Secure Electronic Health Records Storage and Sharing using Blockchain Technology

Author

Muhammad Usman

FALL 2017 - MS-17 (CSE) 00000205343

Supervisor

Dr. Usman Qamar

DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING

COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING

NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY

ISLAMABAD

JANUARY, 2020

Secure Electronic Health Records Storage and Sharing using Blockchain Technology

Author

Muhammad Usman

FALL 2017 - MS-17 (CSE) 00000205343

A thesis submitted in partial fulfillment of the requirements for the degree of

MS Computer Software Engineering

Thesis Supervisor:

Dr. Usman Qamar

Thesis Supervisor's Signature: -_____

DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING

COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING

NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY, ISLAMABAD

January, 2020

DECLARATION

I certify that this research work titled *"Secure Electronic Health Records Storage and Sharing using Blockchain Technology"* is my own work under the supervision of Dr. Usman Qamar. The work has not been presented elsewhere for assessment. The material that has been used from other sources has been properly acknowledged / referred.

_____

Signature of Student

Muhammad Usman

FALL 2017 - MS-17 (CSE) 00000205343

LANGUAGE CORRECTNESS CERTIFICATE

This thesis has been read by an English expert and is free of typing, syntax, semantic, grammatical and spelling mistakes. Thesis is also according to the format given by the university.

_____

Signature of Student

Muhammad Usman

FALL 2017 - MS-17 (CSE) 00000205343

_____

Signature of Supervisor

Dr. Usman Qamar

COPYRIGHT STATEMENT

ACKNOWLEDGEMENTS

I am extremely thankful to **ALLAH** Almighty for his bountiful blessings throughout this work. Indeed, this would not have been possible without his substantial guidance through every step, and for putting me across people who could drive me though this work in a superlative manner. Indeed, none be worthy of praise but the Almighty. In addition, my admirations be upon Prophet **Hazrat Muhammad (PBUH)** and his Holy Household for being source of guidance for people.

I would like to express my special thanks to my supervisor **Dr. Usman Qamar** for his generous help throughout my thesis, and for being available even for the pettiest of issues. My thanks for a meticulous evaluation of the thesis, and guidance on how to improve it in the best way possible.

I am profusely thankful to **Dr. Wasi Haider Butt** and **Dr. Saad Rehman** for the excellent guidance throughout this journey and for being part of my evaluation committee. In addition, I would also like to express my special thanks to **Dr. Ayesha Khalid** for her help in my thesis.

I would like to thank my family & friends, especially to my brother **Muhammad Rizwan** who has been my inspiration and strength and the reason of what I become today and for his great support and continuous care.

Finally, I would like to express my gratitude to all the individuals who have rendered valuable assistance in this period.

Dedicated to my exceptional parents, excellent siblings and my best friends whose tremendous support and cooperation led me to this wonderful accomplishment. I am truly indebted to you all.

ABSTRACT

There is hardly any human activity which has not been positively affected by digital technologies impacting the access and exchange of information. Specifically, within the healthcare sector major improvements could be achieved. Nowadays nearly all medical records are kept in electronic healthcare record (EHR) systems improving the access to clinical data intending to streamline costs. Electronic Health Records (EHRs) are however largely non-portable and kept on the systems where they have been created mainly due to interoperability, security, and liability reasons. This is resulting in a lack of medical quality for the patient and an increase of healthcare costs since the information transfer among different healthcare providers, spread over different locations, is highly dependent on the patient who is not considered as data owner and may not be aware of certain treatments received. Moreover, medical data is strictly personal and the consequences of unintentional disclosure of this sensitive data are severe. Recordkeeping systems embedded in the various healthcare systems must therefore adhere to the highest standards of security and privacy.

The objective of this thesis was to investigate how data security, privacy and scalability challenges within the context of Electronic Health Record systems can be overcome considering the patient as the owner of his patient data by taking advantage of the Blockchain technology. Blockchains allow for immutable recordkeeping, which means that data stored on the blockchain cannot be changed or tampered with. Each block on the blockchain stores the computed hash of the contents of the previous block, which makes each new block dependent on the previous block. Nodes store their own copies of the blockchain and keep them synchronized by using mechanisms for distributed consensus. Distributed consensus mechanisms for blockchains facilitate methods to decide which block is to be added to the blockchain next and essentially decide which version of the blockchain is the correct one.

In this research a blockchain-based EHR framework has been proposed to overcome above mentioned challenges. We have implemented a prototype of Electronic Health Records System using permissioned blockchain platform “Hyperledger”. This system ensures better data security, data privacy, scalability and easy accessibility and availability of medical records. A thorough description has been provided as to which modules are needed to assure a proper data extraction, mapping, monitoring, user access management and intervention with the Blockchain.

In conclusion, it is possible to overcome security and privacy and scalability (in recently proposed blockchain-based EHR systems) issues by using the Blockchain technology for a variety of use-cases, such as secure storage and sharing of medical data. Although the presented implementation is intended for use in EHR systems, it should also be applicable to other types of recordkeeping systems.

Keywords: Blockchain, Electronic health records, Hyperledger, Permissioned Blockchain

Table of Contents

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

CHAPTER 1: INTRODUCTION

1.1. Background

In most countries, the healthcare industry is composed of several healthcare providers, some with private ownership and some with public. To provide the best possible healthcare services for their patients, providers must exchange patient information with each other. By exchanging information with other providers, health personnel can get a complete overview of a patient's health. This can be critical in determining which type of treatment a patient requires, especially in emergency situations. For this reason, as much patient data as possible is being stored electronically, so that it can be shared quickly with providers over a network. These health records are termed electronic health records (EHRs) [1].

Most providers operate and govern their own local databases of patient data. These databases are running in each provider's own private network, which means that EHR databases are spread around in multiple physical locations. The different databases might contain EHRs formed from multiple data formats, which means that providers might have to support multiple proprietary formats to be able to interpret the EHRs from every provider's database. Several common data standards and regulations for EHR systems have been proposed to solve this issue [2].

As soon as data is made available over a network it is susceptible to eavesdropping and remote attacks. EHRs are regular targets of both malicious attacks and misuse. The contents of an EHR is strictly personal and have great potential in being used to blackmail individuals and institutions. Providers that are not properly managing and securing their recordkeeping systems are penalized with large fines and other repercussions issued by governments overseeing the healthcare industry [3]. Being found responsible for mismanagement of health data could mean that the entire institution must be shut down.

IT systems governing such healthcare records must therefore be able to maintain privacy and integrity of the data in all possible scenarios. However, IT systems in the healthcare industry are highly complex with a great amount of legacy code, which means that even a slight change of functionality might require extensive code rewrites. As a result, there is no single easy solution to make these systems secure. If an event does breach confidentially or integrity of the data it is also important that proper auditing facilities are in place, so that it is possible to estimate the extent of the breach, close any exposed vulnerabilities and avoid it from

happening again. To overcome security and privacy challenges such services needs to adopt a new emerging technology that supports a distributed system and protects the integrity and security of medical data.

The concept of blockchains first gained traction in the form of public cryptocurrencies like Bitcoin [4]. In the last few years, however, blockchain has also sparked interest from other domains. This includes the healthcare domain. Blockchain technology can introduce immutable recordkeeping to an institution. Once data is placed on the blockchain it cannot be changed or replaced without anyone noticing it. This means that as soon as a record is placed on the blockchain neither health personnel nor malicious users are able to tamper with a record. So, there is a need of blockchain based efficient, secure and smart healthcare system that can ensure secure data storage, easy data access and also ensure privacy and give patient full control of their medical data where they can decide and control who can view and manage their medical data. Giving patients ownership of their medical data will build their trust toward the technology and encourages them to continuously engage in clinical decision process and in preventing their medical records from adversaries.

1.2. Problem Definition

A centralized infrastructure leads to several downsides when it comes to sharing medical data among different regions and systems. Even if the data structure and semantics could be agreed upon in order to overcome the described interoperability issues, further challenges arise in terms of security, data ownership, data consistency, and data privacy.

Securing data for a centralized infrastructure is a challenging task as it becomes susceptible to eavesdropping and remote attacks. This implies that an extensive effort needs to be done in order to assure that patient information is secured in terms of privacy, assuring that only authorized parties are able to access the data [5]. From the ownership perspective and in the legal sense, healthcare providers perceive patient data as their property [6]. This creates unnecessary and costly obstacles for patients who need to move their medical records to another location. Current EHR systems are not designed to manage multi-institutional life time records. Therefore, patients leave data scattered across various organizations as life events take them away from one healthcare provider to another. As a result, patients and care givers lose easy access to past records while healthcare organizations run into the challenge of record maintenance: constantly modifying and updating the healthcare data in interaction with the

patient, trying to catch-up to the illusive valid healthcare profile of the patient. This may lead to a bigger problem when it comes to liability questions not knowing how accurate the patient data actually is [7]. Another issue appears in respect to the scalability for centrally hosted EHR systems. Given the fact that patient data is continuously added, changed or removed to the EHR, it is difficult to predict what kind of infrastructure is able to cope with a continuously growing amount of data without impacting the actual performance and therefore usability. Hence, centrally hosted systems may have the possibility to upscale processing power on a short term but may face their limits on the long-term due to its predefined data architecture [8].

A technology which might be able to overcome those problems could be the Blockchain technology. This technology was first mentioned in 2008 within the white paper by Satoshi Nakamoto, in which he described the concept of a distributed cryptocurrency, better known as "Bitcoin". This technology allows a purely peer-to-peer online cash transfer among participants without the burden of going through a middleman (i.e. financial institution) handling transactions and being in charge to prevent double-spending. The basic principle of the Blockchain technology is based on timestamped transactions (blocks) hashed into an ongoing chain of a "hash-based-proof-of-work", forming a record that cannot be changed without redoing the "proof-of-work", better known as Blockchain. The Blockchain serves as proof for the sequence of events witnessed and attest that the created chain came from the largest pool of CPU power. One key characteristic of this architecture is that messages are broadcasted on a best effort basis among the participants, where nodes can leave and re-join the network at will, accepting the longest proof-of-work chain as an attest of what happened during their absence [4]. This architecture is by design inherently resistant to unauthorized modification of the data while autonomously managed. As a result, it is considered as a distributed ledger, recording transactions between two parties efficiently and in a verifiable and permanent way. This concept sparked a lot of interest by the media and across industries in order to improve current security and scalability problems. Further areas of application were revealed such as the exchange of electronic healthcare records for being able to overcome the initially described challenges. In the 2016 and 2017 editions of Gartner's Hypecycle for emerging technologies [9], [10], the Blockchain technology has been placed on top of the "Peak of Inflated Expectations". At this stage, early publicity produced a number of success stories often accompanied by scores of failures. Within Gartners Hypecycle 2017 specifically for Blockchain technologies, Blockchains in Healthcare are placed in the first phase of the

Hypecycle, called “Innovation Trigger” where a significant interest by the media is triggered without having a proven concept or product [10].

The focus in this thesis will be on how blockchain technology can help to implement a blockchain-based electronic health records management system that ensures better scalability, data security, data privacy and give patients ownership of their medical records.

Given the fact that the Blockchain technology is a rather new concept, very little research has been conducted. This highlights why scientific research is needed to conclude sufficiently on the realistic potential of the Blockchain technology within the context of EHRs and the described scalability, security and privacy challenges to which this master thesis shall contribute.

1.3. Research Objectives and Research Questions

The objective of this thesis was to investigate how the mentioned data security, data privacy and scalability challenges within the context of Electronic Health Record systems can be overcome considering the patient as the owner of their patient data by taking advantage of the Blockchain technology. Therefore, the following main research question have been proposed:

- **MRQ:** “How can the Blockchain technology overcome current EHR data security and data privacy challenges and scalability issues in blockchain-based EHRs which are recently proposed?”

The main research question is being answered by responding to the following sub-research-questions:

SRQ1: What are current Blockchain technologies available, suitable for a Blockchain-based EHR?

SRQ2: How does a Blockchain-based EHR architecture look like taking all functional and technical stakeholder requirements into account?

SRQ3: What is the behavior of a developed architecture taking important features into account?

SRQ4: What are trade-offs of the developed and tested architecture for a realistic implementation?

1.4. Research Gaps

Data Security: Security is very crucial for patient safety and to preserve patient’s medical history from adversaries. Failing to secure the patient record has financial and legal

consequences, as well as the potential to impact patient care. Sharing of private medical information online between the patient and their respective provider contains sensitive information that can easily be compromised if a proper security measure is not put in place.

Data Privacy: Data privacy involves ensuring only authorized parties may access the medical record. This impacts any healthcare system, as patient privacy is not only an ethical responsibility, but a legal mandate.

Scalability: Some researchers have recently proposed blockchain-based EHR systems. Scalability is one of the issues identified in these blockchain-based systems. These proposed blockchain-based EHR system are not scalable enough to handle large number of transactions.

Data Ownership: Giving patient full control of their healthcare information so they can decide who can view and add their medical records. Giving patient ownership of their data and providing a strong consent mechanism for sharing data with healthcare providers.

1.5. Intended Results

We have proposed blockchain-based framework for electronic health records management which will provide better scalability, security and privacy and will give patients ownership of their medical records. This framework proposes smart contracts for administration, health-providers and patients.

The work on this thesis will result in a description of how such a system can be implemented with use of the Hyperledger Fabric blockchain. Together with Ethereum, the Hyperledger projects are some of best known and recognized open source blockchain projects in development today. Hyperledger Fabric is favored over Ethereum in this thesis because of its design as a permissioned private network, as opposed to Ethereum's public blockchain [11], and because of its unique three-step transaction flow and smart contract implementation [12]. Most components making up a Fabric blockchain are also modular and replaceable, which is an important feature of a system that is intended to be further extended at a later time.

Along with a description of the Fabric implementation of the EHR framework, an actual prototype of the system will be developed. The prototype is intended to showcase how the system will behave in real production systems and allow for testing and measurements to be performed on the system. The prototype will comprise two software packages:

- HLF Network package (Hyperledger Fabric blockchain network)

- Node.js Application package (JavaScript Business logic and front-end)

The HLF Network package will comprise the configuration files and source code required to initialize and deploy the Hyperledger Fabric network. This will be a fully scalable and functional network. One must, however, expect some components to be replaced with components that are compatible with existing protocols and components in specific healthcare network implementations. Such components would typically be e.g. network certification authorities for issuing digital identities.

The Node.js application package will contain business logic for interacting with the Fabric blockchain. The application will utilize the Fabric software development kit (SDK) for Node.js [13]. The Fabric SDKs help applications manage chaincodes and events on the blockchain, as well as making the application able to act on behalf of a specific user context. A graphical user interface (GUI) for invoking the application methods will also be created, which makes for easier demonstration of the system.

1.6. Societal and Research Contribution

As mentioned, the Blockchain technology received much attention throughout several sectors. Don Tapscott, known as one of the leading authorities on innovation, media, and the economic and social impact of technology, claims that “The Blockchain technology is likely to have the greatest societal impact for the next few decades above emerging technologies such as social media, artificial intelligence, and the internet of things.” [14]. This might be true, given the decentralized and secure architecture of the Blockchain technology, having a positive impact on sharing assets. The idea of sharing assets throughout the globe without having adverse effects such as the described data security, data privacy and scalability challenges. Exchanging electronic healthcare records by using the Blockchain technology would therefore benefit not only patients by increasing the healthcare quality and decreasing costs but also provide an insight into how this technology can be applied in the context of protecting valuables such as money, identities, intellectual property, art, and scientific discoveries. The result of this thesis takes part by answering the question how likely the Blockchain technology can disrupt current deficiencies within our society from which every human would benefit for the following examples:

- Protecting rights through immutable records
- Protecting privacy of sensitive information

- Enabling citizens to own and protect their data
- Ensuring easy accessibility of digital records

From the research point of view, this research contributes by providing an insight into what the most appropriate blockchain architecture is and how it interacts with the identified stakeholder requirements. Previous research revealed limitations on the Bitcoin and Ethereum Blockchain, since both platforms are - at the time of this study - only able to process 3 to 20 transactions per second and are computationally expensive technologies. As a reference, financial service provider such as VISA are on average capable of handling 2000 transactions per second [15]. For a realistic use of the proposed solution, scalability is critical and therefore also beneficial for other use-cases. The developed and tested architecture contributes to answer those questions and enables future research within this new field of technology.

1.7. Thesis Outline

The chapters of this thesis, in chronological order, are:

Chapter 1: Introduction - Information about the research background, problem definition, research objectives and research question, intended outcomes of this study, societal and scientific contribution of research and organization of thesis.

Chapter 2: Literature Review - based on the read papers, the main problems of the currently used databases and healthcare systems are described, and confronted with the valences of the blockchain technology. After that, some problems and challenges of this technology in healthcare are expressed, and also, some implementations and solutions, already developed by other authors.

Chapter 3: Theoretical Background - An introduction to the theory that the thesis is based on.

Chapter 4: Proposed EHR Framework – Highlights the proposed EHR framework

Chapter 5: Implementing the Framework - A presentation of the methods used to design and implement the EHR framework.

Chapter 6: System Description - A presentation of the complete system and how it can be deployed.

Chapter 7: Evaluation and Performance - Discussion about the research outcome, evaluation of the developed system and results about the performance, security and privacy and scalability.

Chapter 8: Conclusion - A conclusion derived from the obtained results.

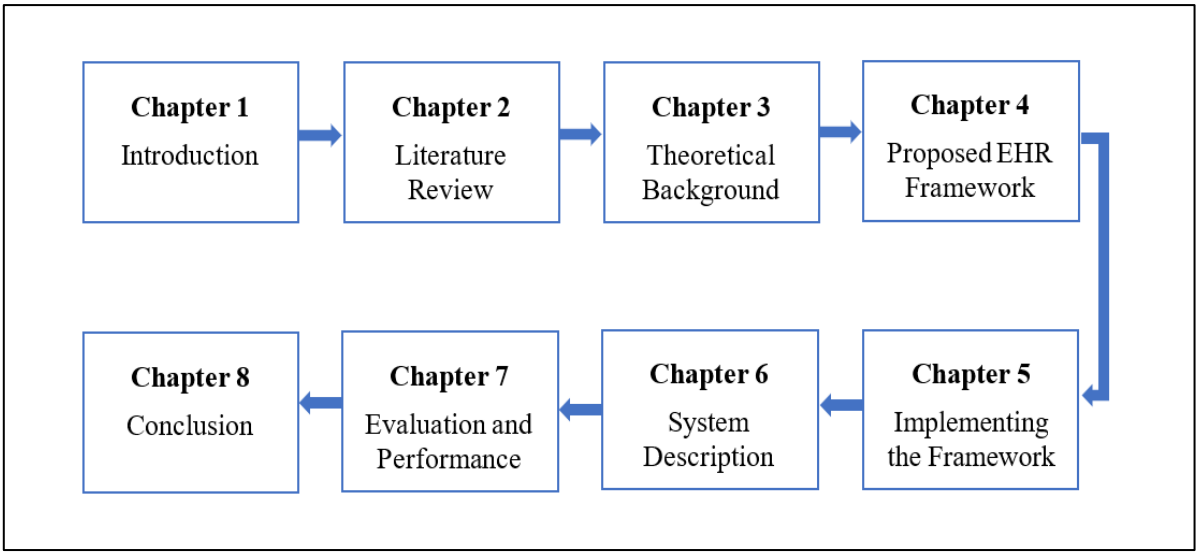

Figure 1.1: Thesis Outline

Chapter 2

Literature Review

CHAPTER 2: LITERATURE REVIEW

As explained in the introduction, extensive research was required to answer the initially posed research question in a scientifically sound manner. The first sections of this chapter are based on revision articles about blockchain technology applications as a database and as an alternative to healthcare systems. In the last section some blockchain implementations in healthcare are analyzed.

2.1. Traditional Databases vs Blockchain

Currently, traditional distributed database management systems (DDBMS) are used to store medical data. Those databases are centralized, although, the data is distributed by many computers. DDBMS synchronize all the data periodically, as if it were all stored in the same computer, so they are logically centralized [16]. Within those systems there are two major types: Structured Query Language (SQL)-based systems, like Oracle, and NoSQL-based systems, like Apache Cassandra, CouchDB and PouchDB [17]. Given the objective of this thesis it's important to realize what are the advantages of using the blockchain technology as a distributed ledger, instead of the traditional databases. Those advantages can be grouped in four central point:

a) Decentralized Management
b) Immutable data records
c) Robustness and Availability
d) Security and Privacy.

As it was said before, blockchain is a P2P, decentralized database management system (a): every node has a copy of the data and runs independently while following the protocols. Decentralized management makes blockchain technology suitable to integrate applications in which healthcare stakeholders wish to collaborate without a central management, which could cause some friction and increase the costs. On the contrary, DDBMS are centralized, which makes them susceptible to the problem of single point of failure [17].

