architecture and Use Cases

As stated in section 8.1, the Smart-Toolbox application environment and permissioned private blockchains rely on various Off-chain services. We have developed a reference XMPP based implementation for the membership and messaging service during the course of this thesis (for more information about XMPP see chapter 7). This section

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

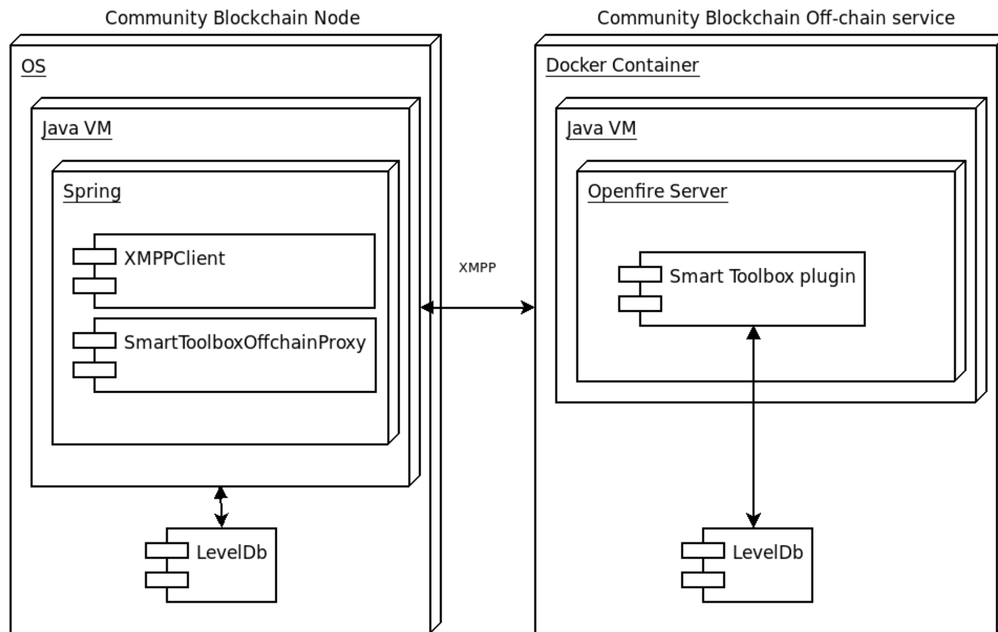

Figure 8.5: Deployment diagram of the XMPP integration implemented over the course of this thesis.

summarizes the architecture of our implementation and explains various design decisions in the context of the requirements defined in chapter 6.

We have chosen to use the spring framework as the base for our implementations as it allowed us to utilize the inversion of control design pattern and therefore dependency injection. Spring also made the integration with the existing Smart-Toolbox infrastructure easier as it also uses spring and relies on the spring mvc library for the GUI.

Our implementation divides responsibility for the requirements according to a client server architecture, which is based on the XMPP protocol. The figure-8.5 shows the deployment diagram of our test system, where the server side is deployed inside a docker container. We have chosen the openfire XMPP server as it supports all required XEP extensions and features a highly extensible architecture.

The server provides all standard XMPP communication features and also acts as a membership service.

The client side is integrated in the Smart-Toolbox project as a spring bean called SmartToolboxOffchainProxy. This bean handles all offchain communication, ensures the current state is not lost upon restarts and hides the details of low level message exchange. This allows for future research comparing different communication technologies to XMPP. Another important part shown in the deployment diagram in figure-8.5 is the XMPPClient bean, which is responsible for low level XMPP communication using the smack XMPP library and which exposes only the required functionality, including key

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

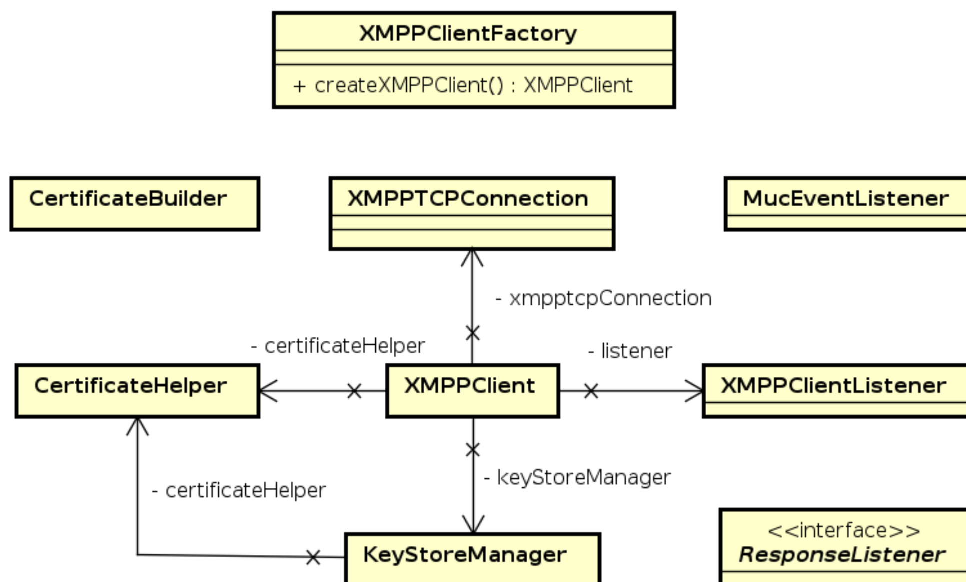

Figure 8.6: Class diagram of the XMPPClient bean related classes.

retrial, message listeners and group management functions, hiding the complexity of the XMPP protocol.

8.4.1 XMPPClient

The figure-8.6 shows the class diagram of the most important classes related to the XMPPClient bean. The XMPPClientFactory class is responsible for the creation of XMPPClient instances. The XMPPClient class represents an XMPP client facade and provides all high-level functionalities required by Smart-Toolbox, while hiding the complexity of all XMPP related operations. This is achieved using the XMPPConnection class, which is part of the smack library and which forwards all XMPP related logic, the CertificateHelper class, which is responsible for all certificate operations, and the KeyStoreManager class (explained in section 8.5), which stores the required certificates in a secure keystore. Additionally, The CertificateBuilder class handles the creation of certificates and the MucEventLister and XMPPClientListener classes are used for notifications in the context of multiuser chat and XMPP events.

The XMPPClient forwards all XMPP related functionality to the smack library, which then handles low level XMPP communication with the server, providing listeners for various notifications. The XMPPConnection class acts as a gateway to the XMPP network and provides synchronous and asynchronous methods for sending stanzas (see section 7.5 for more information). The XMPPClient client class manages the XMPPConnection instance and only exposes the disconnect and isConnected methods. Each XMPP operation, which relies upon a live connection, also verifies the current connection status

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

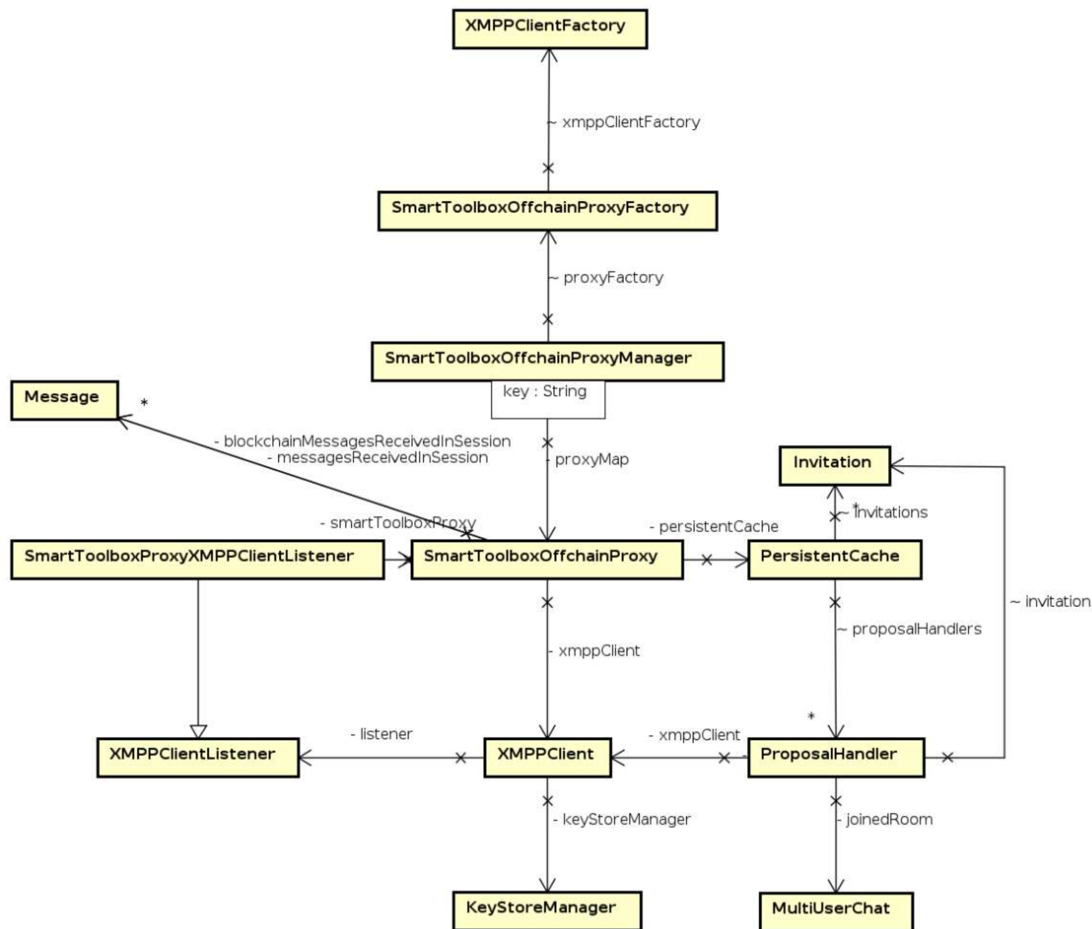

Figure 8.7: Class diagram of the SmartTolboxOffchainProxy bean related classes.

and automatically reconnects if necessary. This process can be seen in figure-8.8 (1.1.1.2). All multi user chat related operations are delegated to the MultiUserChatManager smack class. All contact list related operations are handled by the Roster smack class, and messages are forwarded using the ChatManager class. The figure-8.8 also shows how an anonymous XMPPClient instance is created. Such XMPPClient instances use the ANONYMOUS SASL authentication and do not require a username or password (for more information see 7.7.8). This is achieved using the performSaslAnonymousAuthentification (1.1.1.1.2) call on the smack Builder class. The creation of anonymous XMPPClient instances is implemented using a createAnonymousXMPPClient (1.1) method inside the XMPPClient class, because both XMPPClient classes share the same KeyStoreManager instance. Smart-Toolbox can use anonymous connections to further secure any multi user chat communication (see 8.4.4 for more information).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