Changing the data, already inserted into a blockchain, is nearly impossible, due to the cryptographic links between its blocks (b). Thus, it's said that blockchain only supports read and create options. So, blockchain has the capacity to hold critical information, without the risk

of it being deleted or changed. In contrast with traditional databases, which support create, read, update, and delete functions, managed by a central authority [17].

Both DDBMS and blockchain database management systems are distributed, however, it takes much cost to the DDBMS to achieve the high level of data redundancy blockchain does. Therefore, blockchain is a good way to store records that necessarily need to always be available and preserved (c), such as electronic health records of patients [17].

Finally, blockchain uses cryptographic algorithms that improve the security and privacy of the data (d): encryption algorithms, hashing algorithms. Additionally, blockchain users are identified by a generated hash, rather than, for example, an IP address. Other algorithms can be used to ensure the ownership of the digital assets, by generating and verifying the authenticity of the public and private keys as digital signatures [17].

2.2. Healthcare Traditional Systems vs Blockchain

In the recent years, the awareness of the blockchain key advantages mentioned above has been growing. This fact results in an increased number of studies, related with blockchain distributed ledger technology applied to health and biomedical care. In many of those studies were developed new applications that can be categorized based on their objectives as: enhance insurance claim process; improved medical record management; accelerated clinical/biomedical research and advance biomedical/health care data ledger [17]. Those applications try to solve the existing problems in the traditional healthcare and health information exchange systems.

Traditional health systems are centralized: there is an intermediary who controls and stores all the data [18]. Blockchain is forged by consensus, so every node helps to decide what information is traded and to whom and, of course, every node has a copy of the data stored in the blockchain. In addition, decentralized systems reduce the costs of transactions.

Healthcare records are considered critical information and imply high privacy and confidentiality. However, sometimes, it’s important to share them with someone. With the current healthcare systems this can be difficult, given their lack of interoperability. Most of the systems have not compatible data types, which forbid the exchange of data [18]. Blockchain systems are "based on open source software, commodity hardware, and open API’s (application programming interfaces)" [19]. An open source blockchain for healthcare enhances interoperability between systems and, it’s more efficient handling large volume of data and

more users. The blockchain distributed ledger enables near real time updates in all nodes [18], simplifying the data exchange in the same network.

Blockchain "smart contracts" have a major role in assuring the privacy and confidentiality of the blockchain network. In the current health systems, there are inconsistent rules and permissions that constrain the process of a certain health stakeholder accessing the data of some patient [18]. With "smart contracts" it is possible to create a set of rules to control the access to the patient's data [18]. By doing that, the patient, owner of the medical data, can choose with whom he wants to share is medical information. Besides that, "smart contracts" are one of the major features of blockchain technologies, that critically contribute to the enhancement of decentralize management, because it allows rules and permissions to be written in code, therefore fulfilled automatically without a single central entity to control it.

2.3. Blockchain Challenges

As any other technology still in expansion, blockchain, and especially, permissionless blockchain in healthcare, has some challenges to overcome. The first one is related to transparency and confidentiality [17]. Transparency can be one of the greatest benefits of blockchain, because it enhances trust, yet can be dangerous in healthcare, if not safeguarded with a good implementation of "smart contracts" to manage permissions. In a blockchain network everyone can see the metadata inherent to a transaction of information, and in some cases, this is critical private data. Another issue is related with the "pseudonymity" of the users, in other words, despite the blockchain users be identified only by hash's, they can be reidentified through analysis and investigation of the public metadata contained on the transaction with certain user's hash, thus blockchain can provide only pseudonymity and not total anonymity [17].

The second challenge and biggest one in permissionless blockchains is related to speed and scalability [17]. As explained in the previous section, to ensure the creation of a new block, i.e. validate all the transactions inside it and write them into a block, it's necessary to solve the consensus algorithm, which can take some time. This speed constrain might affect the capacity of the blockchain of being scalable. As an example, Visa has a theoretical maximum speed of transactions of 2000 per second, on the other hand, Bitcoin only has 7 per second [20].

The last challenge is the threat of a 51% attack [17]. This challenge is common to all types of blockchain applications, especially public blockchains. It refers to the non-zero

probability of existing fewer honest nodes than malicious ones. In these cases, the malicious nodes have more computational power than the honest ones, thus, when two blocks, one malicious and the other honest, are competing to the next spot in the chain, the malicious will win. Hence, the malicious nodes take control of the network, by consensus [17].

2.4. Blockchain Solutions and Implementations

After realizing some challenges and issues of a potential healthcare blockchain system, it's necessary to walk through some proposed solutions and implementations, that intend to overcome the previously described points.

There are two major approaches to improve the scalability of the system (with the same objective of reducing the quantity of data stored in the blockchain): use the blockchain only as an index of health data [19] or use it only to store ongoing verified transactions [21]. Rather than storing every information, every medical record, in the blockchain, which would have data storage implications and data throughput limitations, the authors, recommend using the blockchain as an index of healthcare data, a list of all the user's health records and data [19]. To do that, each transaction inside a block would contain the user's hash; an encrypted link to the health record; a timestamp and, to improve the data access efficiency, the type of data of the medical record and other metadata that would facilitate possible queries. The encrypted link work as a pointer to a data repository, where is all the medical data, called Data Lake [19].

Data lakes are scalable data repositories holding data in its natural format. They use a flat architecture, and each data element is assigned with a unique identifier, tagged with metadata [22], encrypted and digitally signed [19]. These repositories support interactive queries, text mining and analytics, and machine learning [19]. However, they are centralized which make them susceptible to the single point of failure problem. In [19] was proposed a general mechanism for recording medical/health data into a blockchain and data lake, that can be visualized in Figure 2.1. With this mechanism it's possible to take advantage of two different ways of acquiring health data: by wearable sensors and mobile applications, and by medical records made by health professionals. To store the data securely, all the data must have a digital signature to verify its information. After that the data would be encrypted and sent to the data lake for storage. When the data is stored, a pointer in the blockchain is registered along with the user's hash and, then, is sent a notification to the user/patient with the information that new medical data was added to his blockchain [19]. Keeping in mind this mechanism, the authors,

proposed that the users could use their personal mobile devices to give permission to a caregiver to access their health data. With that permission the caregiver could decrypt and authenticate the digital signature, accessing to the data lake through the blockchain. Additionally, the user would be able to see the list of the users that access his blockchain [19].

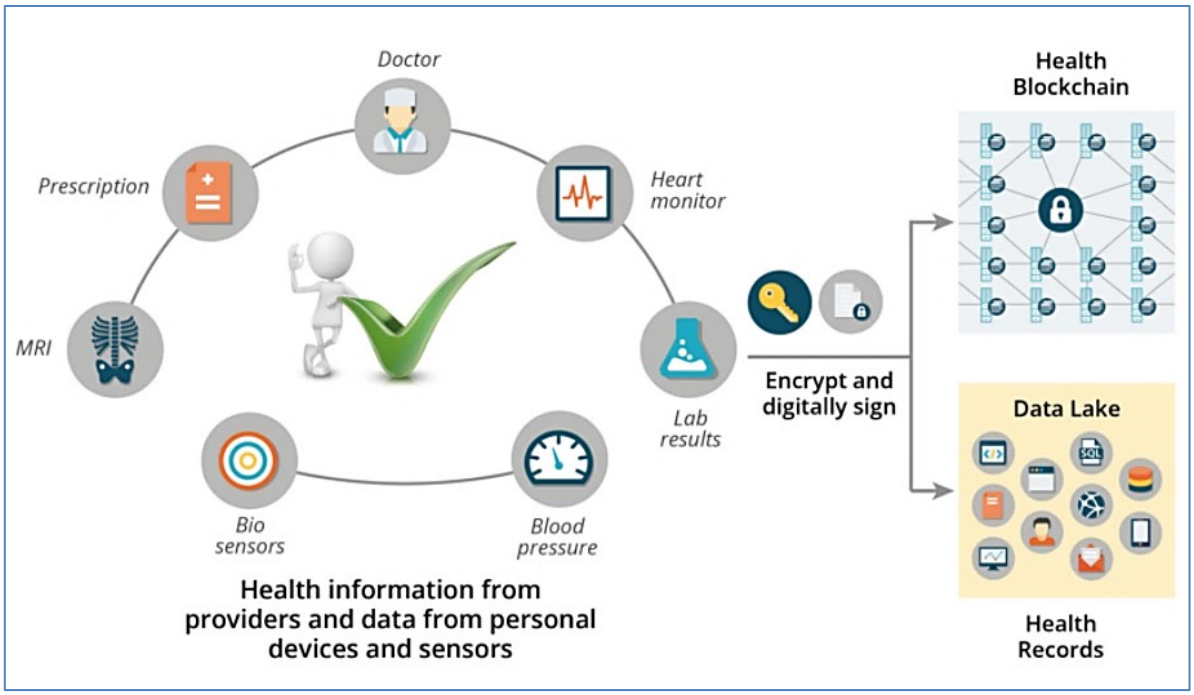

Figure 2.1: Proposed mechanism of storing health data inside the blockchain and data lake [19]

MedRec [23], is prototype system to manage the electronic health records, using blockchain technology, developed by the MIT media lab. The objective of this project was to create a patient-centered management system, as a response to three critical problems: fragmented, slow access to medical data; patient autonomy; improved data quality and quantity for medical research. A block, in MedRec, represents data ownership and viewership permissions shared by all the members of a blockchain network. The MedRec blockchain utilizes "smart contracts", using an Ethereum blockchain, to automate and track changes in permissions or creation of new records. This is done by establishing a patient-caregiver relationship, which associates a medical record from the patient with viewing permissions and pointers to the raw data for the caregiver, to use in external databases. By logging these relationships, the caregivers can add a new record from a specific patient. Patients can share their data with caregivers, generating a notification and a verification, in both cases, to accept or reject the changes made [23]. Like [19], all the raw health data is in an external database. However, each MedRec user has his database with the data that he has the permission to view.

A syncing algorithm manage the exchanges of data off-chain, between the caregivers' database and the patient's database. This system uses PoW (Proof of work) algorithm for consensus mechanism which requires lot of resources for mining that makes it a very expensive solution.

Sofia et al. [24] have presented a conceptual electronic health records access and sharing mechanism that uses blockchain and smart contracts. They have proposed decentralized healthcare platform based on blockchain technology, where the patients' medical records could be better preserved, and which provides efficient access mechanism.

Hongyu Li Et al. [25] have proposed a data preservation system for electronic medical records. This is a blockchain-based system and aimed at preserving the patients' privacy and providing a reliable solution for storage of medical records and ensuring the verifiability and primitiveness of stored data. They have implemented a prototype of data preservation system using Ethereum platform which is a permissionless blockchain platform and have some privacy related issue.

Tengfei Xue et al. [26] proposed a medical data sharing model that is based on blockchain technology and they have discussed the principles and different components of the system, but this design have some implementation problems.

This solution also presents an improved consensus mechanism which solves the problem related to saving, checking and synchronizing health data between different healthcare participants. But this solution has some disadvantages related to data storage.

Xia et al. [69] designed a data sharing framework which is also based on blockchain technology. It utilizes cloud for sensitive data storage and takes advantage of immutability and built-in autonomy features of blockchain technology for address challenges of access control. In a later research they have also proposed a system which is based on blockchain and named as "MeDShare", and this proposed system has minimal risks of data privacy. This system solves the challenges of medical data sharing among different participants of the healthcare in a more trusted environment. Both these schemes still need the assistance of the cloud which is a weak point of these proposed schemes.

Vujičić et al. [70], in their research have introduces blockchain technology, Ethereum and bitcoin. They explained that bitcoin which is a peer-to peer cash system where bitcoin transactions are performed on that peer-to-peer network. They have also discussed the concept of mining in blockchain and proof-of-work algorithm which is used for consensus mechanism.

The authors main focus is on the scalability and that is a severe problem in the blockchain. They also discussed Bitcoin Cash, SegWit, Lightning which were to address the issue of scalability.

The next section of the paper explained the Ethereum and its dependencies. Ethereum was defined as the system that could be used for representation of blockchain that is built upon the Turing complete language of programming. They also explain that Ethereum can be used for empowering its users to define their own access and ownership rules in the applications they create. The paper explains the Ethereum in great detail as it contains the explanation of Ethereum accounts, transactions, messages, and tokens. In general, the complete Ethereum understanding was provided to the readers.

The authors differentiate Ethereum blockchain from bitcoin's blockchain. As, Ethereum contain the transaction details along with the block number, difficulty level and nonce in its header. While bitcoin blockchain do not store the transaction details such as list of transaction and history in its header. They further explained that Ethereum uses Keccak 256-bit hashing algorithm to store the header details of the previous block. Along with the early stages of digital crypto-currency authors defined how blockchain technology has evolved into a technology that serves its purpose in many fields of life. They also identified that scalability is an issue faced by many implementations of blockchain it has certain solutions but it still needs more attention and frameworks to eliminate it further.

Kuo et al. [71] provides a comprehensive overview of biomedical and healthcare applications that could be developed using blockchain technology. They view decentralized management of records, immutability of audit trails, data provenance, availability of data, security, and privacy as benefits to be gained from the implementation of blockchain technology over traditional distributed database management systems. They describe 4 potential use cases:

- Improved Health Record Management
- Enhanced Insurance Claim Process
- Accelerated Clinical/Biomedical Research
- Advanced biomedical/health care data ledger

The authors also discuss potential challenges that blockchain applications may face in a healthcare environment relating to transparency, confidentiality, speed, scalability, and resistance to malicious actors.

Z. Shae et al. [72] have proposed to combine assistive technologies such as electronic patient records, assistance monitoring of internal patients etc, with blockchain technology, looking for the integration of medical data from different health entities, thus treating traceability of information to give a more accurate diagnosis. Participation is also proposed of parallel nodes for the optimization of blockchain networks. With technological models, data protection and encryption measures are proposed by means own for the preservation of information. Also, as an analysis of the results of research that shows weaknesses in security and anonymity issues giving thus attention on issues such as safe and true data. This article covers the scope limitations, impact and challenges of blockchain, proposes measures to be taken for a blockchain network for the medical field safeguarding the privacy and anonymity information of related entities. This solution is proposed on Ethereum blockchain which use computationally expensive algorithm. Moreover, they have not implemented any prototype, so verifiability of this solution is still not known.

Dufel [73] discusses blockchain, alongside other peer-to-peer technologies, and their application towards healthcare. BitTorrent, a peer-to-peer file sharing system, is discussed, as well as distributed hash tables. The authors suggest that only a combination of blockchain, distributed hash tables, and BitTorrent would able to effectively create a peer-to-peer health information exchange system. This review could be helpful to advance the blockchain technology in various domains of biomedical or healthcare by using the identified applications discussed in the review.

Rabah [74] is a review paper that outlines the various challenges and opportunities that blockchain applications face in the healthcare industry. The authors believe that blockchain technology will advance efforts to improve patient care, treatment efficacy, security, and reducing costs. They suggest that electronic medical record management will be made more efficient, disintermediated, and secured through blockchain technology.

SimplyVital Health [75] is a startup company founded in 2016 that aims to leverage blockchain technology in the context of healthcare. Their publicly available web materials indicate that they are working on two products: “ConnectingCare” and “Health Nexus”. ConnectingCare is a care coordination tool used by providers that leverages blockchain

technology to create a secure audit trail. Health Nexus plans to be its own healthcare focused blockchain with an underlying cryptocurrency token called “HLTH”.

2.5. Proposed Framework Comparison with Related Work

In this section we have compare our proposed framework with the above mention (related work) EHR solutions. Our proposed system is based on permissioned blockchain for efficient storage and sharing of electronic health records (EHRs) which provides better security and privacy of data and ensure better scalability. We have compared our proposed framework features with the related work [23] [76] [72].

MedRec a decentralized records management system, first presented in 2016, which use smart contracts to provide patients and providers with addresses linking to existing health records, essentially providing patients with logs and easy access to their health records across providers. Providers are incentivized to participate in the network by receiving aggregated and anonymized data as rewards for validating blocks in the network. Our prototype significantly differs from the framework in [23]. MedRec is based on permissionless blockchain implementation and use Proof-of-Work (PoW), thus involves transaction fees, and requires involvement into “mining” and account management processes. Secondly, Proof-of-Work (PoW) consensus algorithm requires a lot of recourses which make is it a very costly solution. Moreover, it uses cryptocurrency which may also limit its use. In contrast, we have chosen permissioned blockchain implementation based on the requirements from the medical perspective which uses a computationally lighter consensus algorithm Practical Byzantine Fault Tolerance Algorithm (PBFT) and do not involve any cryptocurrency.

Kim et al. [76], also proposed a framework for of healthcare data exchange which is based on blockchain technology. This framework includes storage of two kinds of healthcare information which includes Electronic Medical Records (EMR) and Personal Healthcare Records (PHR). Both these are loosely coupled to handle the different kind of information. The objective of this research was to improve the throughput and fairness of the framework proposed.

Z. Shae et al. [72] have proposed to combine assistive technologies such as electronic patient records, assistance monitoring of internal patients etc, with blockchain technology, looking for the integration of medical data from different health entities, thus treating traceability of information to give a more accurate diagnosis. Participation is also proposed of

parallel nodes for the optimization of blockchain networks. This solution is proposed on Ethereum blockchain which use computationally expensive algorithm. They have not implemented any prototype, so verifiability of this solution is still not known. Moreover, they have not defined any universal data standard for exchange purposes in blockchain.

Our proposed framework ensures improved security of Electronic Health Records and ensure full privacy of data being stored on the Blockchain. The system gives patients full control of their where they can decide who can view and add their health records. This access control ensures privacy of patient sensitive information. The consensus mechanism of the system runs on trusted and pre-defined peers, so chances of data malicious activities on data are also minimal. Moreover, as we are using permissioned blockchain only authorized user will be able to get access to the system and their identities will be known.

Chapter 3

Theoretical Background

CHAPTER 3: THEORETICAL BACKGROUND

3.1. E-Health

The term e-health is a broad definition used for denoting the digital processing of health information e.g. digital prescriptions, appointment scheduling and patient data records [27]. E-health is a vital resource to any healthcare system. Being able to exchange health information digitally is key to effective medical help and is especially important in emergency situations. As with all digital processing systems, the challenges in e-health are generally related to maintaining availability, confidentiality and integrity of the data.

The demand for confidentiality and integrity of data within the field of e-health are among the highest in any industry. Much of the health information transmitted is strictly personal and governed by national privacy laws. Regulatory institutions ensure that healthcare data is not neglected and that it is stored according to the requirements specified by the law.

3.1.1. Electronic Health Records

An electronic health record (EHR) is a collection of a patient's electronically stored health data [28], e.g. test results and medications. The benefit of an EHR over a traditional physical record is that an EHR can be shared electronically and thereby be available to health personnel at other locations much quicker, as illustrated in Figure 3.1. Sharing EHRs between different health institutions over a network means that a patient's health information is available for use immediately when it is needed, no matter which institution maintains the original record.

Several EHR specifications, standards and regulations exist [29]. An EHR specification comprises both data models and communication standards. Communication standards are intended to support interoperability between different systems and to maintain confidentiality of the data. Institutions sharing EHRs must either agree on a common EHR standard or implement measures to interpret data in multiple formats.

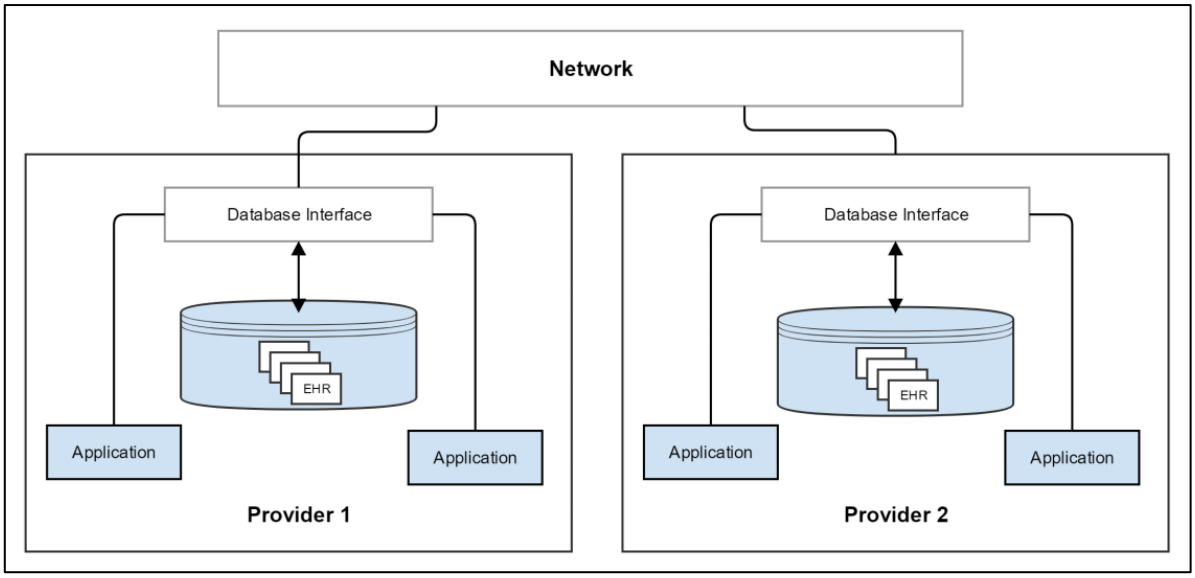

Figure 3.1: A traditional EHR system comprising two providers

3.1.2. Security and Privacy Concerns

The challenges in respect to sharing EHRs are many. The great benefits of electronic access to records come with an increased security risk. Essentially, EHRs facilitate for rapid sharing of records to possibly a large number of people. As a result of potential system errors or malicious entities present in the network, privacy breaches and unauthorized access to EHRs are known to occur and are hard to completely safeguard against.

It is important to realize that privacy can be violated without involvement of any malicious non-authorized entities. Even health personnel that are authorized to access EHRs, are not supposed to access an EHR of a patient if it is not a strict requirement for them to perform their job. In [29], Fernández-Alemán et al. concludes that a harmonization of security and privacy standards found in EHR systems are required, and that auditing is particularly useful to identify suspicious access and common access practice.

Typically, patients are not intended to directly access EHRs, as EHRs are merely used by health personnel to decide how to treat a patient. In some systems, however, EHRs can be partly presented to patients [30] through various web-based services. Depending on the country where the system is deployed, patients might have the legal right to access their complete health information. However, to get hold of a complete EHR one must make an official request.

The benefits and concerns in allowing patients to access their EHRs, e.g. immediate access to test results without waiting for a practitioner to assess the results, are discussed by Beard et al. in [30]. One of the concerns raised with respect to this is that patients are not

qualified to assess test results and might as a result of this misinterpret critical test results, which is why the approach of only partially presenting EHRs to patients is used in most of such systems.

3.2. Blockchain

Blockchain is the term used for a distributed ledger constructed as a chain of blocks, as illustrated in Figure 3.2. The concept was first introduced by Satoshi Nakamoto in 2008 [4] and is best known for its implementation in the Bitcoin cryptocurrency network. The blockchain technology has received large interest in the recent years and many of the world's leading IT companies, such as IBM and Oracle, have devoted substantial amounts of resources to work on the technology.

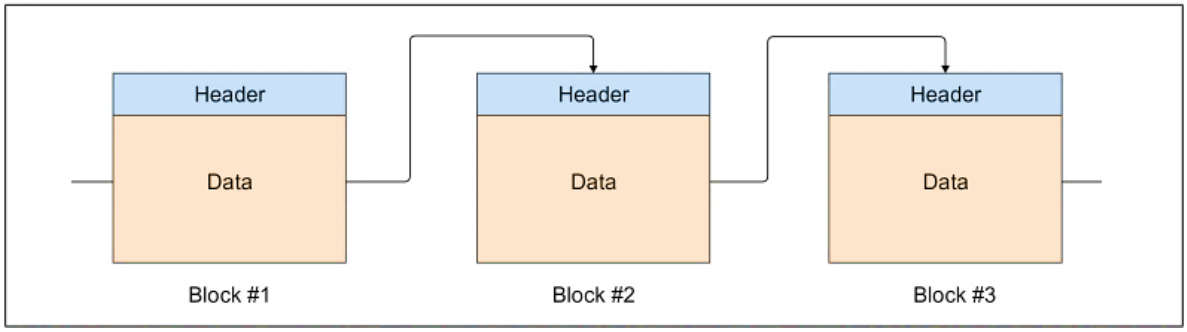

Figure 3.2: Three blocks creating a blockchain by storing the hash of the previous block in the header of each succeeding block.

A distributed ledger is characterized by the fact that information is not stored in a single location governed by a single entity, but instead replicated by every node in the network. Each node holds its own copy of the ledger, meaning that malicious changes made to the ledger on one of the nodes will not be replicated to other nodes, as long as the malicious nodes do not outnumber the functional nodes [4]. The number of malicious or faulty nodes required to break the network depends on the specific method for consensus used in the network.

Information stored on the ledger is placed in blocks. A block consists of a header section and a data section. The data section will typically be composed of several data entries, also known as transactions. A batch of multiple transactions can be placed in a block, instead of creating new blocks for each transaction.

3.2.1. Cryptography in Blockchain

Cryptographic methods are used for a wide range of operations in blockchain networks, e.g. securing blocks and signing transactions. These operations are supported by the use of hash functions and public-key cryptography [31]. A hash function can be thought of as a one-way mapping from a specific input to a specific output. The chances of collisions are extremely small and computationally infeasible to find. We can therefore consider each input to produce a unique output.

Transactions are signed using public-key cryptography, also known as asymmetric cryptography. A basic protocol for public-key encryption is illustrated in Figure 3.3. Authorized entities in the network are supplied with private keys, while the public key is publicly available. An entity would sign its proposed transaction using its private key. Other entities in the network can then decrypt the transaction by using the public key and verify that the transaction was indeed created by an entity with access to a valid private key [32].

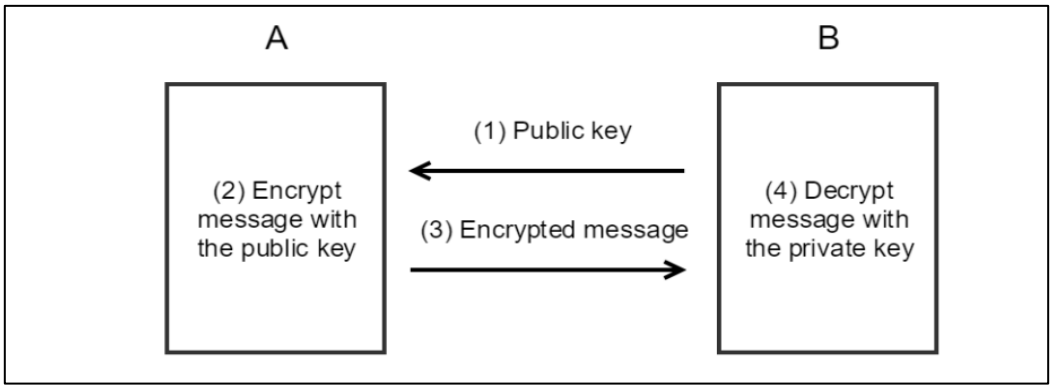

Figure 3.3: A basic protocol for public-key encryption of a message sent from A to B.

The exact information held by a block in the blockchain varies from implementation to implementation. In addition to a section holding the actual data of the transactions, a block would typically include a header holding these fields of information [32], also illustrated in Figure 3.4:

- The block number
- The hash value of the previous block's header
- The hash representation of the data in the block
- A timestamp
- The block size

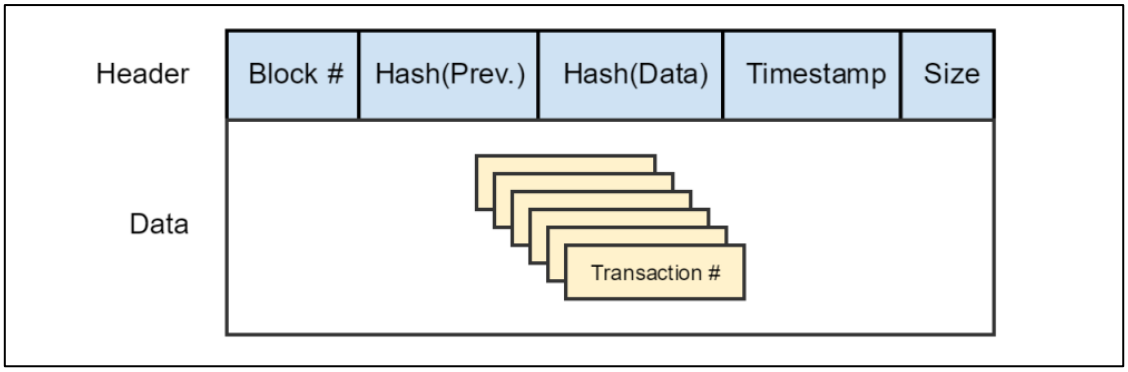

Figure 3.4: A typical block implementation with a data section holding transactions and a header section holding related metadata.

The hash computed for a block is dependent on the data stored in the block, which means that a unique hash is computed for each block. This means that if you modify a block that is already on the blockchain, the hash for this block will change. All subsequent blocks would therefore have to regenerate their previous hash field. Regenerating all the subsequent blocks is a computationally expensive operation and requires a substantial amount of time and resources to succeed.

More importantly, due to the security characteristics of the hash function, it is computationally infeasible to generate blocks that match hashes in the existing blocks. In other words, a malicious node will not be able to recompute all succeeding blocks and it will not be able to convince every other node in the network that its version of the blockchain is the correct version, without having acquired some necessary majority share in the network.

3.2.2. Distributed Consensus

Consensus mechanisms are essential for reliable distributed computing systems and are important components in blockchain networks. Consensus algorithms are responsible for reaching and maintaining consensus in a distributed network, such as a blockchain network. Consensus in a blockchain network is concerned with making sure that the next block that is added to the blockchain is a valid block and that all attempts from malicious or malfunctioning nodes to spoof participants with false blocks are disregarded [32]. When the majority of nodes in a network agree on a version of the blockchain, consensus is reached.

There are several situations in which a distributed network might not be able to reach consensus. A consensus algorithm might be vulnerable to some of these situations and tolerant to other. When choosing a consensus algorithm, it is therefore important to be fully aware of which situations might occur in a specific network. One of these situations is known as

Byzantine faults. Byzantine faults are conditions where it cannot be determined if a component has failed or not [33]. A Byzantine fault tolerant system is generally able to operate as long as the number of faulty nodes does not exceed one third of the total number of nodes in the network. A typical starting point for an implementation of a BFT (Byzantine Fault Tolerance) algorithm is the Practical BFT (PBFT) algorithm [33].

There are several different consensus algorithms deployed in various blockchain networks. The best-known algorithm is perhaps the Proof of Work algorithm [5] used in Bitcoin and several other cryptocurrencies. Proof of Work implements the task of block generation through a process in which nodes are required to solve complex cryptographic tasks, e.g. finding a specific value of which the hash output begins with a specified number of zero bits before the block can be successfully added to the blockchain [5]. When a node finds the correct value, other nodes in the network verify that the value is correct before adding the block to their version of the blockchain. As an incentive to complete the task, nodes typically get a certain amount of cryptocurrency as a reward for successfully completing it.

Although Proof of Work is still a popular consensus algorithm, it is criticized for its huge computational resource requirements which result in a huge waste of energy [34]. Another popular algorithm which requires considerably less computational resources is the Proof of Stake algorithm. In a Proof of Stake algorithm, validators are selected based on their economic stake in the network, e.g. the amount of cryptocurrencies they are in possession of and how long the currency has been in their possession [34], and not on their ability to complete cryptographic tasks.

3.2.3. Smart Contracts

Smart contracts are small programs installed on the blockchain, typically executing some sort of business logic in an automatic response to a change in the blockchain or the network topology [32]. Most blockchain platforms offer some implementation of smart contracts. The original purpose of a smart contract was to represent traditional written contracts in a way that removes the need for trusted third parties, such as a lawyer, to make certain that the criteria of a contract is fulfilled [32]. In practice, however, a smart contract can be any kind of program executing business logic that makes sense to install on a blockchain.

Most commonly, a smart contract specifies a set of constraints that must be fulfilled in order for the program to execute. Figure 3.5 shows a smart contract taking a set of inputs and producing an output in the form of a transaction. When installing the program on the immutable

blockchain, we ensure that these constraints cannot be tampered with [32]. This erases the need for a trusted third-party to validate that the requirements of a contract have been successfully fulfilled. However, the fact that the smart contract is installed on a blockchain also means that it can be difficult to correct bugs in the program, especially in public blockchains where it can be difficult to get all involved parties to agree on a new version of the program.

Figure 3.5: A smart contract executing a program that takes an input and produces a matching output.

Ethereum [11] is an open source public blockchain platform that became popular mainly due to its implementation of smart contracts. Ethereum smart contracts are written in a programming language called Solidity and allow users to add their own functionality to the Ethereum blockchain [11]. Although Solidity is developed by a team of Ethereum project developers, it is also used as the programming language for smart contracts in several other blockchain platforms. Several general-purpose programming languages, such as Java and Python, can also be used for writing smart contracts in some blockchain platforms [35].

3.2.4. Permissioned vs Permissionless Blockchains

Permissioned blockchains do not allow for public unauthorized access to the blockchain, which means that every node in the network must be authorized before they can access it, as opposed to in a permissionless blockchain network where everyone is free to participate [19]. In all blockchain networks, data stored in a block is visible to every node that is part of the network. This stems from the fact that to verify a block, nodes must be able to view the data within the block. This means that all non-encrypted data placed on the blockchain can be viewed by every node in the network. For permissioned blockchains, the nodes seeing the data will of course be limited to authorized nodes.

Permissioned blockchains are usually domain specific and aimed towards comparatively smaller group of participants, which also means that the length of the blockchains are typically shorter than for public blockchains. This allows enterprises to store

larger amounts of data in each block without harming network performance. On a public blockchain, the amount of data must be limited to avoid storage and processing issues as the blockchain grows exceptionally large.

The integrity of both permissioned and permissionless blockchains are maintained by consensus algorithms, which provide a measure for deciding which is the correct version of the blockchain and prevent any attempt from malicious or malfunctioning nodes to corrupt the network [19].

3.3. Hyperledger

Hyperledger is an open source blockchain project consisting of several blockchain related frameworks and tools managed by the Linux Foundation. By the start of 2019, the project comprises six different frameworks for deploying blockchains, along with seven tools for benchmarking, deployment, modelling, analyzing, ledger interoperability and cryptography [37].

- The frameworks are described by Hyperledger as follows:
- Hyperledger Burrow - Permissionable smart contract machine (EVM)
- Hyperledger Fabric - Permissioned with channel support
- Hyperledger Grid - WebAssembly-based project for building supply chain solutions
- Hyperledger Indy - Decentralized identity
- Hyperledger Iroha - Mobile application focus
- Hyperledger Sawtooth - Permissioned and permissionless support, EVM transaction family

Each Hyperledger framework is targeted for different use cases and user groups. Common for all the projects is that they bring something unique to the group and that they are applicable to companies operating in vastly different business sectors. The tools provide additional functionality to the frameworks and are described as follows [37]:

- Hyperledger Aries - Infrastructure for peer-to-peer interactions
- Hyperledger Caliper - Blockchain framework benchmark platform
- Hyperledger Cello - As-a-service deployment
- Hyperledger Composer - Model and build blockchain networks
- Hyperledger Explorer - View and explore data on the blockchain
- Hyperledger Quilt - Ledger interoperability
- Hyperledger Ursa - Shared cryptographic library

3.3.1. Hyperledger Fabric

Hyperledger Fabric was originally initiated by IBM and is currently one of the frameworks under the Hyperledger umbrella. The framework specifies a permissioned blockchain. The main features of the Fabric framework, that for the most parts are not found in other frameworks, are as follows [12]:

- Modular support for consensus protocols
- A three-step transaction flow where each step can be run on a different entity
- Support for smart contracts written in standard general-purpose programming languages
- Support for private sub-ledgers, known as channels
- Configurable and modular membership services

Hyperledger Fabric incorporates a unique three-step transaction flow that is not found in other blockchains. The transaction flow is made up of an execute-order-validate architecture, comprising an endorsement step, a block creation step and a validation step [12]. The tree steps are illustrated in Figure 3.6, where each step is indicated by a different color. In the endorsement step, nodes are required to replicate the transaction in their version of the blockchain, to see if the same output is produced. If the selected nodes return the same result, the transaction is endorsed. The next step is to place the transaction within a block. Nodes in the network then validate the block before adding it to their blockchain [12].

Another unique concept in Fabric is the channel. A channel is a private ledger which provides data isolation and confidentiality [12]. Only authorized nodes can interact with a specific channel. There can be several channels in a single Fabric blockchain network, meaning that nodes that require private information to be exchanged between themselves can create their own separate private channel in addition to being part of the main channel.

Fabric's modular architecture causes many of its components, such as the mechanism of consensus and membership services, to be pluggable and configurable [12]. All main components of a Fabric network operate in their own separate environment, such as a Docker container [38]. A malfunctioning component can be replaced simply by stopping and tearing down its container, and thereafter bring up a new container to replace it.

Another Hyperledger framework sharing parts of the same design philosophy as Fabric is the Hyperledger Sawtooth framework [39]. Hyperledger Sawtooth was initially under Intel development but was incubated in the Hyperledger project in 2016 [40]. Both Sawtooth and Fabric appeal to multiple use cases because of their modular architecture, meaning that they can be used in networks developed for a wide range of different business cases.

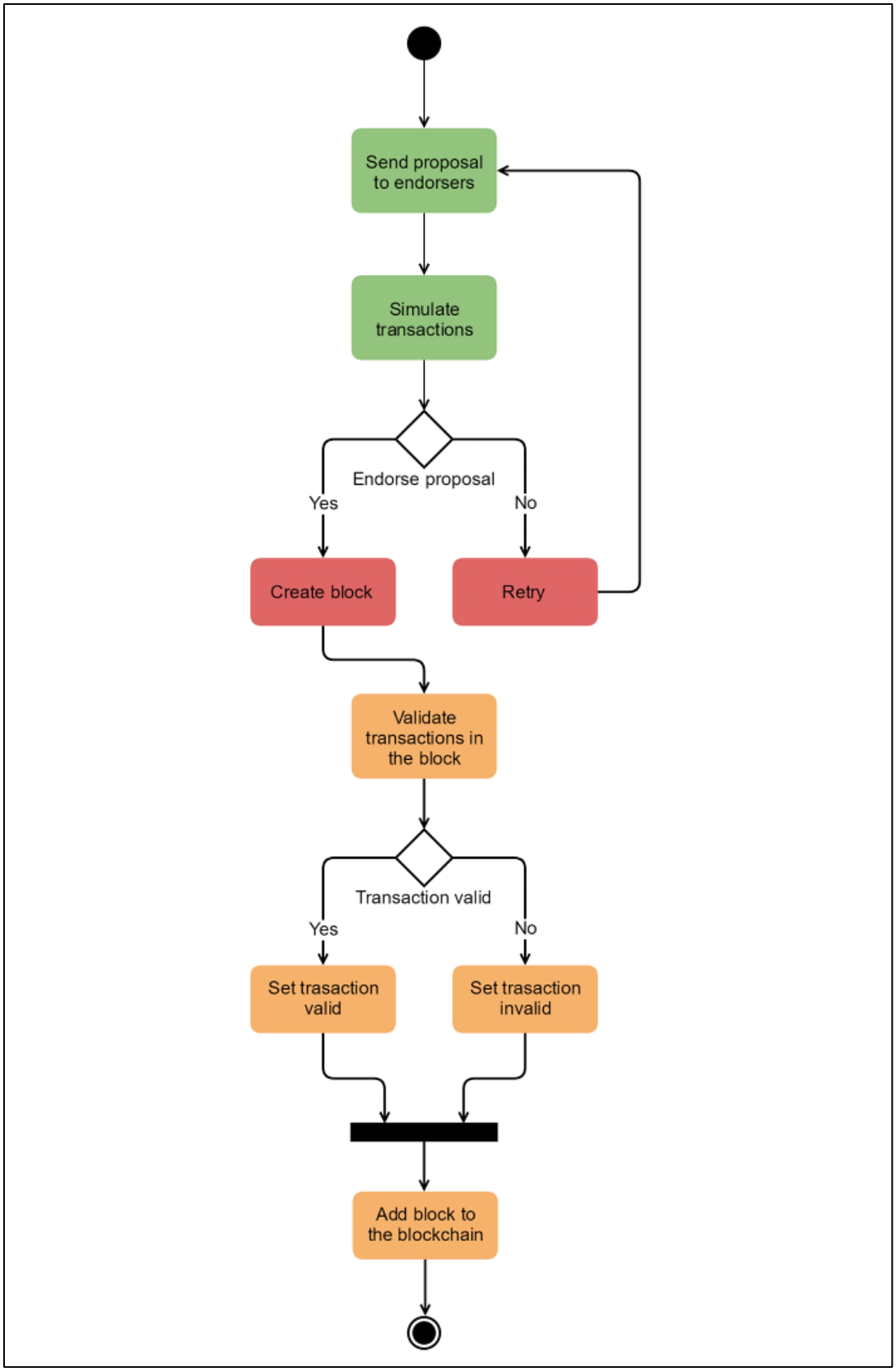

Figure 3.6: Process Illustration of the Fabric transaction flow where each step is represented by activities of a different colors.