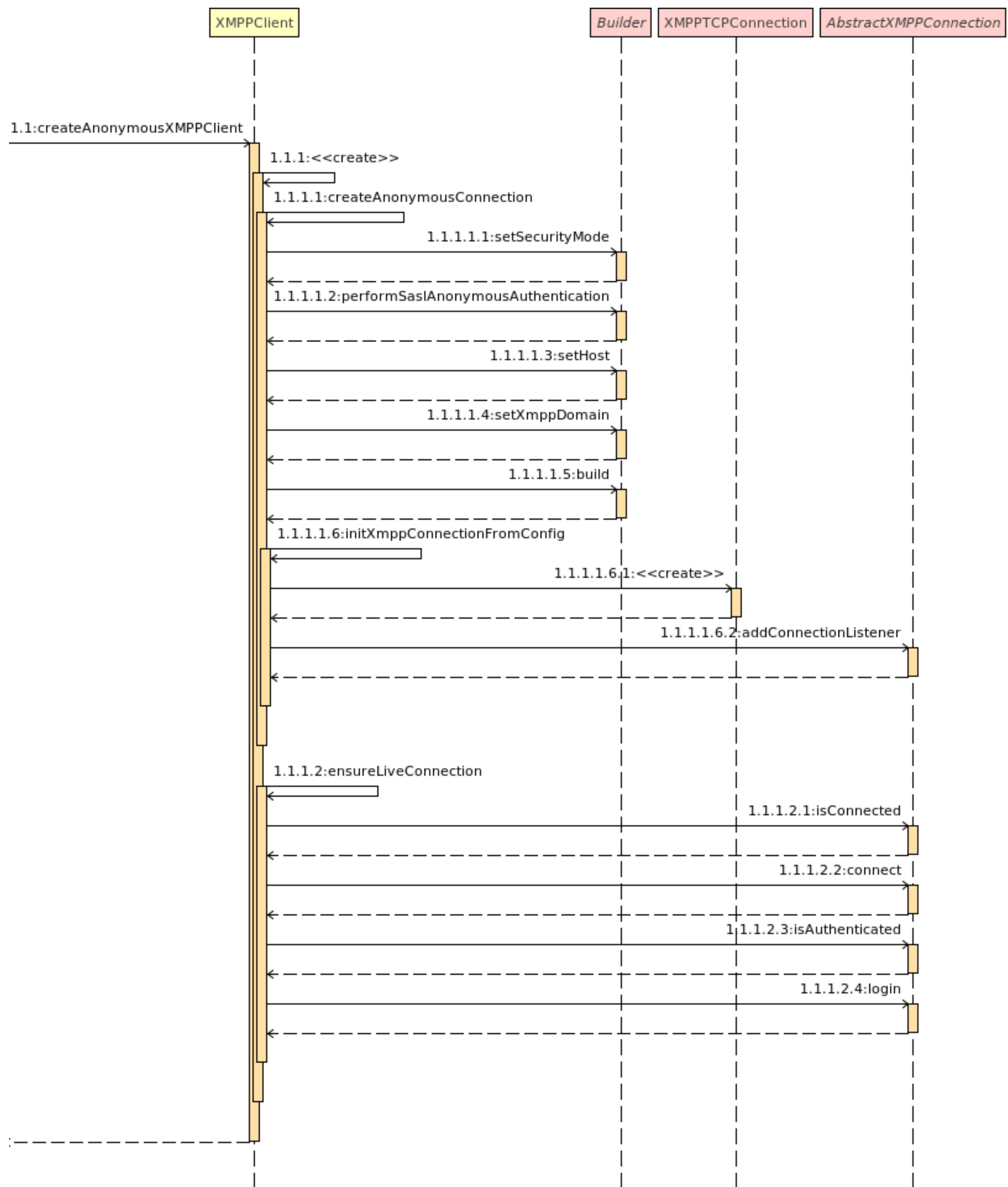

Figure 8.8: Sequence diagram showing the creation of anonymous XMPPClient instances and the automatic reconnection mechanism.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

8.4.2 SmartToolboxOffchainProxy

The figure 8.7 shows the class diagram of the most important classes related to the SmartToolboxOffchainProxy bean, which is responsible for all Off-chain related functions of Smart-Toolbox. This currently includes the messaging, membership service and private data distribution service, but will be extended in the future with a library distribution service as described in section 8.1. The SmartToolboxOffchainProxyManager class holds all instances of the proxy mapped using the username. This makes it possible to login with multiple users within one application instance. The SmartToolboxOffchainProxyManager is implemented as a spring service, ensuring only one instance is present in the Invertion Of Control (IOC) container. This service uses the SmartToolboxOffchainProxyFactory to create a new proxy, which, thanks to the XMPPClientFactory, also creates an XMPPClient instance responsible for all XMPP operations. Each SmartToolboxOffchainProxy contains a list of messages which have been received during the current session. We distinguish between standard messages and blockchain messages, which are used by Smart-Toolbox and should not be visible to the end user. The SmartToolboxOffchainProxy class also exposes a KeyStoreManager instance, which is currently stored in the XMPPClient instance as the underlying keystore is secured using the XMPP login password (see section 8.5 for more information). Each Invitation instance holds all invitation related data, including sender, credentials, channel and description. The ProposalHandler class represents a contract execution, grouping all users which accepted an invitation into one group. This class exposes proposal related methods, and in our XMPP based implementation uses a MultiUserChat instance for low level communication. The PersistentCache stores all Invitations and ProposalHandlers ensuring the current state survives application restarts. Additionally, a listener interface is used to notify other Smart-Toolbox components about new messages and invitations.

8.4.3 Event Listeners

The asynchronous nature of XMPP is usually handled using the listener pattern. Our example implementation combines the standard event listeners of the smack library, which are executed in the context of the current connection in the XMPPClientListener class, which is used as the last parameter in the XMPPClient constructor. This includes the ConnectionListener, IncomingChatMessageListener, OutgoingChatMessageListener and InvitationListener. All events which are related to a multi user chat are combined in the MucEventListener class, which has to be provided as a parameter in the createRoom and joinRoom functions. This includes the InvitationRejectionListener, MessageListener, PresenceListener, ParticipantStatusListener, SubjectUpdatedListener and UserStatusListener interfaces of the smack library. The SmartToolboxOffchainProxy class uses an instance of the SmartToolboxProxyXMPPClientListener, which extends the XMPPClientListener, forwarding messages and invites (handled in section 8.4.4) to the proxy.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

8.4.4 Invite System

Smart-Toolbox and community blockchains rely on the publish-invite-join principle for contract creation. This has been addressed in our XMPP based implementation of the SmartToolboxOffchainProxy bean, which is based on the multi user chat XMPP extension (See section 7.7.4 for more information). The identity of each participant in the chat room is hidden by a nickname. A multi user chat in XMPP might be private, protected by a password, members only, or open without any access control. Because of the privacy requirements of Smart-Toolbox an open room is not appropriate. A member only room also does not provide enough privacy as it is not possible to disable the member list query, which maps nicknames to real users. For this reason, the password protected room was chosen. Unfortunately. password protected rooms still do not prevent admin in the middle attacks, where the administrator identifies the authoring user. This can be mitigated using anonymous connections (see section 7.7.8 for more information), which can be activated using a setting in the SmartToolboxOffchainProxyFactory. The password and the room (or channel) name is part of each Invite instance, which also holds the sender, recipient, contract id and description.

Publish and Invite

The first step in the publish-invite-join principle is publishing the contract. In our implementation this means that a new chatroom has to be created. This is done in the createInvitation (1.3) method of SmartToolboxOffchainProxy, automatically hidden from Smart-Toolbox. The figure-8.9 shows a sequence diagram of this method, which requires a list of invited users, a nickname of the initiator, contract id and a designation as parameters. A user list can be created manually or as shown in the diagram using getAllRegisteredUsers (1.1) or getUserGroup (1.2) methods. The createInvitation method firstly creates a ProposalHandler (1.2.1), which will later be used for proposals operations, and generates a secure password (1.2.2). Then the XMPPClient class is used to create a password protected room, which is assigned to the returned ProposalHandler instance and used to send all invites from the provided list. Additionally, the PersistantCache is used to persist the new ProposalHandler.

Receive Invitation

All invitations are forwarded using the XMPP server as described in chapter 7. The figure-8.10 shows a sequence diagram for receiving invites. All invites are received by the smack library, which forwards the invite to an instance of the SmartToolboxProxyXMPP-ClientListener class (1.1). The listener parses the invite and creates a new Invitation instance, which is then forwarded to the SmartToolboxOffchainProxy (1.1.5) and persisted to the PersistentCache (1.1.5.1). The invitation is cached until it is accepted or declined by the user.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

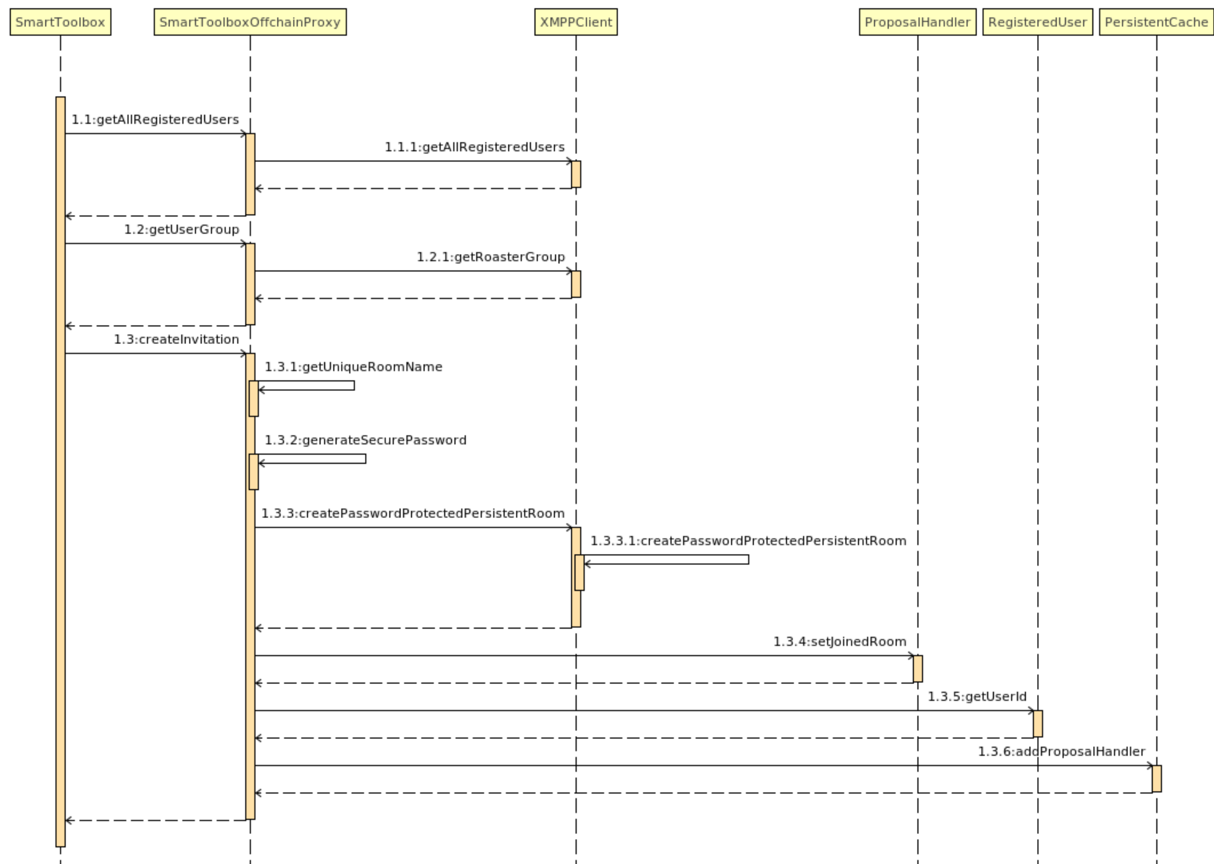

Figure 8.9: Sequence diagram showing the createInvitation method of the SmartToolbox-OffchainProxy bean.