3.3.2. Chaincode

Smart contracts in Hyperledger Fabric are known as chaincodes. Currently supported programming languages for chaincode are Go, Node.js and Java. However, chaincodes running on the same channel must all be written in the same language [12]. Nodes with chaincode installed on them are only aware of the name and version number of the chaincode and not which programming language it is written in.

A chaincode executes in its own container, separate from the node where the chaincode is installed [12]. However, the chaincode container is not created until the node receives its first chaincode request. This generally results in a significant delay for the first chaincode call, but reduces the computational resources required occupied by the network. Before we instantiate or upgrade chaincode on a channel, we must make sure that the chaincode is installed on the required number of nodes [41]. Multiple versions of a chaincode might be installed on a node at the same time.

The Fabric chaincode libraries for Go, Node.js and Java provide methods managing transactions proposed by applications [41]. Methods to invoke chaincode functions from within another chaincode are also provided.

There are three chaincodes, called system chaincodes, used in Fabric by default [41]:

- Lifecycle Chaincode (LSCC)
- Configuration Chaincode (CSCC)
- Query Chaincode (QCSS)

These chaincodes control various system functionality, e.g. controlling the process of installing user-created chaincode.

3.3.3. Node Definitions and Domain

There are three types of nodes in a Hyperledger Fabric network [41]:

- Client
- Peer
- Orderer

Nodes are defined based on the different roles they play in the network. Client nodes invoke blockchain events and transactions through peer nodes on behalf of the applications they represent [12]. Peer nodes hold the chaincodes instantiated on the channel and execute chaincodes involved in a transaction to validate that the proposed transaction produces the same chaincode output on each peer [12].

Each peer in the network is provided with a membership service provider (MSP). MSPs are used for managing identities for the nodes in the network. The peer uses the MSP to sign and validate endorsements when issuing a transaction or when verifying transaction proposals coming from other peers [12]. After enough peer nodes have signed off on the transaction, the client that proposed the transaction sends it to the orderer nodes for block creation. A block can contain a single transaction or a batch of multiple transactions. Peer nodes also validate the transactions constituting a block after the block has been created [12].

Nodes are operating under different organizations. An organization owns and operates a set of nodes in the network. For each organization there is at least one node operating as an anchor peer [42]. The anchor peer is visible for all organizations on the channel, allowing nodes from other organizations to discover it and communicate with other nodes in the organization as well [36]. To avoid single point of failure it is advised to have several redundant nodes of each type.

The orderer nodes are known collectively as the ordering service [12]. Orderer nodes are responsible for creating blocks. After a block has been created, the elected leading peer of each organization pulls the block from the ordering service and distributes it to each peer in its organization [12]. The leading peer of an organization can be set manually or dynamically. Dynamic leader election initially elects one peer for each organization as the leading peer. The leading peer sends updates to the rest of the peers in its organization regularly to show that it is still alive [42]. If peers stop receiving updates from the leading peer, they will elect a new leading peer.

The ordering service component is designed so that it is pluggable, meaning it can be changed based on the needs of the specific Fabric implementation. Currently, there are three types of ordering services officially implemented in Hyperledger Fabric: Solo, Kafka and Raft [43]. Several unofficial ordering service implementations also exist. The Solo ordering service rely on a single orderer node to create blocks and is not intended to be used in production environments [43].

The Kafka ordering service relies on an Apache Kafka cluster [44] to preserve data while the orderer nodes work on creating new blocks. The orderer nodes pull data from the Kafka nodes when they are ready to receive new data. The Kafka cluster relies on an ensemble of Apache ZooKeeper data nodes [45] to track the status of each node in the cluster. Data is replicated to all nodes in the cluster from a node selected as the cluster leader. If the cluster

leader goes offline, the ZooKeeper ensemble is used to elect a new leader. A network using the Kafka ordering service would typically organize all orderer nodes within a single orderer organization, as the decentralized benefit of spreading the nodes in multiple organizations will be violated by communication with the Kafka cluster [43].

Raft offers the same crash-fault tolerant leader-follower pattern [46] as Kafka but gives more in terms of decentralization and less administrative overhead. With Raft, orderer nodes are typically placed within each peer organization and dynamically assigned as a leader, follower or candidate [43]. Nodes initially start out as followers and self-promoter to candidate if the leader is no longer communicating. Nodes then vote for one of the candidates to be the new leader. The process is illustrated in Figure 3.7. A new Byzantine fault tolerant (BFT) ordering service based on the current Raft implementation is also in development [43].

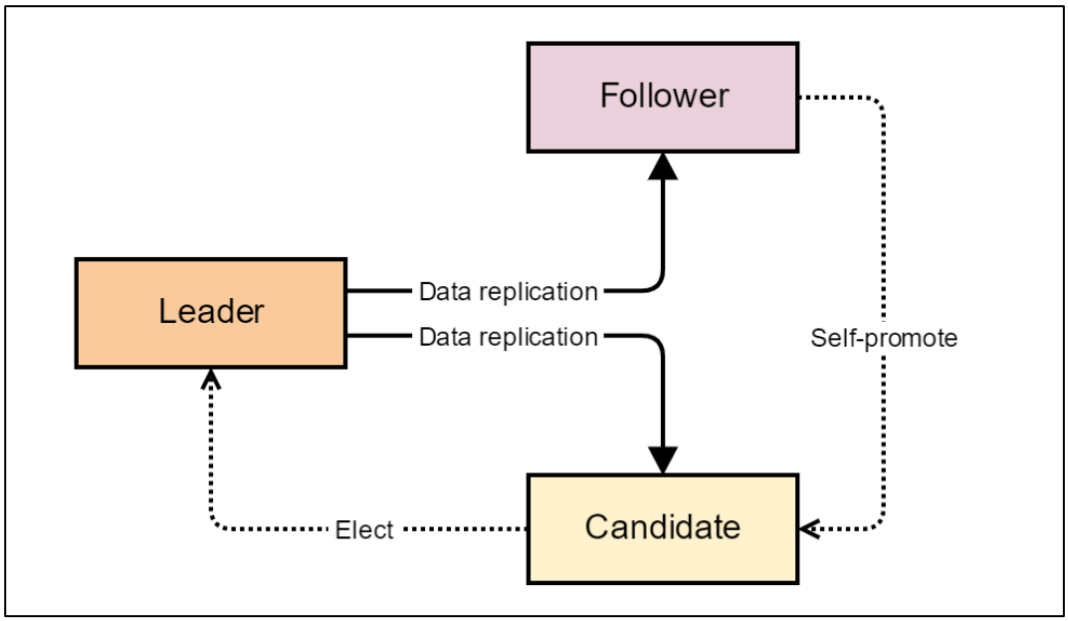

Figure 3.7: The roles and processes applicable to the leader-follower pattern.

3.3.4. State Database

A Fabric network stores data in key-value pairs. Peer nodes store this data in two places: on the blockchain and in the state database. The state database holds the latest value associated with every key, while the blockchain, which is the original immutable source of data, maintains the complete story of the key-value pair [47]. Each new transaction represents an update to a value in the state database. To find the original value of a key, we would search for the block containing the first transaction related to that key.

When a peer has successfully verified a new transaction, it updates the value of the key in the database. It is the value in the state database that is returned when a client queries the Fabric network [47]. A query on the blockchain is only executed when the state database must be recreated or if a historic value for the key is required. The current version of Hyperledger Fabric supports two options for state database: LevelDB and CouchDB [47].

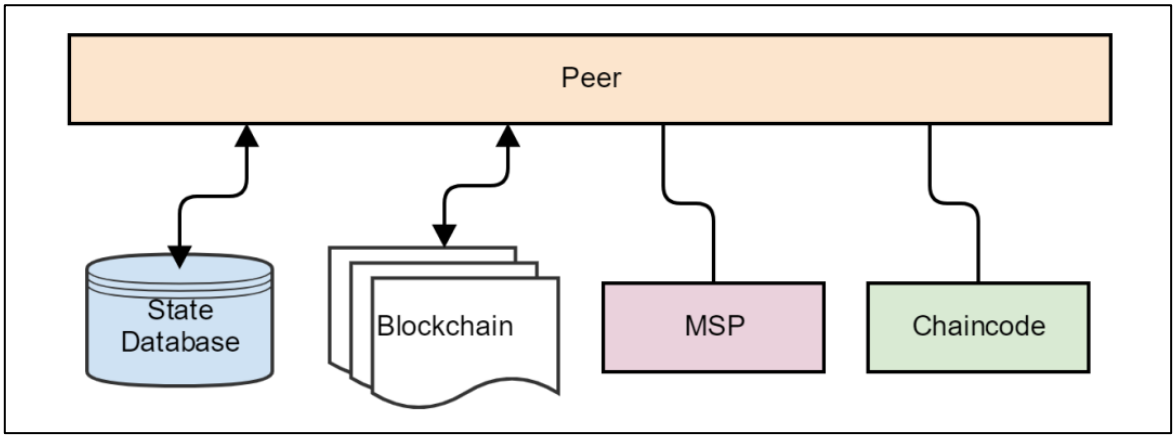

Figure 3.8: Peer nodes are composed of a state database, a blockchain copy, chaincodes and an MSP.

LevelDB is the default state database implementation in Fabric. It is a simple database capable of storing simple key-value pairs. A query to the LevelDB is a traditional query on the key. CouchDB, on the other hand, is an open source NoSQL document database [48] built to handle large amounts of data. Data in CouchDB is stored in JSON [49] format and rich queries using the CouchDB JSON query language are supported [47]. However, CouchDB instances run in separate containers and thereby impose more overhead on the system than the embedded LevelDB implementation.

3.3.5. Block Generation and Consensus

Blocks in a Fabric blockchain network are created by orderer nodes. However, these nodes simply create and distribute the blocks, and are not involved in validating them [12]. The task of validating transactions of a block is placed with the peer nodes. Consensus in a Fabric network is achieved in a process of validating a set of transactions to an endorsement policy [12]. As opposed to other blockchain implementations, consensus is not governed by a single algorithm but by the complete process of proposing a transaction and validating the created block.

Peers initially validate that a transaction has not already been submitted, that the signature is valid and that the client proposing the transaction is authorized to perform the

transaction. This process of transaction validation is known as endorsing a transaction [12] and is performed by selected peer nodes. These peers execute the same chaincodes used to generate the transaction, to check that the same output is produced. Which nodes are required to endorse a transaction is governed by the endorsement policy associated with the chaincode [12]. The endorsing peers execute the chaincode and send the results back to the client. The client then verifies the peer signatures and checks if all peers return the same result.

If the endorsement is successful, the client sends the transaction to the ordering service. No validation of the transaction is performed by the ordering service. The ordering service simply receives transactions and orders them in a block [12]. The block is then distributed via each organization's leading peer to all peers on the channel. To make sure that no changes have been made to the blockchain since the transactions were proposed and that the endorsement policy is fulfilled, transactions are validated by each peer when the block is distributed, and the individual transactions are tagged as valid or invalid. Peers then append the block to their blockchain, update their state database and alert the client if a transaction is deemed invalid [47].

3.3.6. Software Development Kits

Hyperledger Fabric currently offers SDKs for applications developed in Java and Node.js, while SDKs for Python, REST and Go are also in development [36]. The names given to the SDKs in Java and Node.js are as follows:

- Java SDK for Hyperledger Fabric
- Hyperledger Fabric SDK for Node.js

The SDKs provide methods for applications to manage Hyperledger Fabric channels and chaincode, e.g. ordering transactions, querying blocks, listening for events and discovering other nodes in the network. Without the use of an SDK embedded application, these features must be invoked by accessing the application programming interface (API) of the Fabric components directly from the command line [41].

The SDKs do not provide features for persistence and application developers must therefore implement such features themselves, e.g. the embedded application must implement its own method to listen for endorsing peers before ordering a transaction and for peers to validate transactions in a block. If an application sends a transaction request that is not correctly endorsed to the orderer service, the transaction will be deemed invalid in the validation phase after the block has been created and distributed to the peers [12].

Chapter 4

Proposed EHR Framework

CHAPTER 4: PROPOSED EHR FRAMEWORK

4.1. Blockchain Platform Selection

Before designing a blockchain system we first have to decide that whether the system should be based on permissionless blockchain or permissioned blockchain. Almost all the researches we have discussed in the related word have used Ethereum blockchain which is permissionless blockchain platform. In opposition to those I have used Hyperledger for the development of our Electronic medical records system which is a permissioned blockchain platform. Here are some reasons that why we prefer Hyperledger (permissioned blockchain) over Ethereum (permissionless blockchain):

- As in permissionless blockchain anyone can join the network anonymously without permission, that can be problematic. In case of electronic medical records system, identities of all participants of the network must be known. Therefore, it makes sense to use Hyperledger that is a permissioned blockchain platform and where the identities of every participant are known.
- Patients medical information are of highly sensitive nature. In Ethereum data recorded on the distributed ledger becomes visible to all participants of the network because they all take part in consensus mechanism, which is very problematic in case of medical data which requires more serious degree of privacy. Hyperledger permissioned blockchain take proper care of privacy requirements of medical data of patients where data is visible only to nodes authorized by authorities because only those authorized nodes takes part in consensus mechanism.

Ethereum use proof-of-work (PoW) algorithm which is computationally expensive consensus mechanism that require lot of resources for mining and involves fees for transaction execution that could limit the usability of the system. On the other hand, Hyperledger which is a permissioned blockchain uses consensus mechanism that are not computationally expensive. Moreover, unlike Ethereum, Hyperledger do not involve any cryptocurrency.

4.2. Proposed Architecture

Our proposed system is based on permissioned blockchain for efficient storage and sharing of electronic health records (EHRs) which provides better security and privacy of data.

Fig. 4.1 presents the network architecture of our proposed EHR system. The application focuses three type of users: Patients, Healthcare-Providers and Health Administration. Health administration will be responsible for the registration of patients and doctors. The application framework includes Membership Management, User Interface, Distributed Consensus, Smart Contracts (business logic), Data Storage.

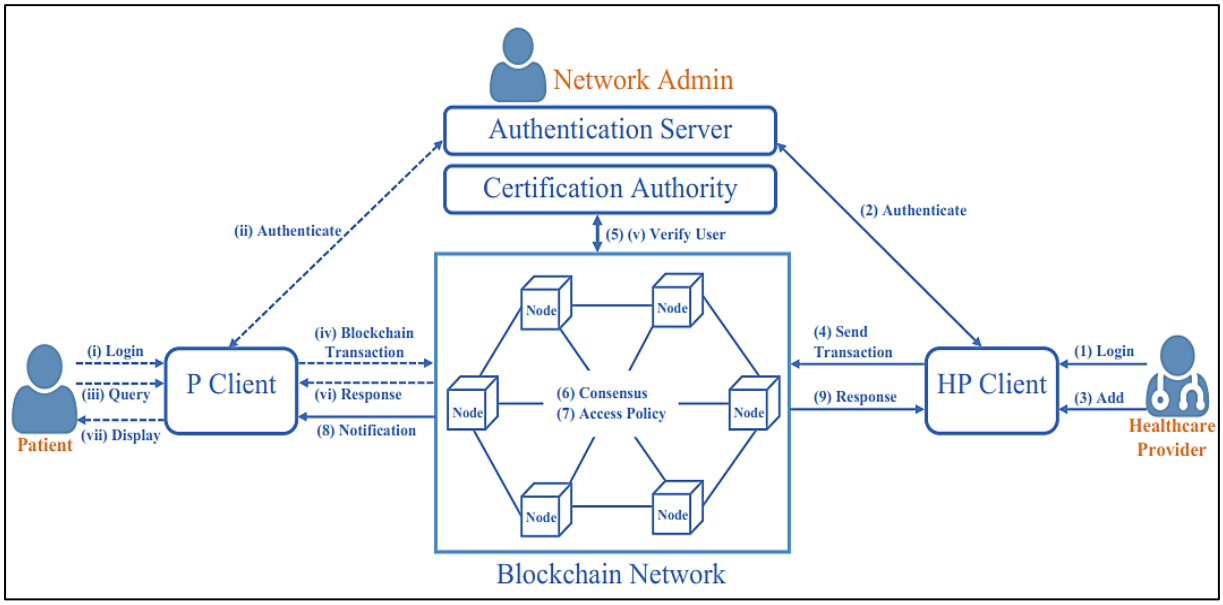

Figure 4.1: Network architecture of the proposed EHR system

4.2.1. Membership Management

In membership management the health administrator registers users i.e. Patients and Healthcare-Providers to the membership service based on their roles. During registration, health administration should make sure that only valid user should be register in membership service. For example, in case of Healthcare-Provider registration they should ensure that he/she is a qualified doctor and must be registered with the government health organization. The membership service also hosts a certification authority that generate key pair for signing and encryption key pair for every user. Patient is issued with a symmetric encryption key (Patient key) which is used for encryption/decryption of patient medical records. When a patient wants to share his medical records with a Healthcare- Provider, the patient can share his/her patient key using the public key of that Healthcare-Provider. Healthcare-Provider can also request this key from patient and when provided he/she can access patients' medical records and can add new records.

4.2.2. User Interface

Our system provides a user interface for every user through which they can interact with the system. The frontend web application is written in HTML, CSS and JavaScript. All the users are provided with their own separate web user interface. Both patients and Healthcare-Providers will use their login credentials (provided by the admin) to login to the system. User interface is further explained in details in later chapters.

4.2.3. Distributed Consensus

Consensus mechanism is the most important part of our blockchain application to verify transactions. All peers that takes part in the consensus mechanism runs a consensus algorithm (Hyperledger use PBFT consensus algorithm) to check whether a transaction is valid or not. If certain number of peers reaches a consensus the transaction will be successful and the transaction will be added to the blockchain. Here in our system we have set a network that will consists of three organization each having two peer nodes that will act as endorsing peers and committing peers at the same time and one orderer node in each organization that will provide the ordering service. Four out of these six peers must reach a consensus for transaction to be successful and added to the blockchain. Every peer node will hold ledger and the chaincode (chaincodes are written in JavaScript) along with its World-State database. Transactions submitted by users are received by the nodes through role-based APIs. When a transaction is submitted by the user, the leader node organizes transaction in a block and initiate the consensus mechanism. All nodes execute the transaction according to implemented chaincode logic. After successful execution the endorsing peers send the endorsement responses to the client. The client now sends the transaction attached with endorsement response to the orderer node which host the ordering service. Ordering service receives the endorsed transactions and orders them into a block. Now it broadcast the generated block to all peers. Every peer verify that the transactions of the received block are signed by appropriate endorser and that enough endorsements are present. If, the verification check passes the peer commit/save the block to their ledger.

4.2.4. Smart Contracts

There are three smart contracts proposed in the proposed methodology: Administration contract, Health-providers contract and Patients contract. All these contracts must be implemented as Hyperledger Fabric chaincodes. The Administration contract holds metadata for each record in a database, e.g. information about the administration and maintainer of the

record, an access control list of who is authorized to access the record and a log of events that has happened to the record. The Health-providers contract holds a list of provider-patient relationships and references to the corresponding metadata. Meanwhile, the Patients contract holds patient-provider relationships. A patient-provider relationship exists if a patient has a record stored by the provider.

4.2.5. Data Storage

The data is stored in the Hyperledger Fabric distributed ledger which stores data in two ways: the blockchain that contains the chain of blocks with each block holding transaction information in the form of key-value pair and World-State database that stores value (asset) of all the last committed transactions according to the specific key. Here a point to be noted is that it is not feasible to store all data in the blockchain as it largely degrades the performance of whole blockchain system. So, the blockchain will only hold transaction information and the World-State database will hold data values (actual data). In our system CouchDB is used as World-State database. Every peer in the network will hold a copy of ledger that includes blockchain and the World-State database.

Chapter 5

Implementing the Framework

CHAPTER 5: IMPLEMENTING THE FRAMEWORK

5.1. Overview

The results presented in this thesis are produced by combining a proposed blockchain-based EHR framework with an open source blockchain implementation. The chosen blockchain implementation is Hyperledger Fabric [50]. The design of Hyperledger Fabric induces several design alterations and adjustments to the proposed EHR framework. Any adjustments made to the framework will be contemplated and discussed in the thesis.

In this section we present the EHR framework and the options of implementation that are available. Essentially, we present the methods required to produce the results presented in Chapter 6. Further discussion and justification on the chosen implementation design and framework alterations will be presented in Chapter 7.

5.1.1. The Proposed Framework

The EHR framework proposed in this thesis provides an interesting starting point for implementing a blockchain network to provide improved security, privacy and scalability of EHR systems.

There are smart contracts proposed in the proposed methodology: Administration contract, Health-providers contract and Patients contract. All these contracts must be implemented as Hyperledger Fabric chaincodes. The Administration contract holds metadata for each record in a database, e.g. information about the administration and maintainer of the record, an access control list of who is authorized to access the record and a log of events that has happened to the record. The Health-providers contract holds a list of provider-patient relationships and references to the corresponding metadata. Meanwhile, the Patients contract holds patient-provider relationships. A patient-provider relationship exists if a patient has a record stored by the provider.

5.1.2. The Blockchain Implementation

The relevant open source blockchain projects currently available are the Hyperledger projects [51] and Ethereum [11]. The EHR framework on which this thesis is based on calls for a domain-specific permissioned blockchain with an incentive mechanism that is not driven by rewards in form of cryptocurrency or other economic stakes. This means that the framework

is best suited to be implemented in one of the Hyperledger projects, as opposed to on the public and economically incentivized blockchain provided by Ethereum.

All Hyperledger open source projects are under continuous development, which makes much of the projects' documentation rapidly outdated. Documentation for all Hyperledger projects and their different versions can be found online [51]. This thesis makes use of Hyperledger Fabric v1.4.1, released April 11th, 2019 [50]. The most interesting feature introduced in v1.4.1 is the new Raft ordering service.

Information on open development issues and potential vulnerabilities of the various Fabric versions are found on the Hyperledger Fabric issue tracking website [52]. The results presented in this thesis are produced using some features that were introduced with v1.4.1 and the results can therefore not be reproduced in earlier versions.

The Fabric and Sawtooth projects were the first two codebases selected for incubation in Hyperledger [40]. Both projects provide implementations that are mature and production ready. Some of the most prominent differences between the current versions of the two projects are [12], [39]:

- Fabric supports strictly permissioned blockchains, whereas Sawtooth supports both permissioned and permissionless blockchains
- Fabric implements a unique transaction flow for achieving consensus in the network, whereas Sawtooth implements a traditional transaction flow and consensus algorithm
- Fabric supports channels for private transaction data between subgroup of nodes, whereas in Sawtooth data from every transaction is visible to all nodes

From studying the description of both blockchain projects, it is reasonable to suggest that the proposed EHR framework can be implemented effectively using any of the two blockchain implementations. However, Fabric's flexibility regarding consensus and incentive mechanisms, as well as pliant membership governance, make it useful for our implementation of the EHR framework and the prototyping application. Support for private channels is also a valued feature for potential future development, e.g. allowing analytics companies to analyze only parts of the data through private sub-ledgers instead of the full ledger.

5.2. Hyperledger Fabric

The binaries for Hyperledger Fabric are hosted in a GitHub repository [50]. There is currently no proper installer provided with the binaries. Instead, a script named bootstrap.sh,

which is included in the repository, can be used to install the binaries along with some sample applications. See more about the requirements for installing the Fabric tools and binaries in the repository's README file [50] or in the prerequisites section of the Fabric documentations website [53]. For information on the development setup used in this thesis, see Appendix A.

A Fabric blockchain network includes three modular and pluggable components of special interest for developers [12]:

- An Ordering Service
- A Certificate Authority
- Membership Service Providers

In addition to these channel-wide components, each peer node in the network is composed of several other modular and pluggable components, e.g. state databases and chaincodes, which are presented in section 3.3.4 and 3.3.2.

5.2.1. Fabric Tools

Two software tools are supplied with the Fabric binaries:

- Crypto Generator (cryptogen)
- Configuration Transaction Generator (configtxgen)

The Crypto Generator tool generates certificates and signing keys for the identities participating in the network [54]. These certificates and keys enable entities to sign transactions and verify identities. The tool can be configured in a YAML [55] configuration file, which is consumed by the tool upon execution. This provides a quick and simple way to produce cryptographic material for use in a development environment. In a production environment, however, a certificate authority (CA) will typically be used for generating the cryptographic material [12].

The Configuration Transaction Generator tool creates our genesis block and other subsequent configuration blocks [56]. Configuration blocks hold only configuration transactions, not regular transactions. The tool is configured in a YAML configuration file, where we specify the ordering service, anchor peers, MSPs, organizational policies and channel wide policies, which were introduced in section 3.3.3. The policies specified in this file are base policies and may be overridden by e.g. specific chaincode policies. Essentially, the policies specify which certificates are required to sign the data for a signature to be valid.

5.2.2. Software Containers

Containers for Fabric entities are created with Docker [38]. Running entities in isolated environments provided by container software is a good way to simulate distributed behavior, even if all entities are in fact running on the same physical machine in a development environment. Each container simulates an entity that could just as well be running on another machine in another physical location, as it would in a production environment. Running entities in containers also eases administration and maintenance of entities, as faulty entities can be removed and replaced quickly.

Hyperledger Fabric provides ready-made docker images for starting the different types of entities making up a network [50]. The types of Fabric entities used for this thesis are (Docker image names):

- fabric-peer: A peer node
- fabric-orderer: An orderer node
- fabric-ca: A Fabric CA
- fabric-couchdb: A CouchDB instance
- fabric-ccenv: System environment used to build chaincode
- fabric-nodejsenv: System environment used for JavaScript chaincode
- fabric-tools: System environment for running software tools

The initial configuration of the individual entities is described in YAML configuration files used as input to the Docker Compose tool [57] when the network is first initialized. Docker Compose consumes the files and creates the specified Docker containers. The Docker images provided for each type of Fabric entity ensure that entities are ready to join the blockchain as soon as the required containers are up and running, without needing to install any additional software.

5.2.3. Ordering Service and Consensus

The ordering service comprises a set of orderer nodes, collectively known as the ordering service. Orderer nodes can be organized in a single orderer organization or as members of peer organizations. Figure 5.1 shows the various nodes making up a peer organization. Configuration of the orderer nodes depends on the type of ordering service used in the network. More information on ordering services is provided in section 3.3.3 and 3.3.5.

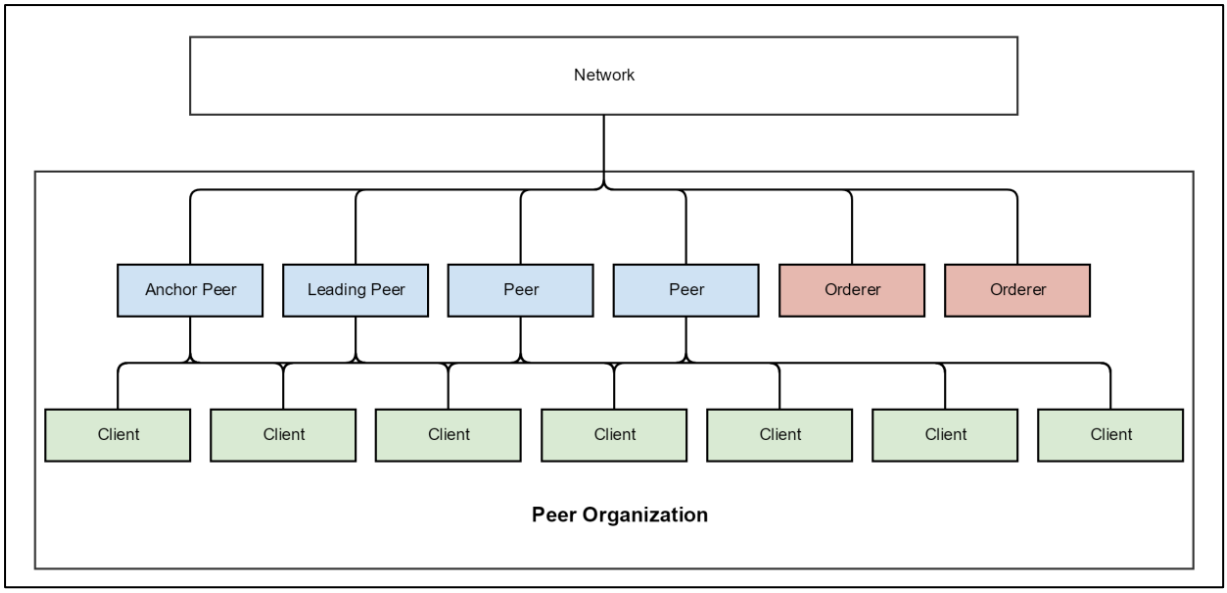

Figure 5.1: An illustration of the communication hierarchy within a peer organization with four peer nodes, two orderer nodes and multiple clients.

For a network using the Kafka ordering service, which until recently was the default ordering service implementation for production-ready systems, orderer nodes are typically placed within a single organization. This is due to the fact that orderer nodes must communicate with the Kafka cluster. This breaks any decentralization benefits gained from multiple orderer organizations. Although organizing nodes in a single organization does not limit where the actual physical nodes are placed, it is considered a centralized approach in terms of policy specifications and administration.

With the introduction of the Raft ordering service in Hyperledger Fabric v1.4.1 [43], communication with an intermediate node cluster is no longer required. It therefore makes sense to place orderer nodes in peer organizations, as opposed to organizing every orderer node in a single orderer organization. Spreading orderer nodes in different organizations increases the decentralization aspect of the network. All organizations that regularly participate in the network should provide orderer nodes to the ordering service.

As described in section 3.3.5 and visualized in Figure 5.2, the transaction flow of a Fabric blockchain network, for any type of ordering service, is as follows:

1. The client issues a transaction proposal
2. The peer representing the client sends the proposal to required endorsers
3. The client checks if the proposal is correctly endorsed
4. The peer representing the client sends the transaction to the ordering service
5. The ordering service creates a block, likely containing several transactions
6. The block is distributed to the leading peer of each organization

7. Leading peers distribute the block to the rest of the peers on the channel
8. Peers validate the transactions of the block before adding it to the blockchain

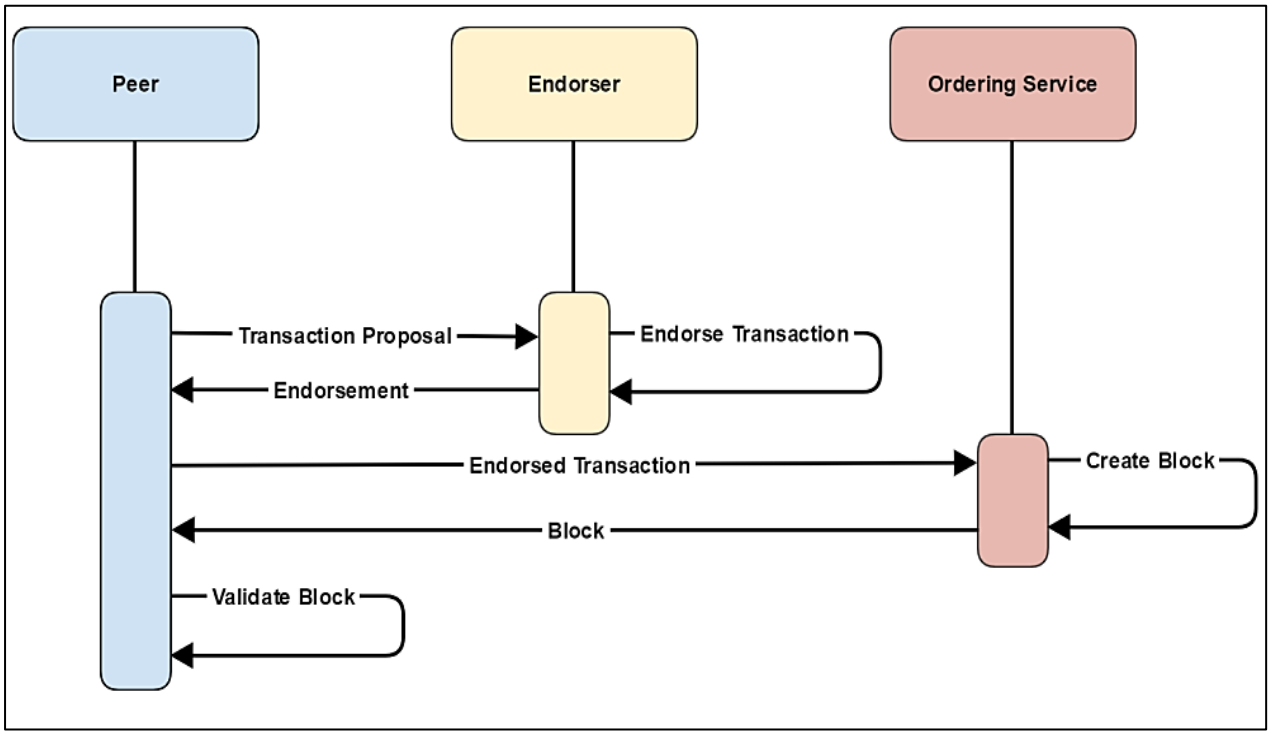

Figure 5.2: Hyperledger Fabric transaction flow featuring a single endorser.

If a transaction is not validated, it is marked as invalid when the peer places the block on its ledger. Clients invoking a transaction must therefore listen to transaction events even after the transaction has been sent to the ordering service, to make sure that the transaction was verified by the peers.

5.2.4. Certificate Authorities and Membership Services

Certificate authorities (CAs) handle identity registration and digital certificates for Fabric networks [12]. Entities communicating in the network identify themselves using certificates issued by one of the CAs in the network. The entities validating the certificates are the MSPs, as introduced in section 3.3.3.

The CA is a pluggable component and multiple CAs can be used in a network at the same time, e.g. one for each organization. Fabric provides a default CA implementation known as the Fabric CA [58].

The configuration of each MSP is what enforces the policies specified in the network. Whereas a CA generates the required keys and certificates for an entity, the MSP is used to validate the credentials when an entity communicates with the network [12]. MSPs also enforce

role checks on whether an entity is e.g. a client, member or admin of the domain. Policies might require that a certain number of entities of each role signs off on a transaction. The trust domains for each organization is specified by the MSPs based on which CA is authorized to issue credentials to members of that specific trust domain.

MSPs are part of the channel configuration and are kept synchronized with the consensus mechanism. There is one MSP for each organization in the channel. Local MSPs are also defined on each node in the network. These local MSPs control e.g. which entities can install chaincode on a peer.

5.2.5. Network Discovery

Peers in the network discover each other using the service discovery [59]. The discovery process uses anchor peers associated with each organization to explore the network and discover peers belonging to other organizations. This eliminates the need to provide static information about each peer in the network. For a peer to be visible to the service discovery process, it must have an external endpoint set in its configuration [59]. It must also know the address of at least one other node in its organization, which again must know the address of a different node. In this way, every node in the network is known to at least one other node, and the complete network can therefore be discovered.

The service discovery process uses information from the gossip protocol to identify connected peers. The gossip protocol continuously identifies which peers in the network are online or offline. It also broadcasts ledger data to other peers on the channel, so that peers that are out of sync can copy any missing blocks [42]. When a new block is created by the ordering service, the leading peer of each organization gossips this block to the rest of the peers in its organization. The protocol also allows for new peers on the channel to transfer ledger data over peer-to-peer connections [42].

5.3. Node.js SDK for Hyperledger

The Node.js SDK for Hyperledger Fabric [13] provides developers with an API for developing Node.js applications for interaction with Hyperledger Fabric networks. The API offers routines related to service discovery and invoking chaincode methods through transaction proposals or query requests.

5.3.1. Communication Clients

The Node.js SDK provides two types of client classes [13]:

- Fabric-client - Hyperledger Fabric Client
- Fabric-ca-client - Hyperledger Fabric CA Client

The client classes are used for invoking methods to communicate with Fabric networks. A fabric-client object comprises several methods for invoking the chaincodes installed on the network. Methods in the client are invoked from user context. The user context is associated with an object that is of a class implementing the Node.js SDK user interface [13]. A class implementing the user interface must hold information about the associated id, userkey and roles. These fields of information are used by the CA when enrolling the user.

The fabric-ca-client class is used for handling events related to the CAs in the network, such as registering a new user. The Fabric CA implementation associate users with their roles i.e. as administration, healthcare-provider or patient.

5.3.2. Query Requests and Transaction Proposals

Transaction proposal requests are constructed from the logic class and invoked through a fabric-client object [13]. The requests hold the name of the chaincode and method that is to be invoked, along with any arguments required by the chaincode method. Queries are created from the query class [13], which hold the same information as specified for the logic class above.

The SDK do not provide methods for persistence [13]. As soon as a proposal is invoked through the fabric-client, no other measures are invoked by the SDK. The application developer must therefore develop methods for listening to responses from endorsers and nodes that validate the transaction.

5.3.3. Collecting Endorsements

The SDK provide an interface for discovering nodes in the network [13]. This means that addresses and hostnames of nodes in the network do not have to be supplied manually to the application. Whenever a peer needs to discover nodes for endorsing a transaction, it simply utilizes the network's service discovery mechanism, which is discussed in section 3.3.5. Service discovery then returns the names of the installed chaincodes, the selected endorsement policies and the name of available orderer nodes and endorsing peers.

The combination of endorsing peers can often be chosen in multiple configurations, depending on the endorsement policy selected for the chaincode. For instance, in a channel with three organizations maintaining two peers each, the following combinations of endorsing nodes can be selected for a chaincode that requires endorsement from at least one node from each organization:

1) Organization 1: Peer 1 - Organization 2: Peer 1 - Organization 3: Peer 1
2) Organization 1: Peer 1 - Organization 2: Peer 1 - Organization 3: Peer 2
3) Organization 1: Peer 1 - Organization 2: Peer 2 - Organization 3: Peer 1
4) Organization 1: Peer 1 - Organization 2: Peer 2 - Organization 3: Peer 2
5) Organization 1: Peer 2 - Organization 2: Peer 1 - Organization 3: Peer 1
6) Organization 1: Peer 2 - Organization 2: Peer 1 - Organization 3: Peer 2
7) Organization 1: Peer 2 - Organization 2: Peer 2 - Organization 3: Peer 1
8) Organization 1: Peer 2 - Organization 2: Peer 2 - Organization 3: Peer 2

The service discovery denotes these configurations as layouts. Each layout holds a list of groups, where each group holds a list of peers. Typically for most implementations, all peers within a group will be from a single organization. The layout also states how many endorsements are needed from each group. The SDK embedded application can then decide which layout it prefers and issue a transaction proposal to selected endorsers from this layout.

The application must adhere to the transaction flow presented in section 3.3.5, which means that after sending the transaction proposal to the endorsing peers, the application must wait for the endorsement responses before sending the transaction to the ordering service. If the application sends a transaction that is not correctly endorsed to the ordering service, the transaction will be marked invalid by the peers validating the block that has been created. To make sure that the transaction is validated by the peers, applications must listen to the channel for events even after the block has been distributed in the network.

Additional documentation of each class and method provided by the SDK can be found in the Node.js SDK for Hyperledger Fabric GitHub repository [13].

Chapter 6

System Description

CHAPTER 6: SYSTEM DESCRIPTION

6.1. Overview

The resulting software system consists of two stand-alone software packages:

- HLF network package
- Node.js application package

The HLF network package comprises the Hyperledger Fabric blockchain configuration and chaincode files. By running the scripts provided in this package, a functional blockchain network can be created. The package's intended place in a system is illustrated in Figure 6.1, where it is termed as the blockchain layer.