Accept and Decline Invitation

The figure 8.11 shows the process of accepting and declining invitations. All invitations are stored in the PersistentCache and Smart-Toolbox can access them using the getInvitations (1.1) method. Then the acceptInvitation (1.2) method is called if a user chooses to accept an invite. The room name (1.2.1) and password (1.2.2) are retrieved from the invitation and used in the joinRoom (1.2.3) method. This creates a new MultiUserChat instance, which is then assigned to a new ProposalHandler. The invitation is removed from PersistentCache and the ProposalHandler is stored to the PersistentCache.

The declineInvitation (1.3) method is called if a user chooses to decline an Invitation, which is then removed from the PersistentCache and declined using the XMPPClient instance. Because the current implementation uses a password protected room it is still technically possible to join the room if the password is known.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

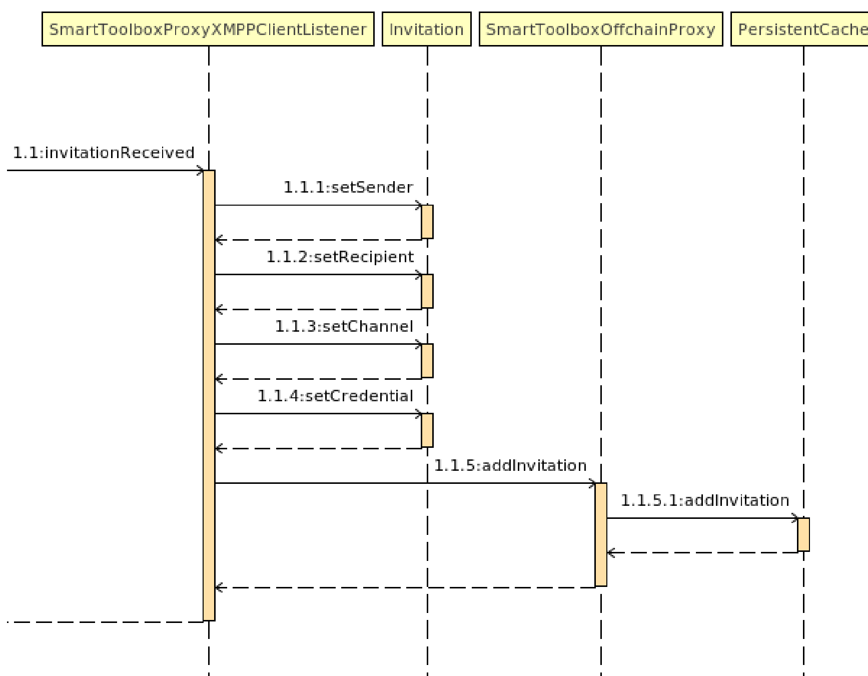

Figure 8.10: Sequence diagram showing how a invitation is propagated to the SmartToolboxOffchainProxy.

8.4.5 Proposal Send And Receive

All communication within a contract group is handled using a ProposalHandler instance, which within the context of our XMPP based implementation represents one chat room. A ProposalHandler instance can only be created by creating or accepting an invitation (see section 8.4.4 for more information). The figure 8.12 describes how a proposal is propagated to the chatroom and thus to all contract participants. First, the ProposalHandler ensures the chatroom connection is valid and reconnects using the stored invitation if necessary. This ensures a stored ProposalHandler in PersistentCache is still able to send proposals after application restart. Second, the sendMessage method is called on the MultiUserChat instance sending the proposal to the chatroom.

The figure-8.13 shows how a proposal is received. The smack library receives a new chat room message and forwards it to the ProposalHandlerMucEventListener, which takes the message body (1.1.1) and stores it to the ProposalHander (1.1.2).

Each ProposalHander has to be destroyed after the contract is finished. This is achieved using the destroyProposalHandler (1.1) method of SmartToolboxOffchainProxy shown in the figure 8.14. The method destroys the XMPP chat room (1.1.2) and removes the ProposalHandler from the PersistenCache (1.1.3).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

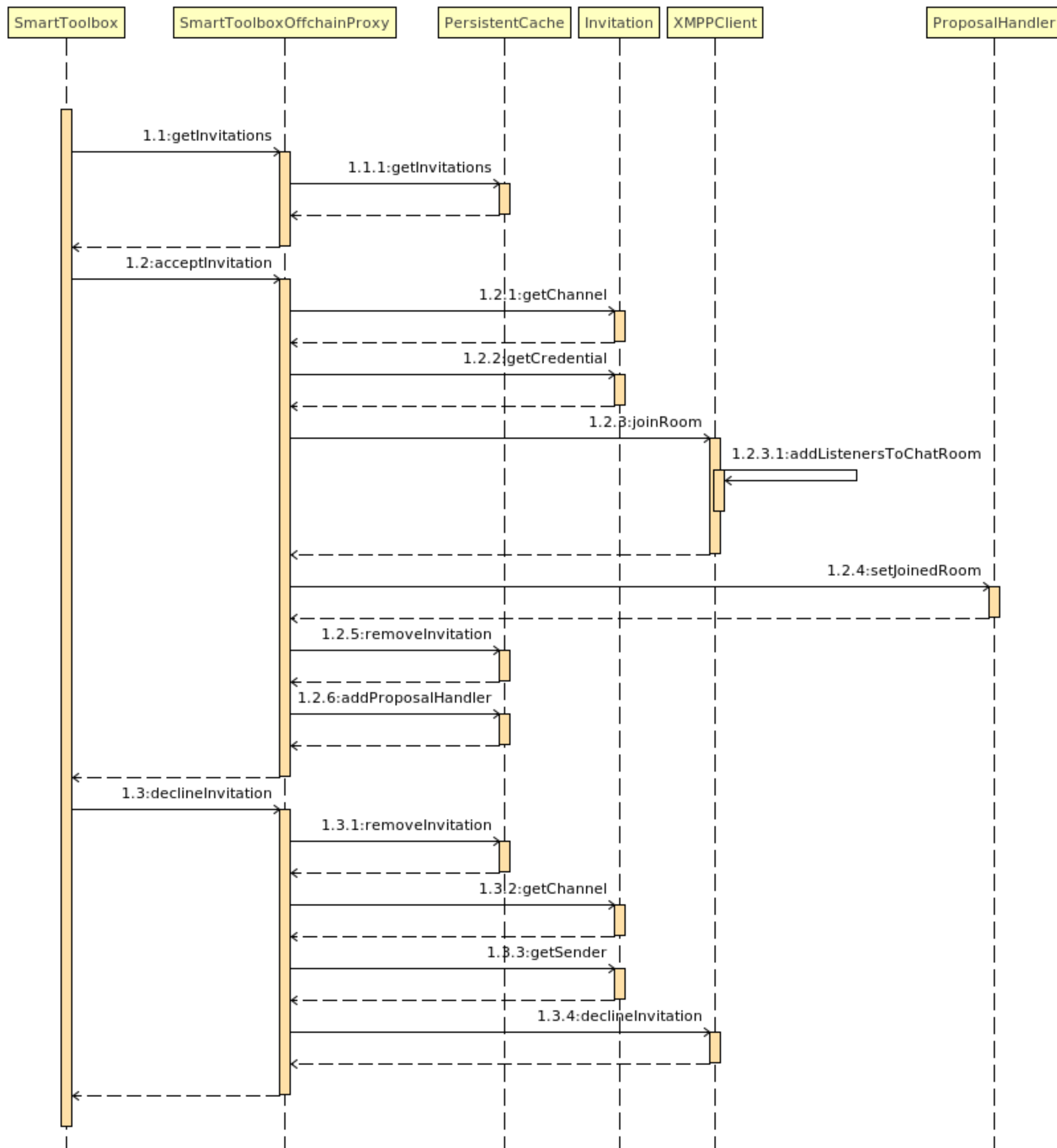

Figure 8.11: Sequence diagram showing how an invitation can be accepted or declined.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

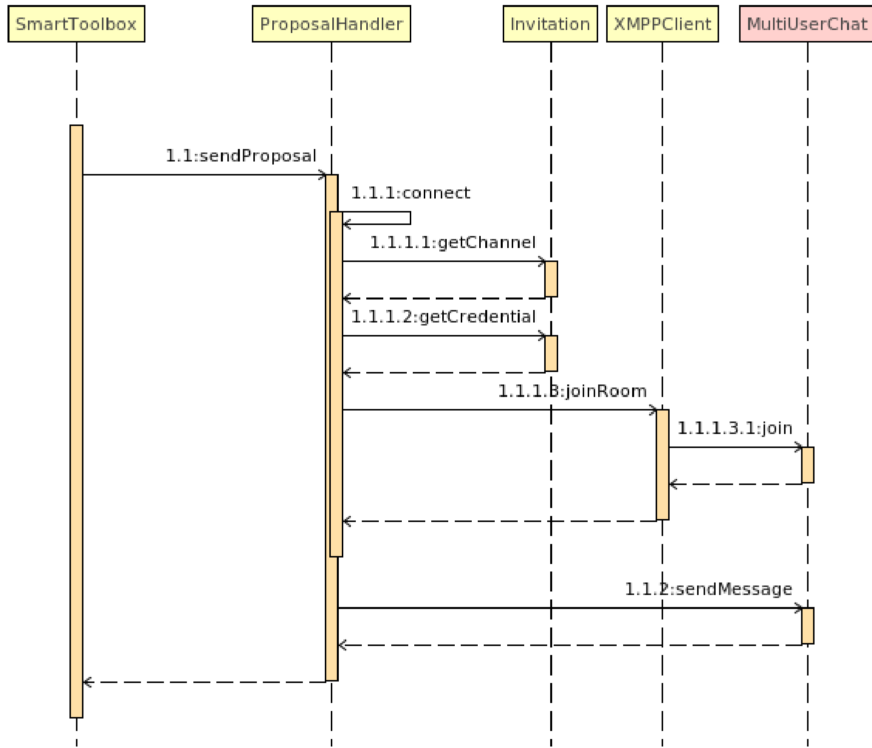

Figure 8.12: Sequence diagram showing how a proposal is send to the multi user chat.