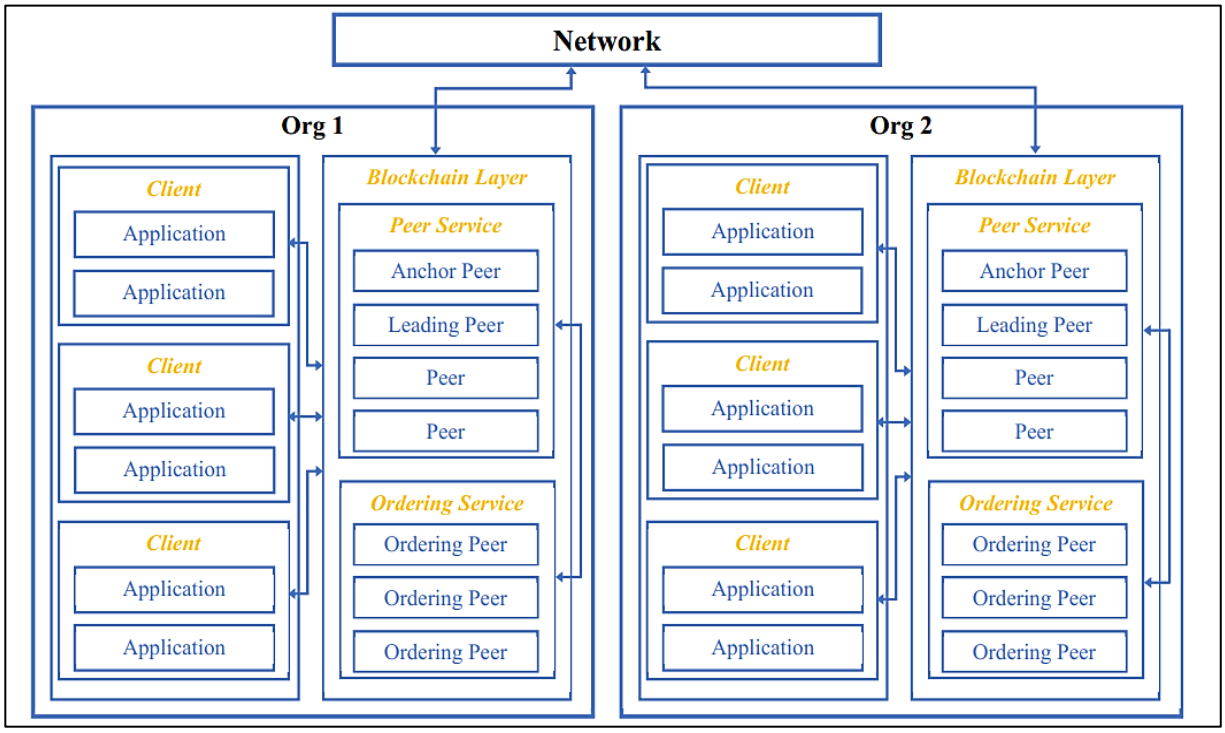

Figure 6.1: The architecture used when deploying the system, illustrated with a network of two providers.

The Node.js application package is merely developed to demonstrate the features of the HLF Network package and is not intended to be deployed in a production environment. It provides, essentially, a simulation of the database interface and can be replaced by any other application implementing a Fabric SDK. Not having to deal with a database and database management component makes demonstration of the blockchain network easier. However, the methods provided for communication with the HLF network can be used as a basis when implementing the Node.js SDK.

In addition to the components illustrated in Figure 6.1, a CA must be present in the network. Most commonly, the package would be configured to be using an existing CA in the system. If such a CA is not available in the network, a new CA can be deployed in the blockchain layer for merely accommodating the HLF network package.

Together, the HLF Network and Node.js application packages demonstrate the core functionality obtained by the EHR framework when using the Hyperledger Fabric blockchain model for implementation. Both packages are made available under the Apache License Version 2 (Apache-2.0) [60].

6.1.1. HLF Network Package

The configuration files in the HLF Network package comprise both container and channel configuration, as well as configuration for creating cryptographic material with the Crypto Generator tool provided with the Fabric binaries. In production environments, material created by the Crypto Generator can be replaced by cryptographic material created by a CA. Even if the package is mainly intended for systems managing EHRs, it is also applicable to other similar recordkeeping use cases.

Any authorized application utilizing one of the Fabric SDKs, e.g. the Node.js application package, can interact with the network and invoke chaincodes. Communication can also be initiated directly from the command line APIs of the various nodes in the network.

All nodes in the network, except the default Fabric CAs, have TLS enabled for secure communication and therefore only accept communication using the TLS protocol [61]. The CAs provided with the package are configured for development and testing purposes only. In a production environment, the CA implementation should be replaced with CAs that are already provided in the existing network.

6.1.2. Node.js application Package

The Node.js Application package comprises both a front-end GUI and back-end business logic for interacting with the Fabric blockchain. The part of the application executing the business logic required to communication with the Fabric network relies on the Node.js SDK for Hyperledger Fabric version 1.4.1 [13].

The GUI is developed with AngularJS and is loosely tied with the business logic part of the application. This means that the business logic can be re-used for purposes where a GUI

is not needed or where the GUI is replaced by some other form of interface. The application is created as a demonstration tool for showcasing the features of the HLF network package.

6.2. HLF Network Architecture

This section provides a design and implementation description of the components utilized in the HLF Network package. All main components are pluggable and can be replaced or edited, generally without rewriting other components. This makes the package flexible so that it can accommodate existing systems in the best possible way.

All entities are running in individual docker containers to simulate physically separated environments. For simplicity in testing for the thesis, all containers have been running on a single physical machine. The entities can, by specifying correct host information during configuration, be placed on any machine and communicate with each other over a network using TLS.

6.2.1. Fabric Tools

The package contains six YAML files when we configure our system to work in multi-organization environment. These files specify the architecture and configuration details of the network. The network is generated by the following configuration files:

crypto-config.yaml

configtx.yaml

compose-with-raft.yaml

compose-with-couchdb.yaml

base/compose-base.yaml

base/peer-base.yaml

crypto-config.yaml governs the creation of cryptographic material to be used by the nodes in the network and is consumed by the Crypto Generator tool. The file includes hostnames and alternative names for all peer and orderer nodes in the network. Material for specified CAs are also generated. However, a CA can also generate this material by itself. The CA would typically also be used to create cryptographic material for other nodes in the network, thereby making the material created by the Crypto Generator unnecessary. The

cryptographic material created by the Crypto Generator is placed in a directory named crypto-config.

configtxgen.yaml specifies channel configuration details, such as the type of ordering service, policies and MSPs, as well as addresses for each organization's anchor peers. The MSP configuration must provide the path to the directory holding the generated certificates. If the Crypto Generator tool is used for creating the certificates, this directory will be located in a sub-directory of crypto-config.

The remaining four configuration files are used with Docker Compose, a tool for configuring and initiating Docker containers [57]. compose-with-raft.yaml specifies names, network addresses and dependencies for the required Docker volumes and services, while compose-with-couchdb.yaml provides additional configuration for CouchDB containers. Each peer in the network requires an associated CouchDB container for the state database. The CouchDB configuration file should be consumed by Docker Compose together with compose-with-raft.yaml, if the network is to use CouchDB as the state database.

The files located in the directory named base are extensions to compose-with-raft.yaml. peer-base.yaml specify configurations that are common for all nodes of a specific type in the network, while compose-base.yaml provides individually dependent container settings. This include unique names and addresses for all containers. Environment variables for MSPs and file paths to the cryptographic material, including certificates for TLS, must also be provided in this file.

6.2.2. Chaincodes

Chaincodes for the HLF Network package are written in the JavaScript programming language. JavaScript objects used to represent transaction data in the chaincodes are stored as JSON strings on the blockchain. The JSON.parse() method is used for converting JSON strings to JavaScript objects and JSON.stringify() method is used to convert JavaScript object to JSON strings [62]. Writing both the chaincodes and the application in the same programming language allows for the same classes to be used in both packages if necessary.

The following three chaincodes have been implemented:

- Administration contract
- Health-providers contract
- Patients contract

Administration contract, Health-providers contract, and Patients contract chaincodes represent the three smart contracts proposed in the EHR framework and discussed in section 3.3.2.

In the record chaincode, a JavaScript class named record.js is used to represent the data that is stored on the blockchain. An object of this JavaScript class holds the following information:

- The ID of the healthcare-provider adding the record
- A unique reference to the record
- The timestamp of the last edit of patient records

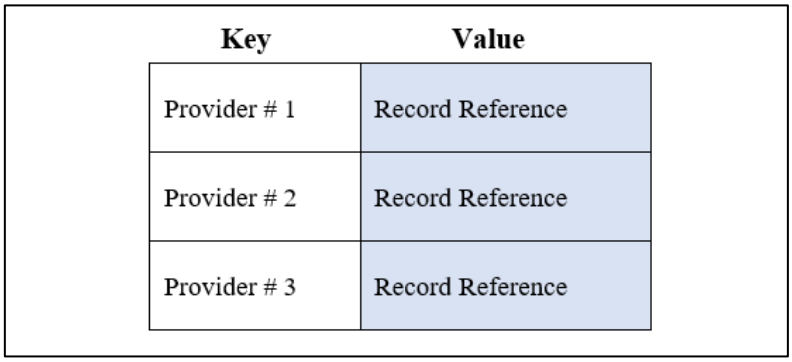

Figure 6.2: The representation of a record contract stored on the blockchain.

The object contains a map with the healthcare-provider ID as the key. The value of each entry in the map is a one-dimensional array of length 2, storing a reference to the record associated with the user-provider pair. Figure 6.2 shows a graphical representation for a record stored on the blockchain. The figure shows the record of a user with three patient-provider relationships. The record object is stored in the state database with the user ID as the key. The JavaScript object is serialized to JSON before being placed on the blockchain. Figure 6.3 shows the class diagram for the record chaincode.

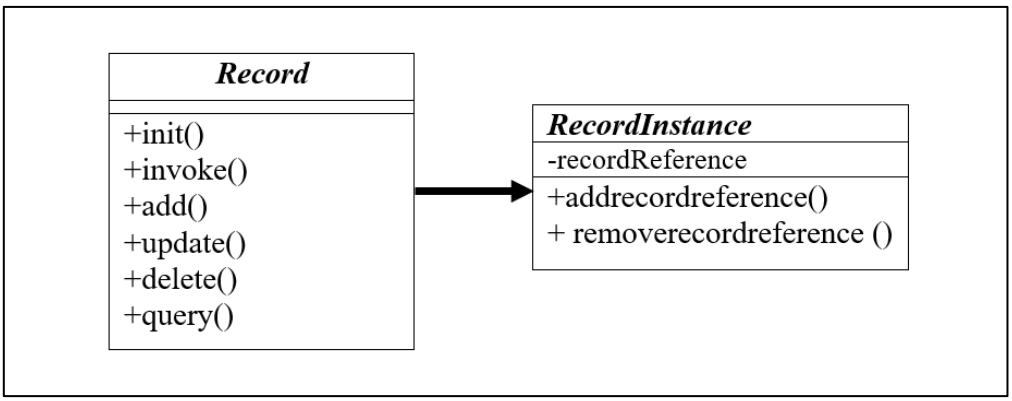

Figure 6.3: Record chaincode class

The results of a query can be presented in the form of the returned JSON string or as a JavaScript object after de-serializing the JSON string. A query for all provider relationships for a patient would simply include the user ID for the patient. The returned object would then hold the record reference and last edit timestamps for all providers where the patient has a record. The record reference is used to query the patientrecord chaincode to get the record associated with the reference.

The record class is used to represent the data of a patient record. Objects of the class are stored on the blockchain with a unique record reference as the key. A record object holds the following three pieces of information:

- An access control list of which entities are authorized to access the record
- The patient records
- A log of all events that has happened to the record

The access control list (ACL) is constructed as a map. Strings holding either a client ID or an MSP ID are used as keys and lists of events as the value. The types of events are defined in a JavaScript class and include events such as READ, WRITE and CREATE. The events listed for a client or MSP in the ACL control which types of events the entity is authorized to perform on the record.

Both individual client access and MSP-wide group access can be granted to a record. An MSP is typically associated with a single provider in a one-to-one relationship. To grant read access to all clients belonging to a provider named healthcare-provider1, we must add MSP healtcare-provider1 to the ACL with event type READ. To distinguish client IDs from MSP IDs, a suffix of "CLIENT" or "MSP" is added to the map key.

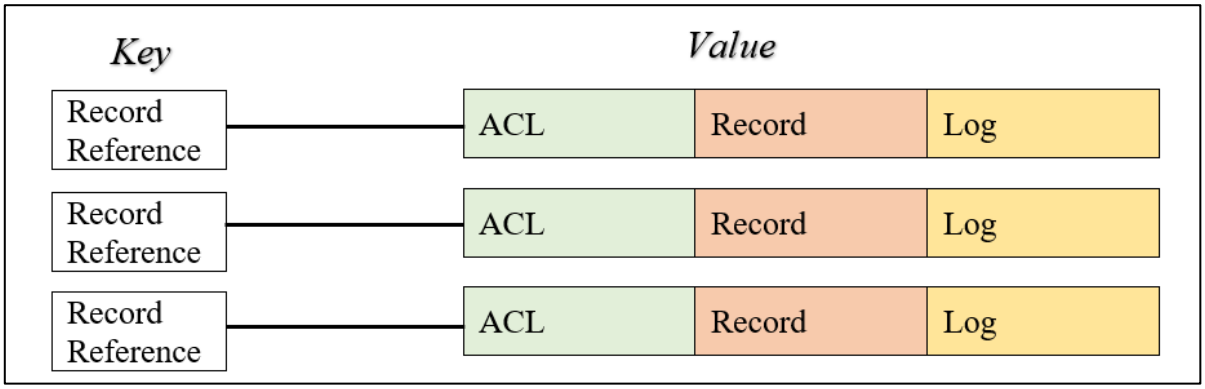

Figure 6.4: Record stored on the blockchain, which use the Record Reference as the state database key.

The RecordInstance object is saved in the state database using the Record reference as the key, as illustrated in Figure 6.4. This unique reference, however, cannot be created by a randomized generator in the chaincode, as this would cause each endorsing node to end up with a different reference and endorsement would therefore fail. The reference must instead be supplied by the application creating the Record. Figure 6.5 shows the class diagram for the Record chaincode.

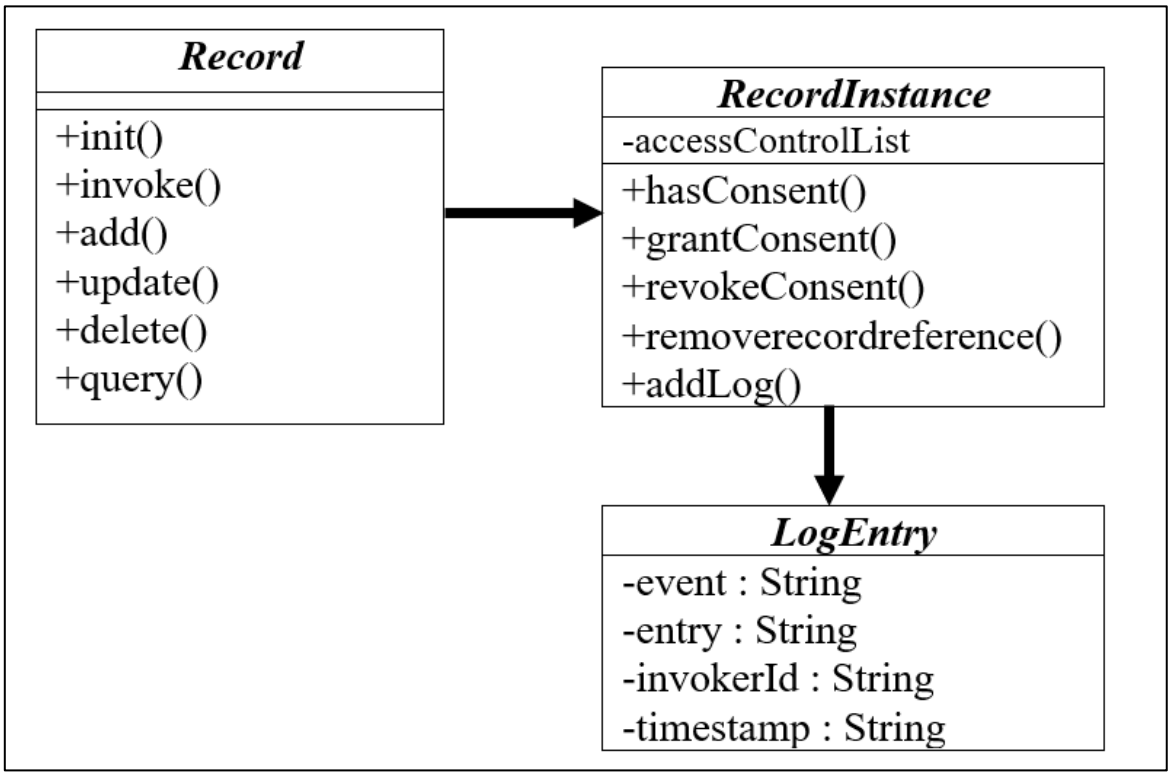

Figure 6.5: Record chaincode class diagram

The type of event that was executed must correspond with the events found in the ACL of the RecordInstance object. If a provider makes a query for reading the Record of a patient, the chaincode will first check if the ID of the client is found in the ACL associated with the Record. If the client is found in the ACL, the chaincode then checks if the client is authorized for the READ event.

If the client is authorized, a LogEntry object containing the type of event, the client ID and the timestamp of the invoked query is created and placed on the blockchain. Only when this procedure is fully completed is the Record returned to the client. Measures for encrypting the LogEntry should be implemented in future versions of the software package.

If the event was a WRITE, e.g. a healthcare-provider enters a few sentences about the latest session with a patient to the patient's record, the new data entered is included in the

LogEntry. Each time a WRITE event is added to the log of a Record, the last edit timestamp of the associated user-provider pair in the SC is updated. To find the creator of a Record, we query the RecordInstance for the LogEntry where the type of event equals CREATE.

Several internal calls between the chaincodes are used to invoke the required methods. For instance, when adding a new Record to the blockchain, the Record chaincode first sends a call to the SC chaincode. The SC chaincode checks if the user has an existing SCInstance object on the blockchain. If the user does not have an existing SCInstance on the blockchain, the SC chaincode must create a new SCInstance object.

The one-to-one relationship between patient records and providers means that there can only exist one Record reference for each patient-provider relationship. If there already exist a Record for the relationship, the SC chaincode will return an error. In a typical scenario where we want to check the log of a Record, we must first query the SC chaincode to get the Record reference and thereafter query the Record chaincode to get the LogEntry object.

6.2.3. Ordering Service

The network will be configured with Raft as the ordering service. As mentioned in section 3.3.3, Raft provides less administrative overhead compared with Kafka, as there are no additional Kafka and ZooKeeper nodes to manage. The lack of a Kafka cluster also improves the decentralization aspect of the network, as orderer nodes do not have to communicate with a single-organization Kafka cluster. For the Raft ordering service, it makes sense to place orderer nodes within peer organizations in the network, as opposed to in a single orderer organization, and each provider can then register as many orderer nodes as it finds necessary.

To ensure that each organization provide the same amount of resources to the network, a policy on how many orderer nodes each organization should provide to the network must be specified. A larger organization would typically invoke more transactions and a heavier load on the network and could therefore be required to provide more orderer nodes to the network. The three organizations specified in the HLF Network package are by default assigned one orderer node each.

The network requires at least three active orderer nodes to achieve crash fault tolerance for one orderer node. Increasing the total number of orderer nodes in the network to e.g. five, will increase the crash fault tolerance to two orderer nodes. However, with the decentralized ordering service approach used in the HLF Network package, it is important that the property

of crash fault tolerance do not depend on a single provider being present in the network. For instance, in a network with five orderer nodes, an organization maintaining three of these nodes might compromise the network if it goes offline or decides to leave the network for good.

6.3. Node.js Application Architecture

This section provides a description of the Node.js application package. The application uses Visual Studio Code [63] as build tool and requires the following dependencies to be added:

- Docker 0.7.0 [64]
- Hyperledger Composer 0.19.12 [65]
- Node.js SDK for Hyperledger Fabric

The Docker extension is used to simplify the management of local Docker images and commands. Hyperledger composer plugin validate Composer model files that define the structure of our business network in terms of Assets, Participants and Transactions. The extension parses composer model (.cto) files and reports any validation errors. The Node.js SDK for Hyperledger Fabric is used for communication with components in the HLF Network package.

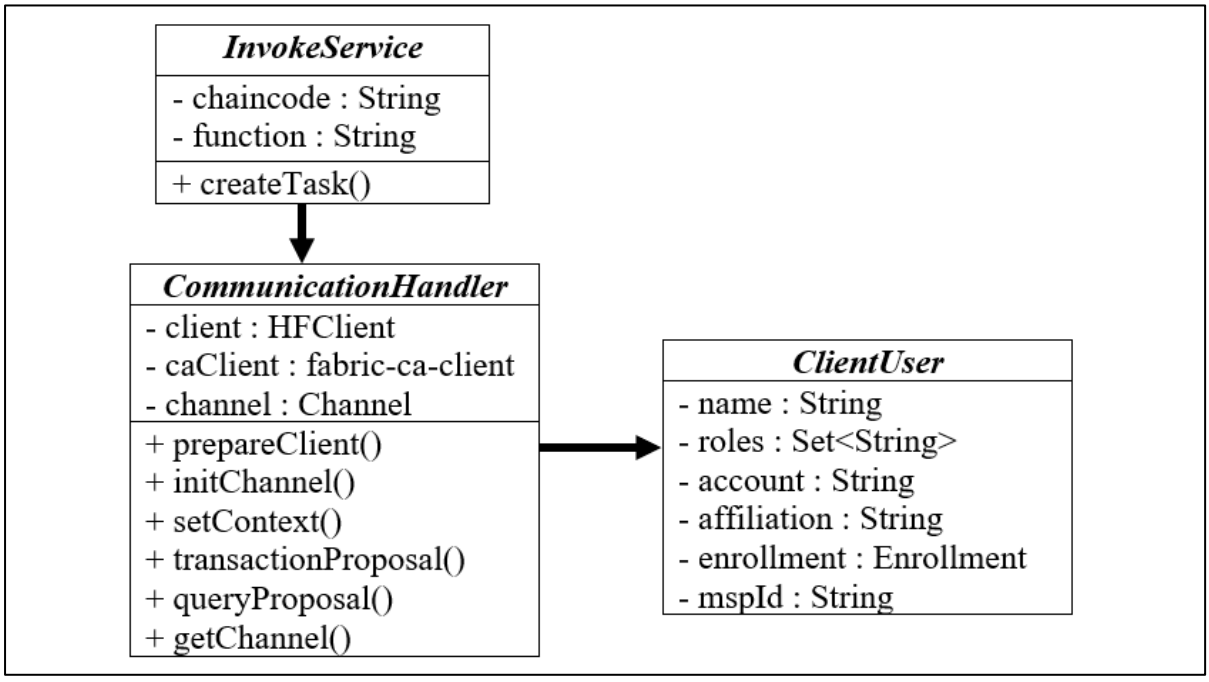

Figure 6.6: Class diagram of the classes executing the business logic in the application.