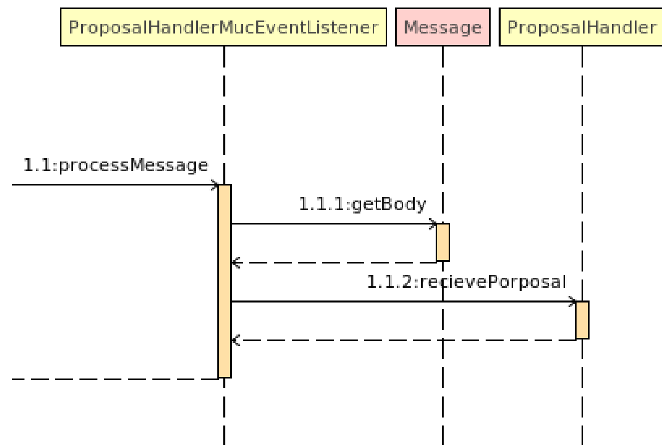

Figure 8.13: Sequence diagram showing how a proposal is received from the multi user chat.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

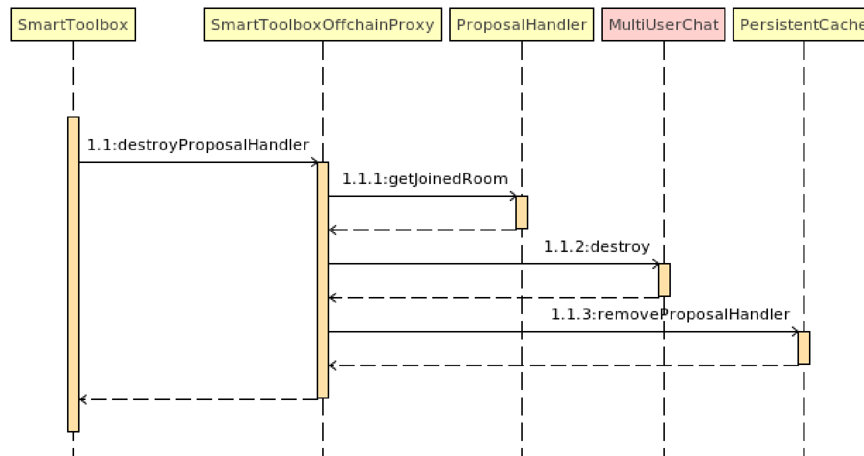

Figure 8.14: Sequence diagram showing the destruction of a ProposalHandler.

8.4.6 Group Management

The SmartToolboxOffchainProxy class provides contact group management functions, which allow the creation and retrieval of groups referenced by a group name. Smart-Toolbox expects that the membership service provides user group support controlled by an authorized user (administrator). Our XMPP based implementation is based on roster groups (see section 7.7.7 for more information). This makes it possible to have a separate contact list for each user, which can add or remove its own contacts and, in combination with contact list sharing provided by the openfire server, have shared groups controlled by an administrator. As you can see in figure 8.15, a UserGroup instance can be created using the createUserGroup (1.1) method, which also creates a new RosterGroup if it does not exist (1.1.1), or getUserGroup (1.3) methods. The UserGroup class exposes the getUsers (1.4) method, which takes all RosterGroup entries (1.4.1) and creates a corresponding RegisteredUser class and the addUser method, which stores a new roster entry to a RosterGroup, while returning a corresponding RegisteredUser instance.

8.4.7 Key Retrieval and Certificate Signing Request

The XMPP standard and its extensions cover most of the communication requirements which are required by Smart-Toolbox. This includes group management and direct messaging, but excludes the distribution of cryptographic material. Although the OpenPGP extension discussed in section 7.7.11 could be used for public key sharing it is not possible to sign a Certificate signing request (CSR) as required by our proof of concept implementation. For this reason, a custom plugin has been implemented. We have defined the following custom iq stanzas (See section 7.5 for more information) to extend XMPP with the required features:

- The CsrIQ get iq stanza is used in the certificate signing process. The namespace is custom:iq:Keyproc and the stanza includes the CSR requests as payload.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

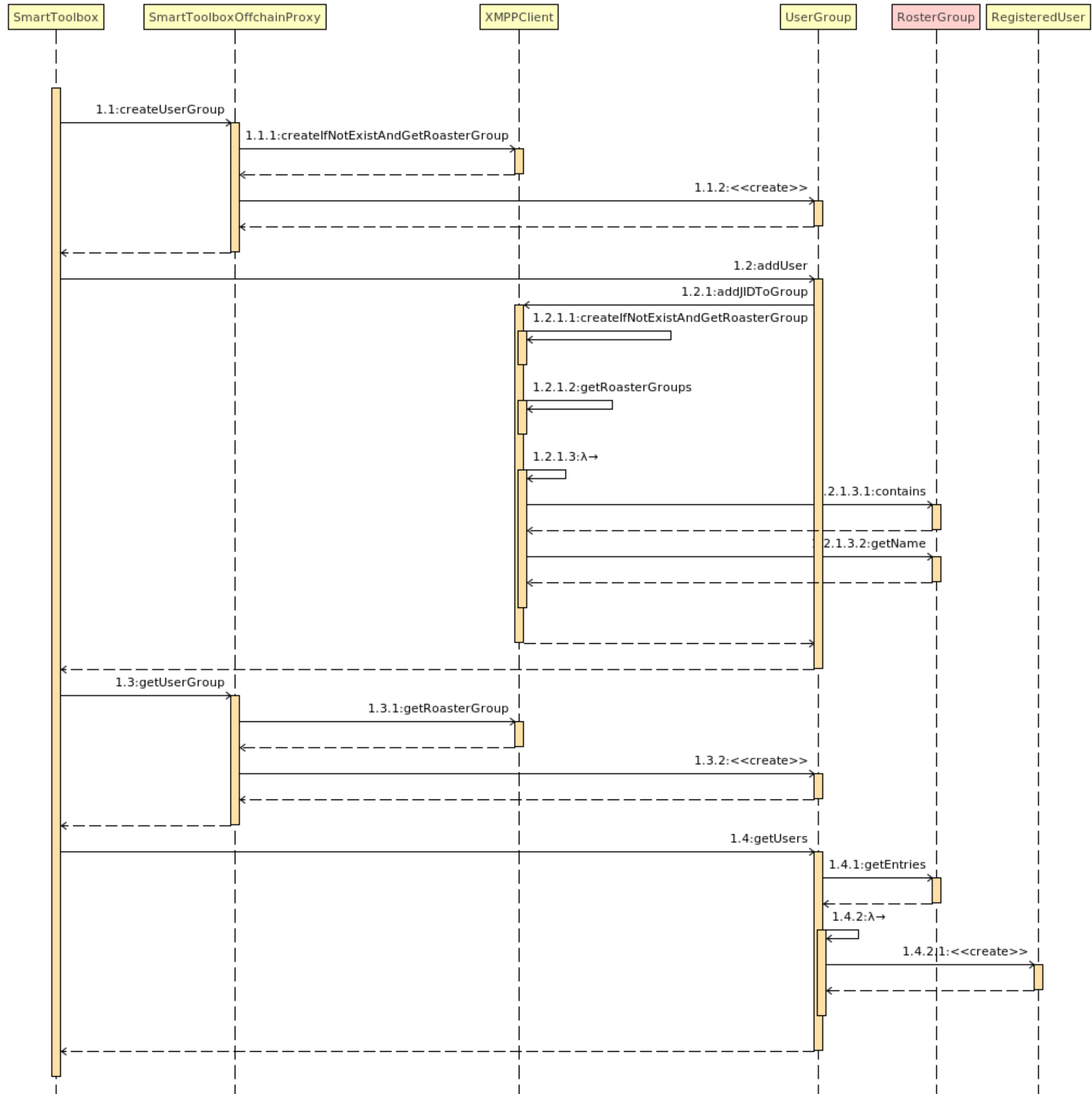

Figure 8.15: Sequence diagram showing group management related methods.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

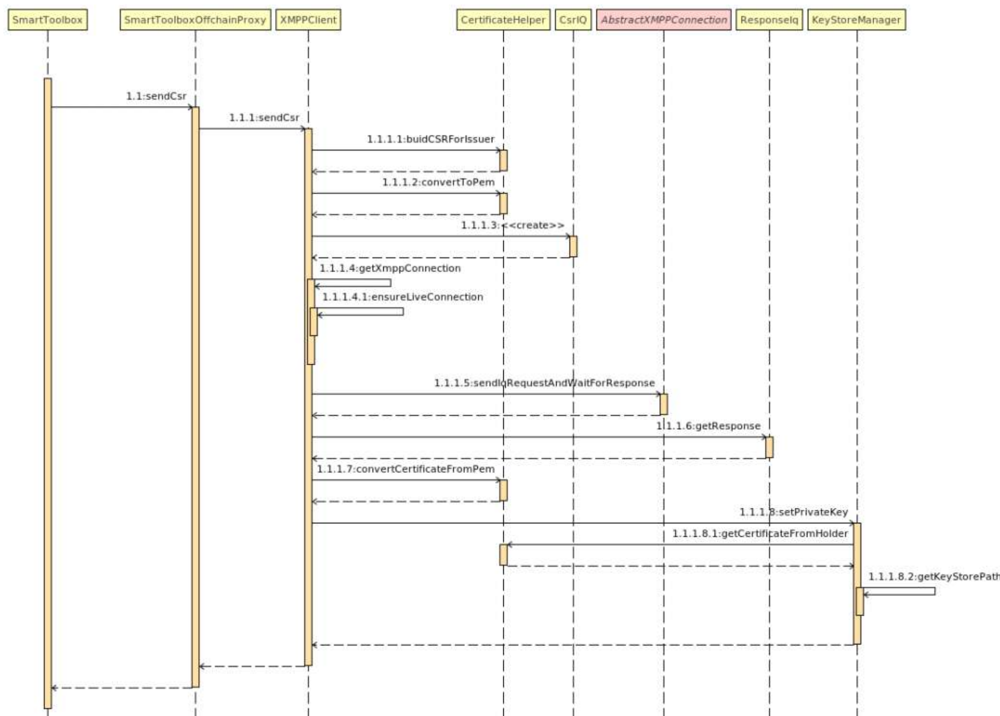

Figure 8.16: Sequence diagram showing the CSR process.

- The GetPublicKeyIQ get iq stanza is used in the public key query process. The namespace is custom:iq:Keyproc.
- The ResponseIq set iq stanza is used for query response and encodes the result of a given query. The namespace is custom:iq:smarttoolbox.

The client side CSR process execution can be seen in figure 8.16. The sendCsr (1.1) method of the SmartToolboxOffchainProxy forwards the request to the XMPPClient instance (1.1.1), which uses the CertificateHelper class to build a CSR (1.1.1.1) and converts it to the pem format (1.1.1.2). The result is applied to a new CsrIQ instance, which is synchronously sent to the server. The server signs the request using the domain certificate and returns the resulting certificate in pem format. The certificate is then stored to the keystore using the KeyStoreManager class (1.1.1.8).

As you can see in figure 8.17, the SmartToolbox class exposes the getPublicKeyFromServer (1.1) method, which takes a key id as a parameter. There are no rules for the Id so it is even possible to retrieve keys, which are not related to a user but are used in other parts of the system. The request is forwarded to the XMPPClient class, which creates a GetPublicKeyIQ (1.1.1.1) stanza. This stanza is then synchronously sent to the XMPP server (1.1.1.2), returning a ResponseIq containing the key in pem format.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

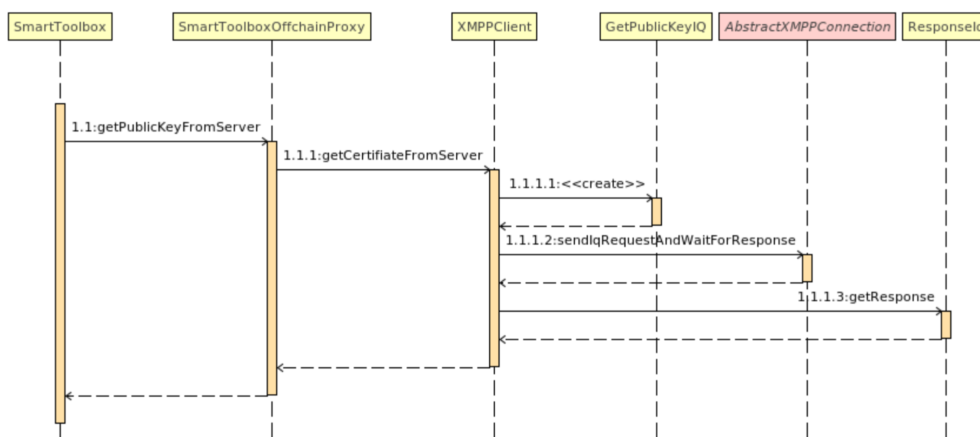

Figure 8.17: Sequence diagram showing the key retrieval from server.