Figure 6.6 shows a class diagram of the classes executing the business logic in the application. GUI operations are controlled by the MainController class, while communication with the Fabric network is handled in the CommunicationHandler class. Utility and

intermediate classes are used to facilitate operations in these two classes. The main tasks of each class are further explained in the following subsections.

6.3.1. Hyperledger Fabric Integration

The Node.js Application package uses the Node.js SDK for Hyperledger Fabric for communication with the Fabric network in the HLF Network package. The SDK is included in the project files as a composer-artifacts folder. All dependencies used in the package are specified are present in this folder. Specific classes of the SDK are imported to classes in the application that require methods from the SDK.

Methods for communicating with the network are provided solely in the main.js class. Before communication methods can be executed, a fabric-client (Hyperledger Fabric Client) object, a fabric-ca-client (Hyperledger Fabric CA Client) and a Channel object must be initialized. These objects are created by running the prepareClient and initChannel methods provided in the class. Before initChannel is called, the application must enroll a user with the CA by calling the setContext method and provide the required enrollment details.

The User interface provided by the SDK is implemented in the separate classes for each user type. Objects of this class are intended to store the user details of a single user enrolled with the CA. ClientUser objects are stored to file so that the client can re-use the enrollment information even if the application has been restarted. The client can also re-enroll to a different user context without restarting the application. When re-enrolling, the previously enrolled user stored in a file.

The InvokeService class is called from the main class when a user requests communication with the Fabric network. The method provided in the service class runs in a separate thread, so that the GUI is still responsive to the user while it waits for communication with the network to finish. The class first checks whether the endorsing peers return the same results and thereafter waits for a transaction event to be broadcast on the channel, indicating if the transaction was accepted by the peers or not.

6.3.2. Endorser Selection

By default, the implementation of service discovery provided in the Node.js SDK provides two methods for endorsement selection: either Endorsement Selection Random or Endorsement Selection Least Required Block height. The former selects endorsing peers

randomly, as long as the peers comply with the chaincode endorsement policy, while the latter prefers endorsers with a smaller block height.

The Node.js application package implements its own method for endorsement selection named Endorsement Selection. The implementation selects peers from the layouts provided by the service discovery. A peer from the organization selected by the addRecord Mechanism chaincode and a peer from the organization invoking the transaction are selected for endorsement.

The layouts received from the service discovery are shuffled, so that if the same transaction is invoked twice, a new set of endorsing peers should be selected. A transaction would typically be invoked again if the first endorsement fails. This might happen if one of the endorsing peers does not manage to finish the endorsement process because of some internal error. Shuffling the layouts ensures that a different peer is likely selected for the next round of endorsement. Source code for Endorsement Selection is found in the Endorsement class, which implements the SDK's Endorsement class [13].

6.3.3. Graphic User Interface

AngularJS [66] is used for all elements comprising the GUI. Control and creation of GUI elements are separated from the business logic of the application and are executed in calls to the methods in the MainController class. Figure 6.7 shows the class diagram for the MainController class. Since the state database relies on JSON data representation, a JSON.parse() method is used for formatting the JSON data returned from state database queries to a humanly readable format.

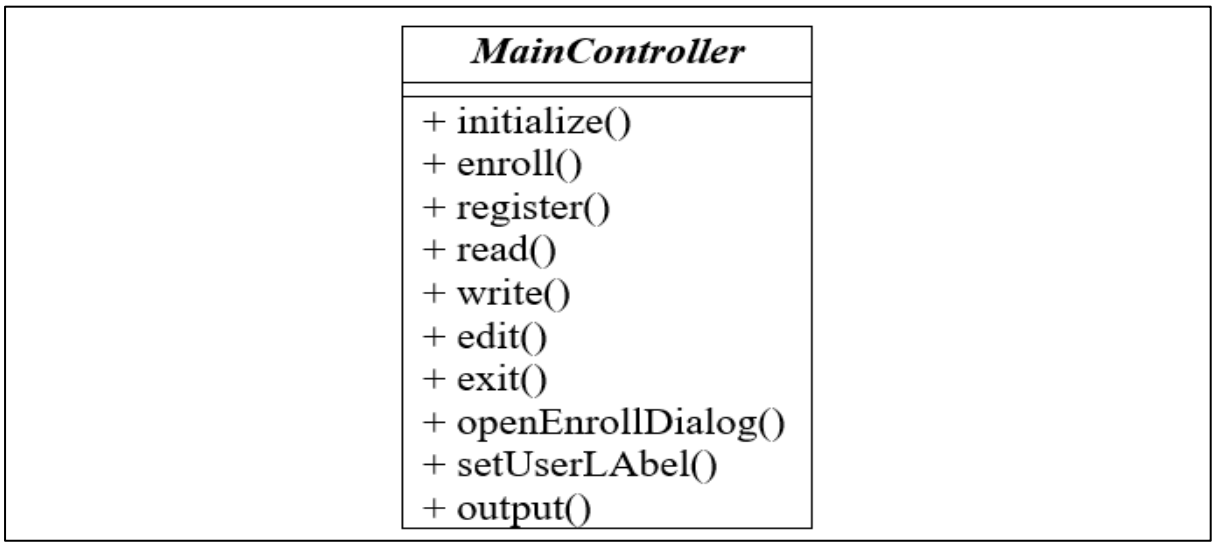

Figure 6.7: Methods provided by the MainController class.

6.4. Deploying the HLF Network Package

The HLF Network package can be deployed on any platform that satisfy the requirements listed in the prerequisites in the Hyperledger documentation [53]. A general description of how to run and configure the package is provided in this section, while a step-by-step user guide to set up and start the software is found in Appendix. For configuration details of the Fabric network that we do not touch upon in this thesis, we refer to the original documentation supplied in the Fabric GitHub repository [50] and on the Fabric Read the Docs webpage [51].

6.4.1. Initial Configuration of the Network

The HLF Network is comprised of several configuration files. This section provides a summary of how to configure the files and the network properly. The configuration files are explained in the order they are consumed by the setup scripts, which are also provided in the package. Further documentation on the semantics of the configuration files is found in the Fabric GitHub repository, as well as in the documentation on Docker Compose [57].

The Crypto Generator, which is introduced in section 3.2.1, generates the cryptographic material that is configured in the file named crypto-config.yaml. All peers and orderer nodes in the network must be listed in this file, so that corresponding cryptographic material can be created. Nodes must be listed under their associated organization. Hostname, common name and subject alternative names (SANs) must be specified according to the environment where nodes are deployed. This information is included in the certificates that are being generated. If a node operates from another address than the ones specified in this configuration, the certificate will be deemed invalid.

The next file that requires configuration is configtx.yaml. The path to the MSPs' cryptographic material must be specified in this file, along with at least one anchor peer for each organization. The ordering service used for the network must also be specified, along with hostnames, addresses and paths to cryptographic material for the orderer nodes in the ordering service. The network is governed by signature policies and implicit policies. Signature policies are used by MSPs to evaluate if signatures are valid, while implicit policies aggregate the results of signature policies in context of configuring the network. Both types of policies must be specified in configtx.yaml.

The remaining four configuration files are used for configuring the Docker containers and are consumed by the Docker Compose tool:

- compose-with-raft.yaml
- compose-with-couchdb.yaml
- base/compose-base.yaml
- base/peer-base.yaml

Each node in the network must be listed and configured as a volume and service in compose-with-raft.yaml. The same goes for CouchDB instances in the compose-with couchdb.yaml file. If an IP address is omitted from a container configuration, it will be assigned an IP address dynamically. This requires that an application contacting the container uses DNS for hostname to IP mapping.

Common configuration details for all peers and orderer nodes are provided in base/peerbase.yaml. Paths to cryptographic materials and configuration details for TLS and gossip protocols must be set in this file. Finally, base/compose-base.yaml provides individual environment configuration details such as addresses and endpoints for each node. These configuration details must match with the details provided in the cryptographic material for TLS handshakes to succeed.

6.4.2. Network Lifecycle Management

Three bash scripts for quick setup and break down of the network on a Linux installation are provided. It is advised to run the network on a fresh virtual machine to avoid other programs and custom configurations from affecting the network. See Appendix B for steps on how to successfully run the network and Appendix A for information on the software environment used during development.

Run the scripts in the order listed below to successfully create and thereafter break down a network using the HLF Network package:

- generate.sh - Generate network artifacts and cryptographic elements
- start.sh [seconds] - Create docker containers and request channel creation from an orderer node
- clean.sh - Stop and remove docker containers, and remove all generated artifacts and elements (Note: this script removes all Docker containers in the system)

The first script invokes the Configuration Transaction Generator and Crypto Generator tools. Use of the Crypto Generator is for testing purposes only and should be replaced by a CA in a production environment. For testing, the Crypto Generator offers an easy way to generate

the necessary cryptographic certificates and keys for each identity in the network before the network is brought up.

The next script starts by creating the necessary Docker containers. The required amount of time for initializing the containers may vary depending on the system's hardware configuration. If container services are invoked before they have been initialized, an error will occur. The start.sh script takes the number of seconds to wait for initialization as an argument when running the script. In general, no more than 20 seconds should be required for all containers to get ready for receiving communication requests. See section 7.3.2 for the average amount of time required for initialization during testing.

Docker containers offer a simple way to run several isolated entities on a single machine, communicating in the same manner as if they were on separate physical machines. To connect with entities running on other machines, the corresponding hostnames and IP addresses must be configured in the YAML configuration files.

The configuration files bundled with the HLF Network package are initially configured with three organizations representing each of the made-up providers named organization1, organization2 and organization3. The organizations are configured with two peers each, as well as a single orderer node per organization. New organizations and peers can be configured either in the configuration files before the network is started or by utilizing one of the Fabric SDKs while the network is already up and running. The Fabric binaries also provide some tools for adding organizations and peers using the command line.

Secure end-to-end communication is achieved with TLS. All entities in the network, except the CAs, are configured for communication over TLS. The default Fabric CAs have TLS disabled since they are provided for demonstration and testing purposes only. However, TLS can be enabled for CAs as well by adding the tls.enabled flag to the CA start command in compose-base.yaml, as well as the file paths to the CA's TLS certificate and key. If the default CA registrar name or password is changed, these details must also be updated in compose-base.yaml. Note that the CA certificates created by the Crypto Generator require the full hostname of the CA to be used in communication.

After the first two scripts have been executed, the network is up and ready for communication. Note, however, that a chaincode is not instantiated on a node until the first time the chaincode is called on that specific node. This instantiation process may cause transaction requests to timeout if it takes too long. The failed transaction request can then be

re-initiated later. This behavior ensures that system resources are not wasted on chaincode containers that are not in use, as redundant peers only install chaincode after the main peer is down and new peers start to receive chaincode invocations and endorsement requests.

The scripts used by generate.sh and start.sh to bring up the network are located in a directory named sample-setup:

- create-truststore.sh - Creates a trust store to be used with the Node.js Application package
- create-channel-request.sh - Sends a create channel request to the ordering service
- join-peers-to-channel.sh - Joins the listed peers to the channel
- define-anchor-peers.sh - Defines anchor peers for each organization
- instantiate-chaincode.sh - Installs chaincodes on the listed peers and instantiates chaincodes on the channel
- create-affiliations.sh - Adds the listed affiliations to the Fabric CAs

The files are invoked in the order listed above. When new peers are added to the configuration files, they must also be added to join-peers-to-channel.sh and instantiatechaincode.sh, if they are to join the channel and run chaincode. If new affiliations are required, they should be added to create-affiliations.sh or added manually by using the command line interface of the CA container.

6.5. Deploying the Node.js application Package

The source code for the Node.js application package is organized in main folder with subfolder for each user in the main directory. JavaScript classes are located in the subfolders found in home/usman/Medical-Blockchain/, and resource files are also located in user/usman/hp-app/resources. Classes are organized in the following packages:

- caliper - Classes used in communication with the Fabric network
- hp-app/web-app/patient-app - Classes for GUI elements and interaction
- util - Utility classes such as JSON manipulation and String constants

With the Node.js application package, the user can simulate database actions and thereafter query the blockchain to check if it updates correctly. The user can operate with different client identities from the same application instance, to see how access control is enforced.

6.5.1. Required Resource Files

The application requires icons to be placed on resources folders and endpoint.txt and a text file and a trust store to be placed in user/usman/Medical-Blockchain before the application is loaded.

The configuration files in the HLF Network package are configured with static IPs for peer and orderer nodes. This makes it possible to create cryptographic material with the Crypto Generator before the Docker containers are created. If not, cryptographic material must be created after each container has been assigned an IP dynamically.

If the hostname to IP address mapping is not available over DNS, the Node.js application must read hostnames and addresses from the system's hosts configuration file. In Linux operating systems, the hosts file is located in /Medical-Blockchain/endpoint. This file should be updated with the IP address of each Docker container and the associated hostname mapping. Hostnames for the Docker containers are specified in base/compose-base.yaml in the HLF network package. The mapping for the initial network configured in the HLF Network package is as follows:

172.18.0.40 peer0.organization1.example.com

172.18.0.50 peer1.organization1.example.com

172.18.0.60 peer0.organization2.example.com

172.18.0.70 peer1.organization2.example.com

172.18.0.80 peer0.organization3.example.com

172.18.0.90 peer1.organization3.example.com

172.18.0.100 orderer0.organization1.example.com

172.18.0.110 orderer0.organization2.example.com

172.18.0.120 orderer0.organization3.example.com

Make sure that the Docker containers are assigned IP addresses that are not already in use and that they are correctly included in the node's certificate.

6.5.2. User Interface Interaction

Our system provides a user interface from where all three users can navigate to their own separate user interfaces. Administration have open access to their part of system. They will use their interface to register other two users i.e. Healthcare-providers and Patients. Administrative staff will add user information and register them into the system. At successful

registration the system generates login credential of user such as ID, Patient-key and Private-key for patients' user and ID and Private-key for Healthcare-provider. Administration provide these login credentials to users. Now the user will use these login credential provided by the hospital administration to login to the system and become able to interact with the system. The frontend web application is written in HTML, CSS and JavaScript.

In case if a selected user is already stored in the user's directory, the application will read the ClientUser object from the file instead of trying to re-enroll the user with the CA. The enrollment dialog can also be opened from the main menu after the application has been started.

The application features a simple and intuitive user interface for all user types. Interface for hospital administration who is responsible for user registration and record maintenance is shown in Figure 6.8. The interface presents the administration user with two different tabs, each representing a different use case:

- Participants – This tab has further sub-tabs as dropdown: Healthcare-providers, Patients
  - Healthcare-providers – For management of Healthcare-providers register
  - Patients – For management of Patients register
- Transactions – Transactions tab have some test functions for developers. They can generate a new patient key by clicking on "Generate". The patient key will show underneath the button.

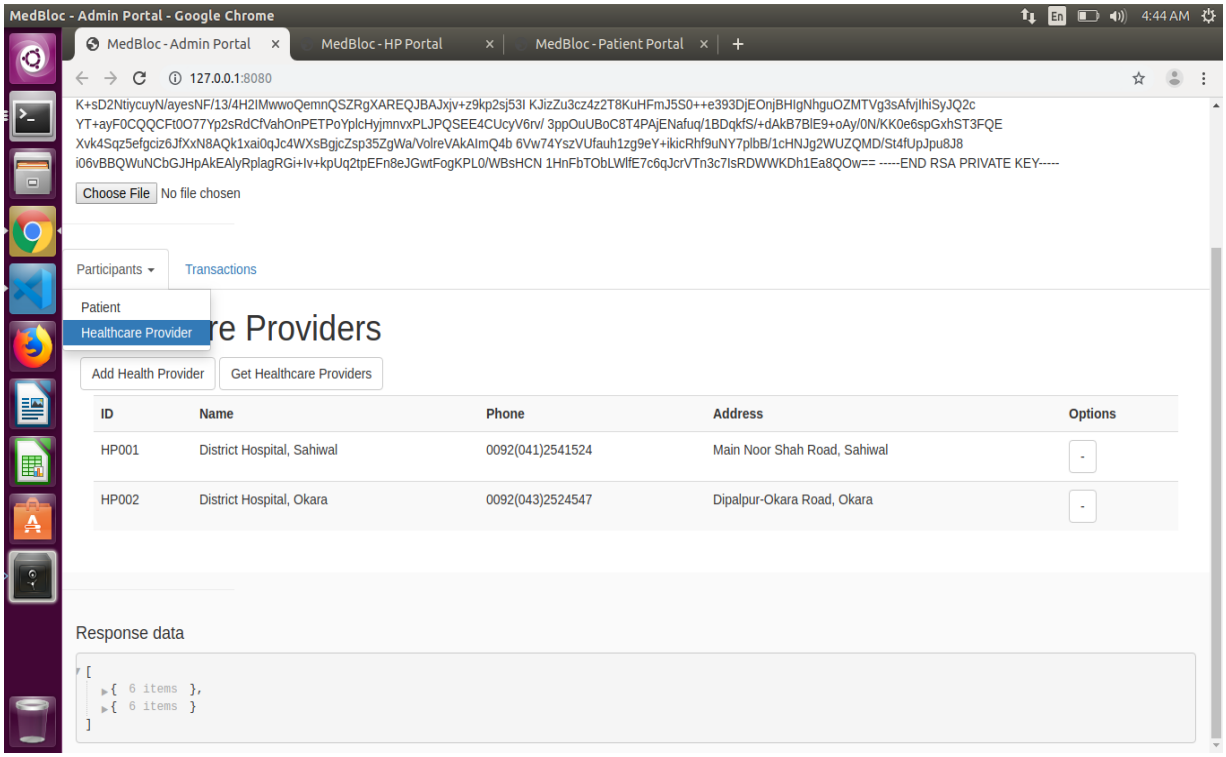

Figure 6.8: Administration portal user interface

When we click the Participants tab it displays dropdown menu with sub-tabs Healthcare-providers and Patients. By clicking on Healthcare-providers tab it navigates to Healthcare-providers register page where it displays all the registered Healthcare-providers of that organization. To register a new Healthcare-provider click on "Add Health Provider" button. It opens a modal where you will have to add user details to register a new user. Another button "Get Healthcare Provider" is also available to refresh Healthcare-providers list.

Healthcare-provider use its user interface to login to the system by providing its login credentials (Figure 6.9). A successful login navigates the user to his home page. This page has navigating bar to navigate to other pages. This includes following tabs:

- Home – To display home page
- Patients – To view list of patients
- Request sharing – To request consent from patients
- Patient keys – To view list of patients who have active consent
- Notifications – To show notification about key sharings

Home tab show user it's personal details. He can only view his personal details and cannot update any details. Patients tab show the list of patients registered with that hospital (organization). Every row in that list has "View Record" and "+" button to add new record. The Healthcare-provider can view and add new records of patient but only if he has the patient's consent for that. In Request sharing he Healthcare-provider can request patient-key from any patient by providing his/her ID. Patients keys show the list of patients who have shared their patient-key with that Healthcare-provider. User gets notification regarding key sharing in the notification tab.