8.4.8 Login and Initialization

This subsection describes how the SmartToolboxOffchainProxy is created, how the XMPP connection is created, and how the initialization process ensures that all ProposalHandlers are reinitialized and updated.

SmartToolboxOffchainProxyManager

The SmartToolboxOffchainProxyManager class holds a map of all authorized instances of the SmartToolboxOffchainProxy class. The figure-8.20 shows a sequence diagram describing the getProxy and createProxy methods. Each instance can be retrieved using the getProxy (1.2) method, and the key for retrieval is the login user name which was used to create the given instance. The createProxy (1.1) method takes the username, password, and server address as arguments and uses the SmartToolboxOffchainProxyFactory to create a new SmartToolboxOffchainProxy instance (1.1.1.1). This method creates an instance of the XMPPClient class using the XMPPClientFactory (1.1.1.1.1.1), initializes the PersistentCache (1.1.1.1.1.2), and reconnects all ProposalHandler instances (1.1.1.1.1.3) which were stored in the PersistentCache.

XMPP Login

The login process can be seen in figure 8.18. The XMPPClientFactory class contains a static method (1.1.1.1), which creates an XMPPClient instance for the given username, a password and an XMPP server hostname. The constructor creates a self signed certificate (1.1.1.1.1.1), which is used for xmpp encryption and builds an XMPPTCPConnection. Then the ensureLiveConnection (1.1.1.1.2) method is called, which logs the user into the XMPP server. Additionally, a KeyStoreManager is created using the credentials provided in the constructor.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

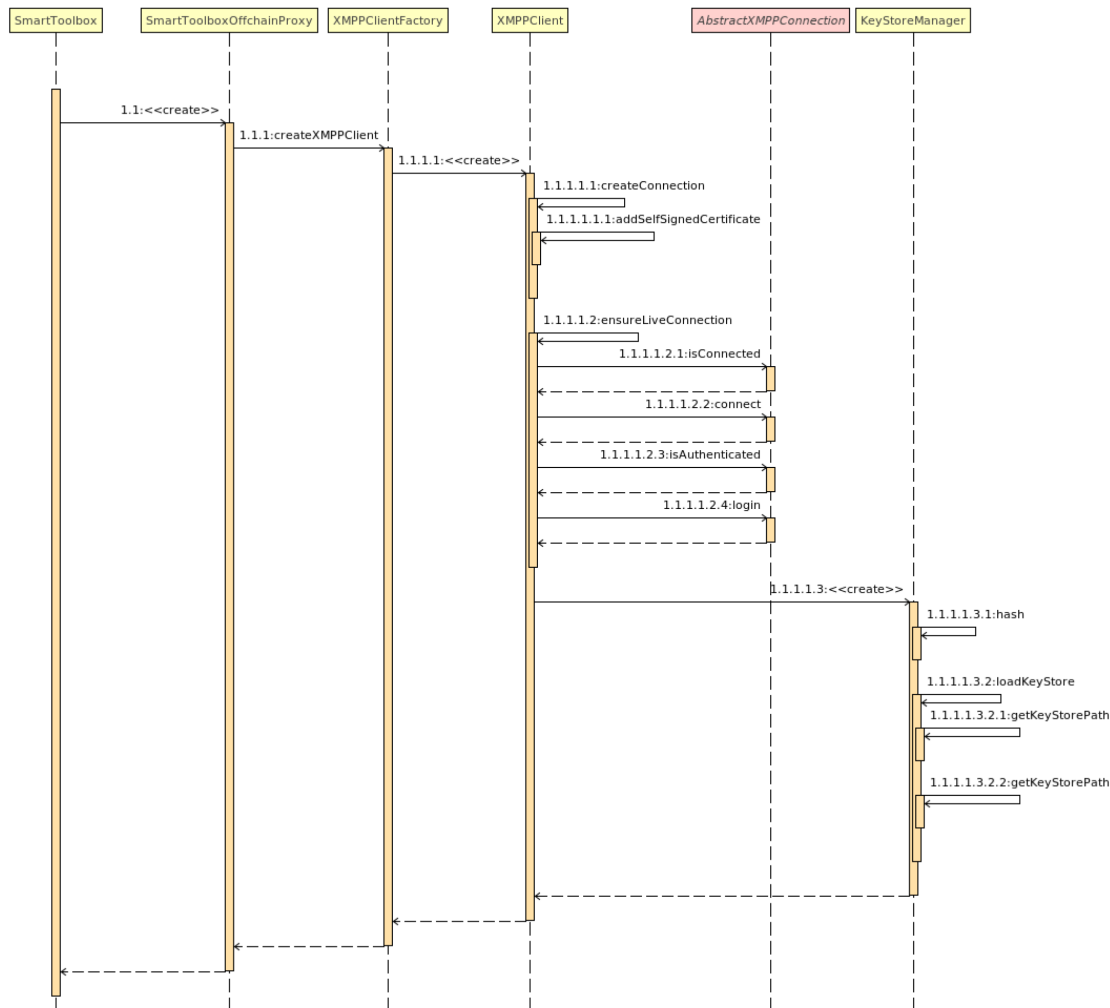

Figure 8.18: Sequence diagram showing the login process.

All methods exposed by XMPPClient which depend on a live XMPP connection will automatically reinitialize a disconnected or broken connection using the ensureLive-Connection (1.1.1.1.2) method and effectively re-login if necessary. This mechanism is hidden from Smart-Toolbox and done automatically.

ProposalHandler Reconnect

The figure 8.19 shows the reconnection process of ProposalHandler instances (1.1.3). PersistanceCache stores all data in a database, which ensures data consistency across application restarts. The first step is to retrieve all stored ProposalHandler instances from the PersistanceCache (1.1.3.1), which are then reconnected using the connect (1.1.3.2) method. This is achieved using the Invitation instance stored as a field in each ProposalHander. The Invitation holds the channel (room) and credentials (password),

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

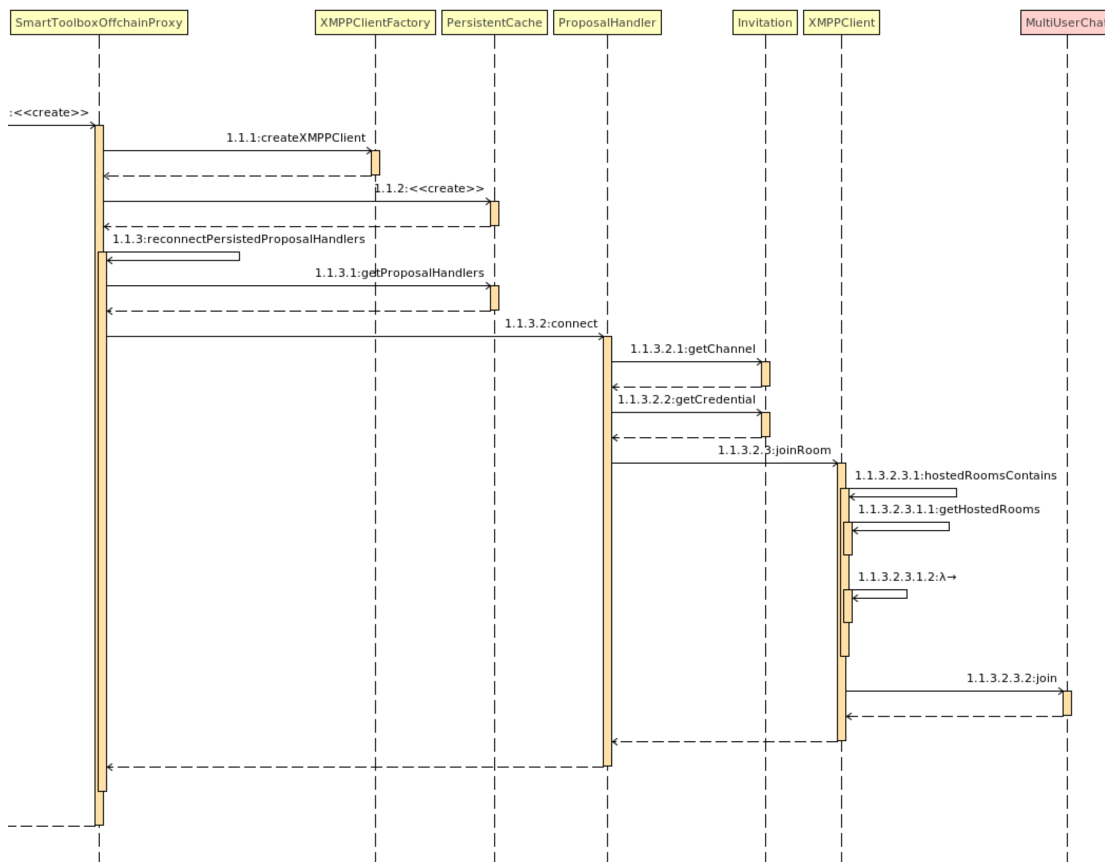

Figure 8.19: Sequence diagram showing the reconnection of ProposalHandlers during the initialziation process.

which are used to join the multi user chat room (1.1.3.2).

8.4.9 Support for synchronous Administrative Commands (RPC)

Our implementation provides a send action function as shown in figure 8.21. This mechanism allows the synchronous execution of custom commands using the XMPP protocol and is defined as follows. The sending client invokes the sendAction method providing an action name, list of parameters and a callback method (1). The XMPPClient instance then sends the request using the SendActionIQ stanza (1.1). The XMPP server then forwards the iq stanza to the target client and the smack library executes the handleIQRequest method in the XMPPClient (3). the SendActionIQ stanza is then recognized and the newIncomingAction callback is invoked (3.1). The smart-toolbox application can then execute the action with the given parameters. The value returned in the newIncomingAction callback is then encoded and returned to the sending client as a payload in a ResponseIq stanza. The sending client invokes the reciveResponse callback (2.1) after the ResponseIq is received (2). This shows that it is possible to implement an

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

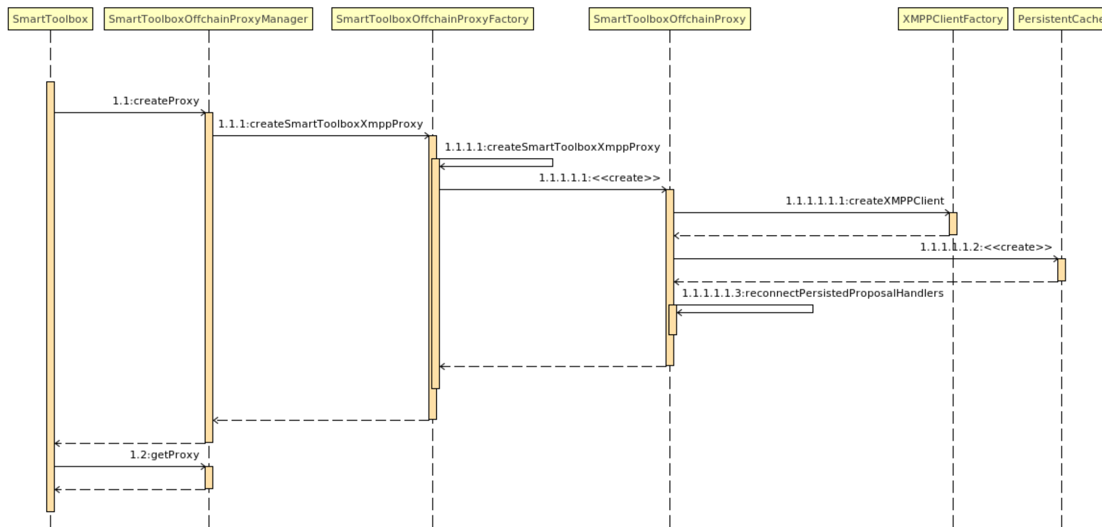

Figure 8.20: Sequence diagram showing the initialization process.

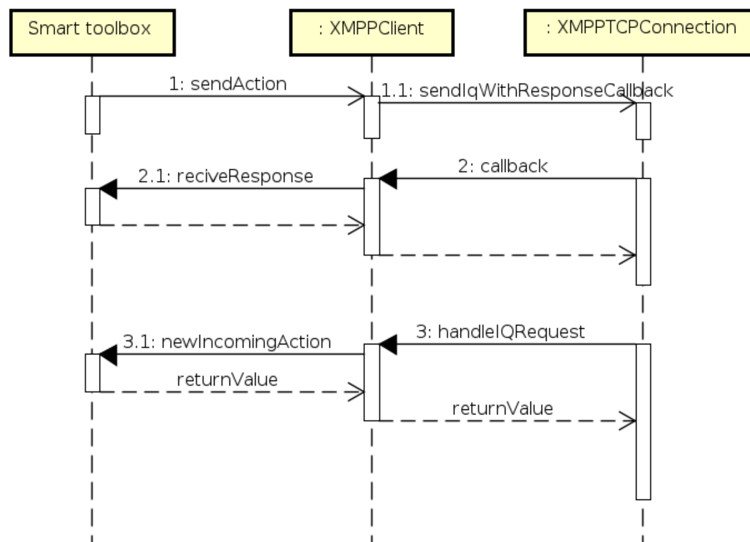

Figure 8.21: Sequence diagram showing how an action can be sent between participants.

RPC mechanism using XMPP.

8.5 Key Management

This section summarizes all key management related logic implemented in the course of this thesis. The server side plugin is responsible for the signing of CSR requests and provides a key based storage of signed certificates. The server uses the openfire domain

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

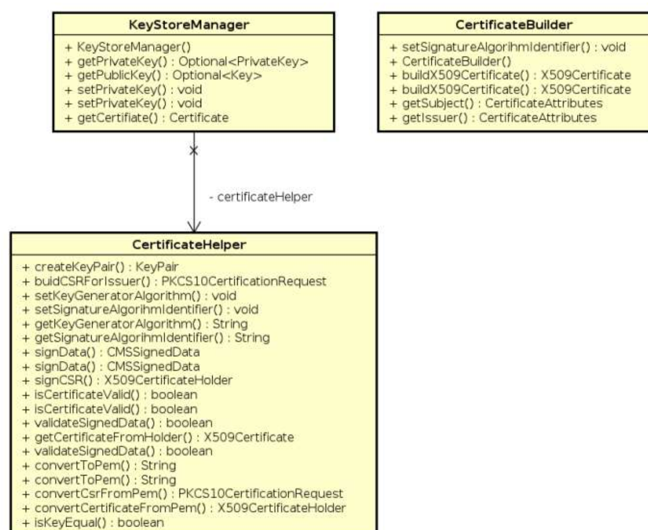

Figure 8.22: Class diagram of key management related classes.

certificate for certificate signing and a local LevelDB instance, which ensures all signed certificates are not lost after the server restarts. The client server communication related to this feature is handled in section 8.4.7.

The figure 8.22 shows the class diagram of all cryptography related classes. The CertificateBuilder is responsible for the creation of new certificates and its implementation follows the builder design pattern. The CertificateHelper class holds all low level cryptographic operations and the KeyStoreManager handles the storage and retrieval from the keystore. The CertificateHelper and CertificateBuilder are both strongly based on the bouncycastle library. Each XMPPClient bean has a KeyStoreManager field, which is initialized by the constructor or, in the case of anonymous XMPPClient instances, provided in the constructor parameters (see section 8.4.1 for more information). The figure-8.23 shows how a keystore manager instance is created. The constructor (1.1) has the following parameters: keyStoreSuffix, password and pepper. The keyStoreSuffix is appended to the key store file name so that it is possible to distinguish key stores on the file level. The password and peper parameters are used to unlock the keystore. The current implementation uses the xmpp password for the password parameter and the user name provided during the login as the keyStoreSuffix and pepper parameters. The pepper and password parameters are concatenated with the salt field, creating an input string used in the hash (1.1.3) function, which repeats the SHA-512 hash 1000 times. This process takes about 3 seconds on i7-9750H and its purpose is to prevent dictionary based attacks. This also ensures that the password is not stored in memory in plain text format and cannot be read using a debugger. Then the loadKeyStore (1.1.4) function resolves the key store path (1.1.4.1) and loads all keys to memory (1.1.4.3).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

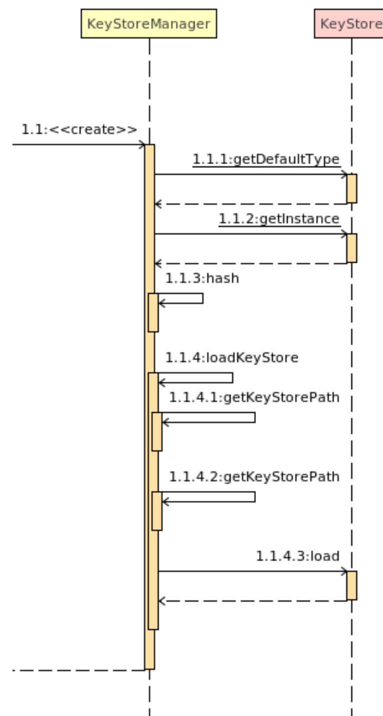

Figure 8.23: Sequence diagram showing the initialization of a KeyStoreManager instance.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 9

Evaluation

This chapter evaluates the features of our XMPP based implementation of Off-chain services, which are required by permissioned private blockchains and compares them to related systems presented in chapter 5. The evaluation is based on the requirements presented in chapter 6.

9.1 Private Data Distribution

The system should be able to distribute private data to authorized users.

- Corda distributes private data automatically between nodes during flow execution. This is achieved using the AMQP protocol in combination with TLS (see section 5.1 for more information).
- Hyperledger Fabric distributes private data using the gossip protocol, which is a peer to peer protocol based on epidemic multicast (see section 5.2 for more information).
- Multichain stores private data directly on the blockchain, secured by cryptography. Privacy is ensured using three streams (see section 5.3 for more information).
- Tendermint provides an X25519 secured STS protocol, which can be used to distribute private data (see section 5.4 for more information).
- Quorum uses Constellation or Tessera for private data distribution. Both systems are peer to peer based and besides private data distribution also handle key management and synchronization of new nodes (see section 5.5 for more information).
- Our implementations provide the distribution of private data using the Multi User Chat extension (see section 7.7.4 for more information). Private data is distributed using chat room messages, which are visible only to authorized users.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

9.2 Offline Capability

All messages transferred within the communication system should be delivered even if the recipient is offline or not reachable due to network outages.

- The AMQP protocol employed in corda allows offline operations and automatic re-synchronization after reconnect (see section 5.1 for more information).
- Hyperledger Fabric uses gRpc, which provides only direct communication without offline support (see section 5.2 for more information).
- Multichain is based on the bitcoin core and does not provide Off-chain messaging (see section 5.3 for more information).
- Tendermint uses JSONRPC or URI/HTTP, which does not support offline operations [ten20b].
- Quorum uses Constellation or Tessera for general purpose messaging. Neither supports offline communication [Con20a].
- Our implementations ensures offline capability for group messages using Multi User Chat room history (see section 7.7.4 for more information).

9.3 Direct Messages

The message system should facilitate direct message exchange between participants.

- Corda uses AMQP, which supports direct messaging [Fou21].
- The gRPC library employed in Hyperledger Fabric allows direct message exchange using RPC [Fou20].
- Multichain is based on the bitcoin core and does not provide Off-chain messaging (see section 5.3 for more information).
- Tendermint uses JSONRPC, which is an RPC mechanism allowing direct message exchange [ten20c].
- Quorum uses Constellation or Tessera for general purpose messaging, allowing direct messages [Con20a].
- Our implementation relies on XMPP, which provides direct message support as basic functionality.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

9.4 Message Broadcast

The message system should be able to send the same message to a defined set of recipients at once.

- Corda uses AMQP, which uses the publish subscribe pattern for message broadcast [Fou21].
- The gRPC library employed in Hyperledger Fabric does not support message broadcast (see section 5.2 for more information).
- Multichain is based on the bitcoin core and does not provide Off-chain messaging (see section 5.3 for more information).
- Tendermint uses JSONRPC, which provides message broadcast in websocket mode [ten20c].
- Quorum uses constellation or Tessera for general purpose messaging, allowing message broadcast [Con20a].
- Our implementations rely on the Multi User Chat room messaging system (see section 7.7.4), which ensures that all room messages are delivered to all reachable participants.

9.5 Synchronous Messages

The system should support synchronous message delivery.

- Corda uses AMQP, which provides a synchronous message api [Fou21].
- The gRPC library employed in Hyperledger Fabric allows for synchronous RPC calls (see section 5.2 for more information).
- Multichain is based on the bitcoin core and does not provide Off-chain messaging (see section 5.3 for more information).
- Tendermint uses JSONRPC, which supports synchronous messages [ten20c].
- Quorum uses Constellation or Tessera for general purpose messaging, although neither provide synchronous communication [Con20b], [CON21].
- Our implementation relies on XMPP, which provides synchronous communication using iq stanzas (see section 7.5.1 for more information).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

9.6 Asynchronous Messages

The system should support asynchronous message delivery.

- Corda uses AMQP, which provides an asynchronous message api [Fou21].
- The gRPC library employed in Hyperledger Fabric allows for asynchronous RPC calls (see section 5.2 for more information).
- Multichain is based on the bitcoin core and does not provide Off-chain messaging (see section 5.3 for more information).
- Tendermint uses JSONRPC, which supports asynchronous messages in websocket mode [ten20c].
- Quorum uses Constellation or Tessera for general purpose messaging and both support asynchronous communication [Con20b], [CON21].
- Our implementation relies on XMPP, which provides asynchronous messaging using message stanzas (see section 7.5.2 for more information).