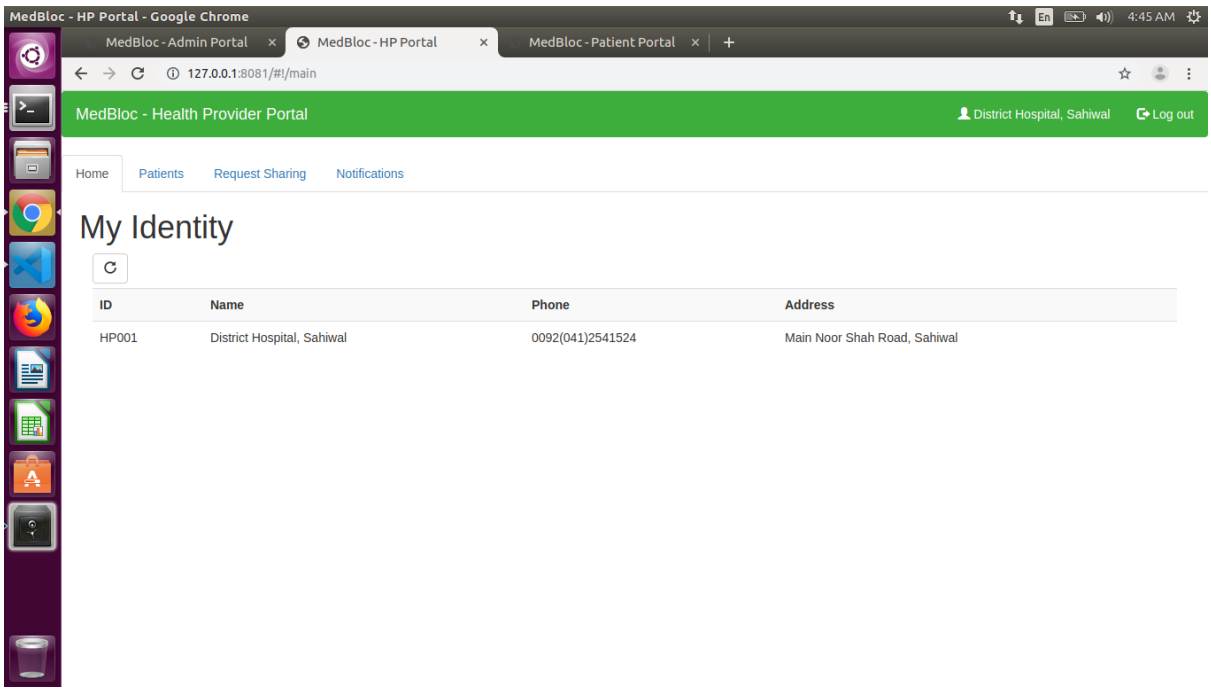

Figure 6.9: Health-Provider portal user interface

Patient web app's user interface is very simple and intuitive. Patient can access their profile page by logging in to the system by using their login credentials provided by the hospital administration. After successful login the user navigate to his profile home page. Patients user interface also has navigation bar for different uses which is shown in the Figure 6.10.

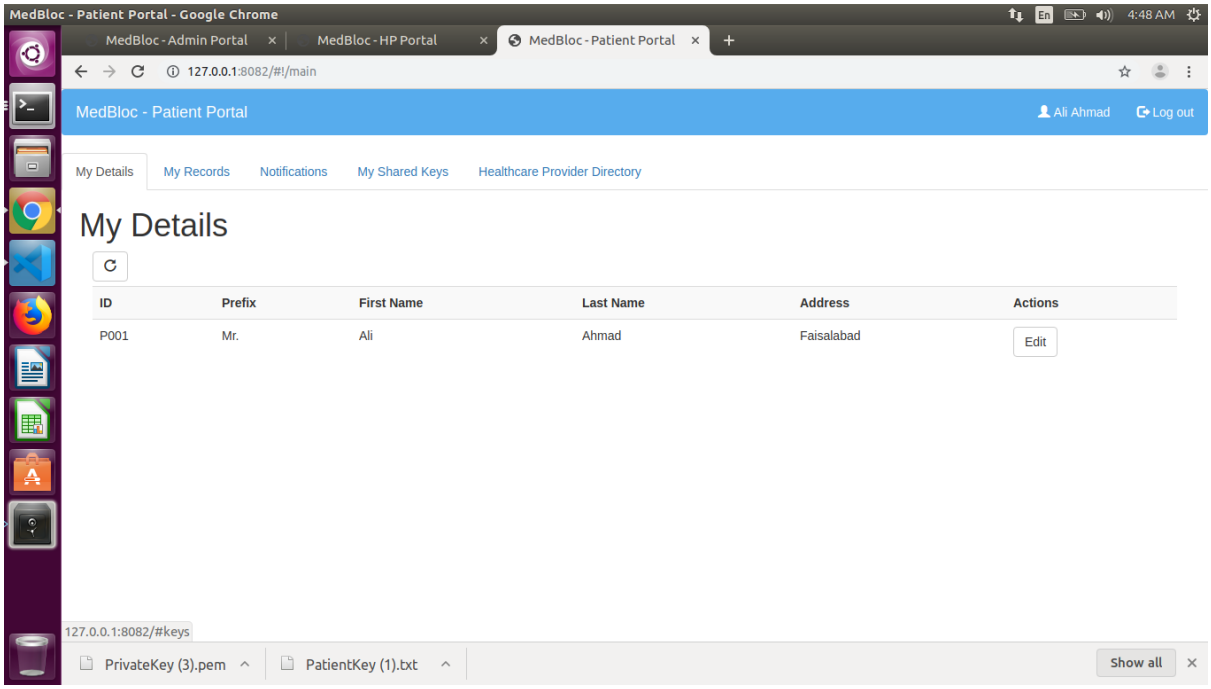

Figure 6.10: Patient portal user interface

Home displays the user with his/her personal details i.e. Name, Address etc. By clicking on "My Records" tab the user gets his medical records. This page has toggleable buttons on the left side to view specific types of records as shown in Figure 6.11. The next tab "Notifications" that displays notifications about key requests from Healthcare-providers. You can share your patient-key with that HP by clicking the "Share" button. When you receive a request from HP a red exclamation mark icon become displayed on the navigation bar which can be dismissed by clicking on it.

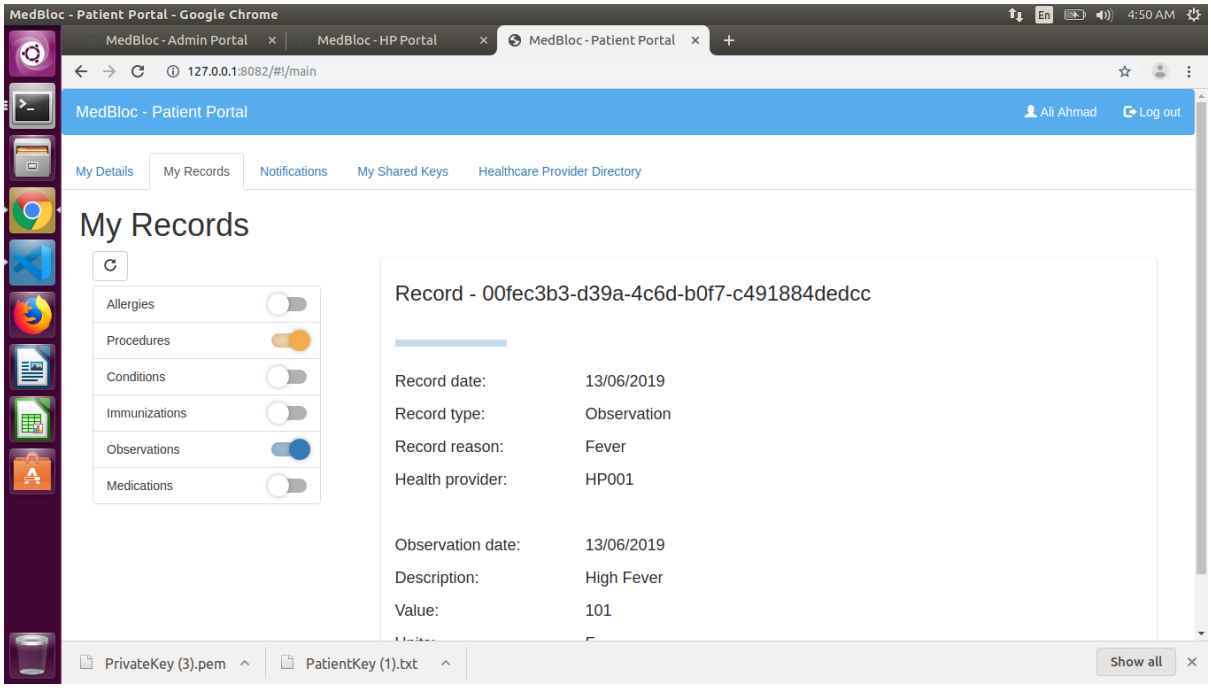

Figure 6.11: Patient's record display interface

Next is "My shared keys" that shows details of all HPs to whom show have given consent to view and add records as shown in Figure 6.12. The patient-key can also be shared manually with any HP by simply providing that HP's ID in the textbox and pressing "Share" button. To revoke the consent from the HP there is a "Revoke" button in the options column. After revoking the consent that HP will no longer be able to view are add your records. "Refresh" button is to reload the details. And the last one is "Health Providers Directory" that show the list of all doctors who are registered with that hospital.

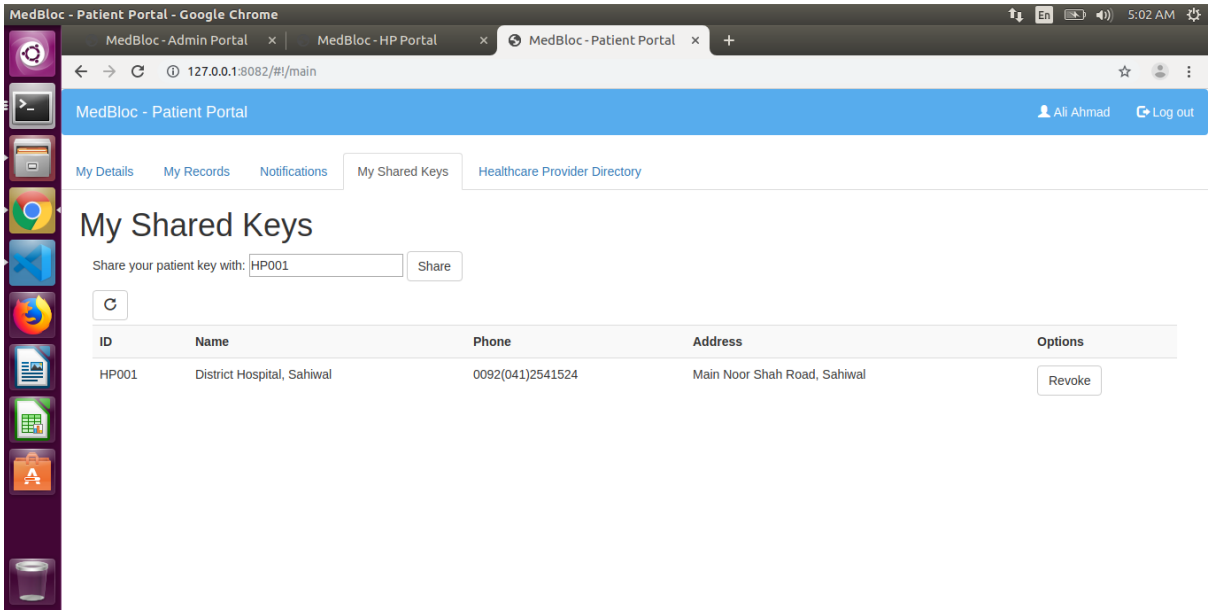

Figure 6.12: Patient's shared keys display

Chapter 7

Evaluation and Performance

CHAPTER 7: EVALUATION AND PERFORMANCE

7.1. Benefits Over Existing Healthcare System

The HLF network developed in this thesis brings several improvements in the areas of security, privacy and scalability over existing healthcare system and recently proposed blockchain-based EHR systems. The prototype was designed to explore and understand the potential of blockchain platform for the storage and sharing of electronic medical records, that ensure data security and privacy and solve the scalability issues in blockchain-based EHR systems. This system gives patients the ability to control and monitor their medical record by allowing them with a secure access and certain level of privacy. The permission management allows patients to share and exchange medical records with the healthcare provider and keep appending auditable-log of shared record, state transaction into the distributed ledger. Any transfer of values and records that changes the state of the blockchain is appended as a block in the distributed ledger and has relevance for both the patient and healthcare-provider and a patient can grant permission to healthcare providers to view their medical history and add new medical records. Patient can revoke this permission at any time he wants.

This system also solves scalability, data privacy and some authorization related issues in the permissionless blockchain-based healthcare data storage system that are recently proposed by some researchers. As these permissionless system allows anyone to join the blockchain network without any authorization and start participating in the consensus mechanism. Due to this permissionless structure this system possess some privacy and security issues. We have tried to solve these issues in our permissioned blockchain-based system.

In the following we have discussed security, privacy, scalability based on the finding from deployment and testing of the HLF network and compare with other related blockchain technologies discussed in Chapter 3 such as Ethereum.

7.1.1. Security

As a blockchain based distributed ledger, HLF network utilizes vital security models that help to mitigate the treats concerning safety and security related to patients' health records identified in the existing EHR systems. This blockchain-based system provides a secure distributed ledger where individual patients can keep their health records secure, accessible and if needed share with their healthcare-providers. The HLF network relays on PBFT consensus protocol and participating nodes to prevent a single point of failure.

To ensure confidentiality of data that medical records are encrypted with patient-key. Only the patient can allow healthcare provider to access to his/her medical records by sharing his/her patient-key with that healthcare-provider. Now after permission from the patient healthcare-provider can access patients medical records. These shared medical records data are hashed and signed with patient's secret key of that patient.

Only an authorized user can participate in the network making the system more secure compared to permissionless blockchain-based system. Hyperledger fabric membership service enroll users into the network using a certificate authority. Membership service defines the user enrollment rules, identities verification, authentication and access control. The Certificate Authority is a pluggable interface of the membership service, and it is responsible for assign certificate to user after verifying their identity.

In the recent past some researches have proposed and implemented blockchain-based EHR systems that uses permissionless blockchain technology. In this section we have looked into the attacks associated with permissionless blockchain and explained how our proposed framework can mitigate with such type of attacks and ensure more perfect security compared to some previously proposed blockchain-based EHR systems. The attacks include a range of mining attacks (such as the majority attack (51% attack)) and network attacks (such as DDoS and sybil). Testing the security of a blockchain network poses some challenges as the literature is still discussing the representative indicators and the testing tools are still in incubation. The literature approach towards security is generally based on modeling, trust assumption and theoretical assessments because the experimental validation is not always exhaustive.

7.1.1.1. The 51% Attack (Majority Attack):

The 51% attack which is also referred as the majority attack is renowned vulnerability in applications that uses blockchain technology. Such vulnerable application can be exploited when a mining pool or even a single attacker in the network attains the majority of the computing power and take full control of the blockchain. And after the attacker take control of the network by gaining the 51% computing power, the attacker will be to a) stop blocks or transactions verification b) prevent miner from mining any available block c) reverse transactions and d) can modify the transaction data, which may cause double-spending.

Recently proposed blockchain-based EHR systems maintain mutual trust in the network by using Proof of Work algorithm for consensus mechanism. In this algorithm the work done by the miners decides the probability of mining the block. This consensus algorithm is vulnerable to 51% attack also referred as the majority attack because any one can join the blockchain network and start mining the blocks and identities of these mining nodes are not known thus having possible malicious nodes presence.

As a Permissioned Blockchain based distributed ledger, our EHR framework utilizes vital security models that help to mitigate the treats concerning such attack. In our proposed system we have used Hyperledger based permissioned blockchain where we have set a network of trusted nodes who will participate the consensus mechanism. The combination of strict access control and network monitoring of any node in the network and the fact that the Fabric blockchain is permissioned means that the appearance of malicious nodes in the network is unlikely. So, the chances of 51% of attack is minimal in our proposed EHR framework.

7.1.1.2. Distributed Denial of Service:

A Distributed Denial of Service (DDoS) attack is the kind of network attack that disrupts the network by making the network resource unavailable to its users. Due to the distributed nature of blockchain ledger Distributed Denial of Service attack (DDoS) is a major security concern in blockchain based systems. For example, if an attacker wants to push large numbers of spam transactions to the blockchain network by compromising computational recourses, this would increase the transaction processing time and thus will create potentially a denial of service, as the peer nodes will be validating the incoming transactions.

Previously proposed permissionless public blockchain-based storage systems and some EHR systems, even though they are distributed peer-to-peer systems, they still prone to DDoS attacks. Blockchain based applications, such as Bitcoin and Ethereum, have frequently suffered from these kinds of attacks. Distributed denial of service attacks manifests themselves in different ways, depending upon the nature of application, architecture of the blockchain network, and nodes behavior. For example, in the Ethereum based network, the majority attack (51% attack) can lead to denial of service.

In our proposed permissioned blockchain based EHR system, it will be possible for the peers to agree to ignore or even block the user who is issuing such spam transactions. And as the nodes who are performing the consensus mechanism are trusted nodes and the fact that the Fabric blockchain is permissioned means that the appearance of malicious nodes in the network is unlikely and thus an attacker will not be able to control a majority of the consensus nodes and will not be able to disrupt the network.

7.1.1.3. Sybil Attack:

A Sybil attack occurs when a group of single or group of malicious entities manipulates the blockchain network by generating several false identities that are used to gain a substantial influence on the validation and the verification of the consensus process and effects the whole network. A single adversary may control multiple nodes on a network. It is not known to the blockchain network that these nodes are being controlled by the single adversarial entity. For example, an adversary can use multiple computers, IP addresses and virtual machines.

A sybil attack is possible in ethereum based permissionless blockchain where any one can join the network and their identifies are not known. A single adversary can control multiple nodes in in permissionless blockchain-based system. Sybil attacks are always possible in ethereum based blockchain system that use proof of work (PoW) consensus algorithm except under extreme and idealistic assumptions of computational resources. So, recently proposed ethereum based EHR systems which use Proof of Work (PoW) algorithm for consensus mechanism cannot prevent Sybil attack from occurring.

In our proposed system where we are using Hyperledger based permissioned blockchain for implementing our EHR system the possibility of sybil attacks ae minimum due to private permissioned nature of the blockchain network. Where we have used a combination of strict access control and network monitoring of any node in the network and nodes who are responsible for running the consensus mechanism are operating in full trusted environment. The identities of these nodes will be known to the system thus will perceive as lesser threats due to limited access to the blockchain network, trusted nodes and strict access control.

Table 7.1: Security comparison of proposed solution with previously proposed schemes

Type of Attack	Ethereum (Permissionless) based EHR Systems	Proposed Permissioned Blockchain based EHR System
Majority Attack (51% Attack)	Yes	No
Distributed Denial of Service attack (DDoS Attack)	Yes	No
Sybil Attack	Yes	No

7.1.2. Privacy

Ethereum based permissionless-blockchain systems have several limitations regarding privacy of data. The transfer of medical data within the public blockchain network is conducted at the ledger level and the activities are visible to all the participant nodes. The transfer of values and records within the distributed ledger are not entirely anonymous and become visible to all nodes participating in that permissionless network where any one can participate in the network. However, in our HLF network which is a permissioned-blockchain system these activities are performed at the transaction level and they are only visible to the authenticated nodes within the channel.

In this developed system patient's privacy is ensured by providing the patient with a possibility to specify fine-grained access control over his data via permissions. Permissions are enforced by chaincode logic and, therefore, cannot be violated by any user, unless the consensus protocols fail. The latter could happen only if a fraction of the verifying nodes intentionally tries to damage network operations. Centralized membership service already protects against Sybil attacks. Moreover, in the permissioned network, the nodes identities are known, therefore, there is no incentive for malicious behavior. In the case if a node still behaves maliciously, access to the network could be promptly restricted for this node.

Table 7.2: Privacy comparison of proposed solution with some previously proposed schemes

Solutions	Privacy
MedRec [23]	No
MeDShare [69]	No
Proposed Framework	Yes

Membership service also controls the identity of the users. Before registering a healthcare-provider the administration verifies that he/she is valid healthcare-provider. A patient is registered with a Unique ID, but all his data are linked to the pseudonym generated using his secret key. Therefore, Membership service does not have an access to the patient's clinical data, yet guarantees authenticity of the users (via digital signature verification).

7.1.3. Scalability

Scalability is the primary issue identified in previously proposed permissionless blockchain i.e. Ethereum based EHR solutions. As a permissionless and open blockchain, patients and healthcare-providers enroll into the network freely without an authentication and verification process. Patients and healthcare-providers who joins the system are participating in a consensus process and engages in state transactions by sending and receiving medical records and values. Hyperledger permissioned blockchain based EHR solution can efficiently solve this scalability problem.

Healthcare data sharing requires scalability of the system in terms of both the number of users and the number of nodes. The HLF network is more efficient handling large volume of data and more users. PBFT consensus protocol provides excellent scalability in terms of the number of users as well as in terms of the number of Nodes. Frequency of creating a block or number of transactions in a block (batch size) could be adjusted. It is necessary to check how the prototype behaves and how it performs to understand whether it