9.7 Group Support and Dynamic Group Creation

The system should be able to create a group for private data exchange.

- Corda supports dynamic group creation by authorized members, which are able to create and manage business networks [Ltd20c].
- Hyperledger Fabric provides group functionality using channels, which are overlay blockchains with separate consensus. Fabric channels can only be created by a system administrator and are not dynamic [ABB+18].
- Multichain hides the complexity of blockchains into streams, which also provide group functionality. Depending on configuration either every participant or participants with a special permission can create a stream dynamically [Ltd20a].
- Tendermint does not provide a group concept [ten20b].
- Quorum provides group functionality on a transaction level using a 'private for' parameter containing a list of cryptographic keys [Con20a].
- Our implementation uses the Multi User Chat room messaging system as a group system (see section 7.7.4 for more information), which provides an invite join mechanism and private communication.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

9.8 User Query

The system should be able to list all registered users, which can then be invited.

- Corda provides Accounts, which might be used to split the corda vault into logical sub-vaults [Ltd20b].
- Hyperledger Fabric provides a Membership Service, which holds all registered users [ABB+18].
- Multichain provides a function to list all addresses associated with a node [DGG20].
- Tendermint does not provide user support [ten20b].
- Quorum supports account plugins providing user listing support [Con20a].
- Our implementation relies on XMPP, which provides a user registry and group management system (see section 7.7.7 for more information).

9.9 User Groups

The system should be able to group users into logical groups (for example departments). This will allow the invitation of multiple users at once.

- Corda currently does not support account grouping [Ltd20b].
- Hyperledger Fabric can organize users into groups [ABB+18].
- Multichain does not support user grouping [DGG20].
- Tendermint does not provide user support [ten20b].
- Quorum does not support groups [Con20a].
- Our implementation relies on XMPP, which provides a user registry and group management system (see section 7.7.7 for more information).

9.10 Cryptography Storage

The system should be able to store Cryptographic material.

- Corda provides an identity service implementation, which provides an identity directory and stores cryptographic keys (see section 5.1 for more information).
- Hyperledger Fabric provides a membership service, which handles key management and can act as CA [ABB+18].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

- Multichain does not provide an Off-chain mechanism for key management as all keys and permissions are stored On-chain using special transactions [Ltd20a]
- The Tendermint project provides a key management service called tmkms [ten20b].
- Quorum uses Constellation or Tessera for general purpose messaging. Both of these systems also provide management, storage and distribution of cryptographic keys (see section 5.5 for more information).
- Our implementation relies on an Openfire plugin, which provides key management, storage and distribution of cryptographic keys.

9.11 Non-Functional Requirements

All non-functional requirements defined in chapter 6, including performance, scalability, availability, encryption and security are essential for every blockchain and are met by every reference system. Our implementation relies on the features provided by XMPP. If not configured otherwise XMPP connection uses TLS to encrypt communication and a password protected keystore for private key storage. Performance, scalability, availability were out of scope of this thesis and are handled in section 10.2.

9.12 Disadvantages of XMPP

The major disadvantage of XMPP in the context of blockain systems is its centralized nature, which even with clustering creates a central point of failure in a decentralized system like a blockchain.

Another problem is that the private data transferred over XMPP might be different from the data referenced from the blockchain. The validity of this data has to be checked on the application level.

Additionally, privacy can be jeopardized by an administrator, who has read access to the server and can read the message author and recipients.

9.13 Summary

The figure-9.1 shows the requirement fulfilment of our implementation and the requirement fulfilment of different support systems employed in compared private permissioned blockchains. Our implementation fulfils all requirements defined in chapter 6, which shows the suitability of XMPP for Off-chain messaging.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

	Corda	Hyperledger Fabric	Multichain	Tendermint	Quorum	XMPP
Private Data Distribution	•	•	•	•	•	•
Offline Capability	•	◦	N/A	◦	◦	•
Direct Messages	•	•	◦	•	•	•
Message Broadcast	•	◦	◦	•	•	•
Synchronous Messages	•	•	N/A	•	◦	•
Asynchronous Messages	•	•	N/A	•	•	•
Group Support	•	•	•	◦	•	•
Dynamic Group Creation	•	◦	•	◦	•	•
User Query	•	•	•	◦	•	•
User Groups	◦	•	◦	◦	◦	•
Cryptography Storage	•	•	N/A	•	•	•

Table 9.1: This table shows the feature comparison table for different private blockchain systems and ourXMPP based implementation.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 10

Conclusion and Future Work

10.1 Conclusion

This thesis analyzed the suitability of the XMPP communication system for Off-chain services, which includes key management, private data exchange and dynamic mutable group creation. We have analyzed existing Off-chain support services employed in various private permissioned blockchain systems (see chapter 5 for more information) and used the results to define a list of requirements, which is summarized in chapter 6. This list was used to define a suitable architecture of our proof of concept implementation and later to evaluate the results.

Our proof of concept implementation relies on basic XMPP features, especially the roster (see section 7.7.7 for more information), which provides user management and grouping, and the Multi User Chat extension (see section 7.7.4 for more information), which provides an invite join mechanism, private asynchronous message broadcast and history. These standard extensions fulfill most of the defined requirements including private data distribution and offline capability. XMPP defines a standard extension for public key discovery (see section 7.7.11 for more information), but it does not support CSR and is in a deferred state. Therefore, the key management and cryptography storage has been implemented using a custom extension in the form of a plugin, which defines custom stanzas for CSR and public key retrieval (see chapter 8.4.7 for more information). This shows that XMPP is highly extendable and can be modified as needed, but also provides all required functionality to be suitable for Off-chain services. Unfortunately, XMPP is a centralized system based on a client-server architecture and the XMPP server creates a central point of failure, which contradicts the distributed nature of blockchains.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

10.2 Future Work

The aim of this thesis was to analyze the suitability of XMPP for Off-chain services including private data exchange and cryptographic storage. This section presents opportunities for future research, which were omitted from this thesis because of low importance to the suitability analysis.

10.2.1 Library Distribution Service

The contract implementation and other dependent libraries evolve over time and need to be deployed on the network nodes regularly. Therefore an update mechanism, which ensures all nodes are using the same version, is needed. This might be implemented using custom stanzas (see section 7.5 for more information) in combination with the file transfer extension (see section 7.7.3 for more information), or with an update repository.

10.2.2 Security

Security was addressed in this thesis only partially. The cryptographic keys are protected in a password protected keystore, and the communication channels are protected using TLS encryption preventing man in the middle attacks (see section 7.6 for more information), but there is a possibility of an admin in the middle attack, where an administrator accesses protected data. This is partially mitigated using anonymous connections (see section 7.7.8 for more information), which hide the author of a given message even from an admin, but it is still possible to access the openfire database for message content. This can be mitigated using additional encryption layers.

10.2.3 Performance

The performance of our implementation was not tested as it is out of scope for this thesis. We expect that the performance is reasonable as the system relies on a proven XMPP implementation, but this needs to be verified using a stress test.

10.2.4 Availability

Availability can be achieved using clustering, which is supported by the openfire server. Our plugin was not tested in this context and it might be needed to modify the current implementation.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

List of Figures

2.1 A chain of blocks. This figure is a reprint from [Nak19]. 5
2.2 Transaction workflow of a blockchain. This figure is a reprint from [Nak19]. 6

3.1 Permissioned blockchain systems referenced in recent literature 11
3.2 Fabric network, this figure is a reprint from [ABB+18]. 14
3.3 Quorum architecture, this figure is a reprint from [Cha20]. 17

7.1 Distributed client server architecture. This figure is based on [SA11a]. . . 30
7.2 Bidirectional communication using two streams. This figure is based on [SA11a]. 33
7.3 Iq stanza example. This figure is based on [SA11a]. 35
7.4 Example of a message stanza. This figure is based on [SA11a]. 35
7.5 Example of a presence stanza. This figure is based on [SA11a]. 36
7.6 Example of a stanza error. This figure is based on [SA11a]. 37
7.7 Xep life cycle. This figure is based on [SAC10]. 38

8.1 Smart-Toolbox application environment. This figure is a reprint from [sma20]. 45
8.2 Contract layout. This figure is a reprint from [sma20]. 46
8.3 Abstract blockchain architecture, this figure is a reprint from [sma20]. . . 47
8.4 Smart-Toolbox application environment. This figure is a reprint from [sma20]. 48
8.5 Deployment diagram of the XMPP integration implemented over the course of this thesis. 51
8.6 Class diagram of the XMPPClient bean related classes. 52
8.7 Class diagram of the SmartTolboxOffchainProxy bean related classes. . . 53
8.8 Sequence diagram showing the creation of anonymous XMPPClient instances and the automatic reconnection mechanism. 54
8.9 Sequence diagram showing the createInvitation method of the SmartToolbox-OffchainProxy bean. 57
8.10 Sequence diagram showing how a invitation is propagated to the Smart-ToolboxOffchainProxy. 58
8.11 Sequence diagram showing how an invitation can be accepted or declined. 59
8.12 Sequence diagram showing how a proposal is send to the multi user chat. 60
8.13 Sequence diagram showing how a proposal is received from the multi user chat. 60

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

8.14 Sequence diagram showing the destruction of a ProposalHandler. 61
8.15 Sequence diagram showing group management related methods. 62
8.16 Sequence diagram showing the CSR process. 63
8.17 Sequence diagram showing the key retrieval from server. 64
8.18 Sequence diagram showing the login process. 65
8.19 Sequence diagram showing the reconnection of ProposalHandlers during the initialziation process. 66
8.20 Sequence diagram showing the initialization process. 67
8.21 Sequence diagram showing how an action can be sent between participants. 67
8.22 Class diagram of key management related classes. 68
8.23 Sequence diagram showing the initialization of a KeyStoreManager instance. 69

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

List of Tables

9.1 This table shows the feature comparison table for different private blockchain systems and ourXMPP based implementation. 77

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Acronyms

AMQP Advanced Message Queuing Protocol. 11, 12, 21, 22, 71–74

BFT Byzantine Fault Tolerant. 15, 16, 23

CA Certification Authority. 13, 75

CSR Certificate signing request. 61, 63, 67, 79, 82

EOF End Of File. 23

GUI Graphical User Interface. 44, 46, 47, 51

HTTP Hypertext Transfer Protocol. 23

IOC Invertion Of Control. 55

JID Jabber ID. 29–31, 33, 34, 40, 41

LAN Local Area Network. 42

mDNS Multicast DNS. 42

MSP Membership Service Provider. 13

RPC Remote Procedure Call. 16, 22, 23, 67, 72–74

SASL Simple Authentication and Security Layer. 31, 32, 41, 53

STS Station To Station. 23, 71

TLS Transport Layer Security. 12, 21, 22, 31, 32, 37, 42, 71, 76, 80

URI Uniform Resource Identifier. 23

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

UTXO Unspent Transaction Output. 8

XEP XMPP Extensions Protocol. 29, 38–42, 51

XMPP eXtensible Messaging and Presence Protocol. 1–3, 19, 20, 29–32, 34–39, 41–43, 49–52, 55, 56, 58, 61, 63–67, 71–77, 79–81, 83

XSF XMPP Standards Foundation. 29, 38

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Bibliography

[AA20] A Averin and O Averina. Review of blockchain frameworks and platforms. In *2020 International Multi-Conference on Industrial Engineering and Modern Technologies (FarEastCon)*, pages 1–6. IEEE, 2020.

[ABB+18] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the Thirteenth EuroSys Conference*, pages 1–15, 2018.

[Alr21] Malak Suliman Alrumaih. Introducing contemporary blockchain platforms. *International Journal of Computer Science & Network Security*, 21(4):9–18, 2021.

[B+14] Vitalik Buterin et al. A next-generation smart contract and decentralized application platform. *white paper*, 3(37), 2014.

[Bal17] Arati Baliga. Understanding blockchain consensus models. *Persistent*, 2017(4):1–14, 2017.

[Bas18] Imran Bashir. *Mastering Blockchain Second Edition*. Packt Publishing Ltd, 2018.

[BCGH16] Richard Gendal Brown, James Carlyle, Ian Grigg, and Mike Hearn. Corda: an introduction. *R3 CEV, August*, 1:15, 2016.

[BHLC09] Tim Bray, Dave Hollander, Andrew Layman, and J Clark. Namespaces in xml. *World Wide Web Consortium*, 2009.

[BKM18] Ethan Buchman, Jae Kwon, and Zarko Milosevic. The latest gossip on bft consensus. *arXiv preprint arXiv:1807.04938*, 2018.

[BVGC21] Rafael Belchior, André Vasconcelos, Sérgio Guerreiro, and Miguel Correia. A survey on blockchain interoperability: Past, present, and future trends. *ACM Computing Surveys (CSUR)*, 54(8):1–41, 2021.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[C+16] Christian Cachin et al. Architecture of the hyperledger blockchain fabric. In *Workshop on distributed cryptocurrencies and consensus ledgers*, volume 310, page 4, 2016.

[Cha20] JP Morgan Chase. Quorum white paper (2016). https://raw.githubusercontent.com/ConsenSys/quorum/master/docs/Quorum%20Whitepaper%20v0.2.pdf, 2020.

[Con20a] ConsenSys. account plugins. https://docs.goquorum.consensys.net/, 2020.

[Con20b] ConsenSys. Constellation readme. https://github.com/ConsenSys/constellation, 2020.

[Con20c] ConsenSys. Quorum readme. https://github.com/ConsenSys/quorum, 2020.

[Con20d] ConsenSys. Tessera private transaction manager. https://docs.tessera.consensys.net/en/stable/, 2020.

[CON21] CONSENSYS. Tessera api reference (latest-77595415). https://consensys.github.io/tessera/, 2021.

[CPV+16] Michael Crosby, Pradan Pattanayak, Sanjeev Verma, Vignesh Kalyanaraman, et al. Blockchain technology: Beyond bitcoin. *Applied Innovation*, 2(6-10):71, 2016.

[DGG20] Coin Sciences Ltd Dr Gideon Greenspan. Multichain private blockchain — white paper. https://www.multichain.com/download/MultiChain-White-Paper.pdf, 2020.

[DXM+19] Yueyue Dai, Du Xu, Sabita Maharjan, Zhuang Chen, Qian He, and Yan Zhang. Blockchain and deep reinforcement learning empowered intelligent 5g beyond. *IEEE Network*, 33(3):10–17, 2019.

[EHM+04] Ryan Eatmon, Joe Hildebrand, Jeremie Miller, Thomas Muldowney, and Peter Saint-Andre. Xep-0004: Data forms. *Jabber Software Foundation*, 2004.

[Fou20] Linux Foundation. grpc documentation. https://grpc.io/docs/, 2020.

[Fou21] Apache Software Foundation. Apache activemq artemis documentation. https://activemq.apache.org/components/artemis/documentation/latest/book.pdf, 2021.

[Gro13] JSON-RPC Working Group. Json-rpc 2.0 specification. https://www.jsonrpc.org/specification, 2013.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[Hea16] Mike Hearn. Corda: A distributed ledger. *Corda Technical White Paper*, 2016, 2016.

[HKG+21] Saqib Hakak, Wazir Zada Khan, Gulshan Amin Gilkar, Basem Assiri, Mamoun Alazab, Sweta Bhattacharya, and G Thippa Reddy. Recent advances in blockchain technology: A survey on applications and challenges. *International Journal of Ad Hoc and Ubiquitous Computing*, 38(1-3):82–100, 2021.

[HMESA08] Joe Hildebrand, Peter Millard, Ryan Eatmon, and Peter Saint-Andre. Xep-0030: service discovery. *XMPP Standards Foundation, Tech. Rep*, 2008.

[Hyp19] Hyperledger. Private data. `https://hyperledger-fabric.readthedocs.io/en/release-1.4/private-data/private-data.html`, 2019.

[Hyp20] Hyperledger. Ledger. `https://hyperledger-fabric.readthedocs.io/en/release-2.0/ledger/ledger.html`, 2020.

[IVBA+14] Antti Iivari, Teemu Väisänen, Mahdi Ben Alaya, Tero Riipinen, and Thierry Monteil. Harnessing xmpp for machine-to-machine communications & pervasive applications. 2014.

[Kwo14] Jae Kwon. Tendermint: Consensus without mining. 2014.

[LBSA+09] Scott Ludwig, Joe Beda, Peter Saint-Andre, Robert McQueen, Sean Egan, and Joe Hildebrand. Xep-0166: Jingle. *XMPP Standards Foundation*, 2009.

[LD20] IBM Developer Luc Desrosiers, Ricardo Olivieri. Oracles: Common architectural patterns for hyperledger fabric. `https://developer.ibm.com/technologies/blockchain/articles/oracles-common-architectural-patterns-for-fabric/`, 2020.

[LSP19] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. In *Concurrency: the Works of Leslie Lamport*, pages 203–226. 2019.

[Ltd20a] Coin Sciences Ltd. Introducing multichain streams. `https://www.multichain.com/blog/2016/09/introducing-multichain-streams/`, 2020.

[Ltd20b] R3 Ltd. Accounts documentation. `https://github.com/corda/accounts/blob/master/docs.md`, 2020.

[Ltd20c] R3 Ltd. Documentation and training for corda developers and operators. `https://docs.corda.net/docs`, 2020.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[Mil05] Matthew Miller. Xep-0050: Ad-hoc commands. *Standards track, XMPP Standards Foundation*, 2005.

[MJS+14] Andrew Miller, Ari Juels, Elaine Shi, Bryan Parno, and Jonathan Katz. Permacoin: Repurposing bitcoin work for data preservation. In *2014 IEEE Symposium on Security and Privacy*, pages 475–490. IEEE, 2014.

[Nak19] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Technical report, Manubot, 2019.

[NRP+21] Samudaya Nanayakkara, MNN Rodrigo, Srinath Perera, Geeganage T Weerasuriya, and Amer A Hijazi. A methodology for selection of a blockchain platform to develop an enterprise system. *Journal of Industrial Information Integration*, 23:100215, 2021.

[RFC11] Extensible RFC6121. messaging and presence protocol (xmpp): instant messaging and presence. *P. Saint-Andre, Cisco*, 2011.

[SA08a] Peter Saint-Andre. Xep-0045: multi-user chat. *XEP-0045 (Standards Track)*, 2008.

[SA08b] Peter Saint-Andre. Xep-0174: Serverless messaging. *Standards track, XMPP Standards Foundation*, 2008.

[SA08c] Peter Saint-Andre. Xep-0174: Serverless messaging. *Standards track, XMPP Standards Foundation*, 2008.

[SA09] Peter Saint-Andre. Xep-0175: Best practices for use of sasl anonymous. *Informational, XMPP Standards Foundation*, 2009.

[SA11a] P Saint-Andre. Rfc 6120: Extensible messaging and presence protocol (xmpp): Core (2011). `http://tools.ietf.org/html/rfc6120`, 2011.

[SA11b] Peter Saint-Andre. Extensible messaging and presence protocol (xmpp): Address format. Technical report, RFC 6122, March, 2011.

[SAC10] Peter Saint-Andre and Dave Cridland. Xep-0001: Xmpp extension protocols. *Viitauu*, 27:2013, 2010.

[SAD04] Peter Saint-Andre and Russell Davis. Private xml storage. 2004.

[SAS11] Peter Saint-Andre and Lance Stout. Xep-0234: Jingle file transfer. `https://xmpp.org/extensions/xep-0234.html`, 2011.

[SIHC21] Abdurrashid Ibrahim Sanka, Muhammad Irfan, Ian Huang, and Ray CC Cheung. A survey of breakthrough in blockchain technology: Adoptions, applications, challenges and future research. *Computer Communications*, 2021.

TU WIEN Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

[sma] Smart-toolbox for community-blockchains. https://projekte.ffg.at/projekt/3344680.

[sma20] Technical report – smart-toolbox. Complang research group, 2020.

[SSB16] Florian Schmaus, Dominik Schürmann, and Vincent Breitmoser. Xep-0373: Openpgp for xmpp. *Hämtad juli*, 17:2016, 2016.

[Sta12] OASIS Standard. Oasis advanced message queuing protocol (amqp) version 1.0. *International Journal of Aerospace Engineering Hindawi www. hindawi. com*, 2018, 2012.

[SZ15] Yonatan Sompolinsky and Aviv Zohar. Secure high-rate transaction processing in bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 507–527. Springer, 2015.

[Szy04] Michael Szydlo. Merkle tree traversal in log space and time. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 541–554. Springer, 2004.

[Tee17] Cale Teete. Quorum consortium network in azure marketplace. https://entethalliance.org/quorum-consortium-network-in-azure-marketplace.pdf, 2017.

[ten19] tendermint.com. Amino spec (and impl for go). https://github.com/tendermint/go-amino, 2019.

[ten20a] tendermint.com. Blockchain. https://github.com/tendermint/spec/blob/953523c3cb99fdb8c8f7a2d21e3a99094279e9de/spec/blockchain/blockchain.md, 2020.

[ten20b] tendermint.com. Tendermint. https://docs.tendermint.com/master/, 2020.

[ten20c] tendermint.com. Tendermint rpc. https://docs.tendermint.com/master/rpc/, 2020.

[ten20d] tendermint.com. Using tendermint. https://docs.tendermint.com/master/tendermint-core/using-tendermint.html, 2020.

[Vuk15] Marko Vukolić. The quest for scalable blockchain fabric: Proof-of-work vs. bft replication. In *International workshop on open problems in network security*, pages 112–125. Springer, 2015.

[Wan21] Gang Wang. Sok: Applying blockchain technology in industrial internet of things. *Cryptology ePrint Archive*, 2021.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[XWS+17] Xiwei Xu, Ingo Weber, Mark Staples, Liming Zhu, Jan Bosch, Len Bass, Cesare Pautasso, and Paul Rimba. A taxonomy of blockchain-based systems for architecture design. In *2017 IEEE International Conference on Software Architecture (ICSA)*, pages 243–252. IEEE, 2017.

[ZXD+20] Zibin Zheng, Shaoan Xie, Hong-Ning Dai, Weili Chen, Xiangping Chen, Jian Weng, and Muhammad Imran. An overview on smart contracts: Challenges, advances and platforms. *Future Generation Computer Systems*, 105:475–491, 2020.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub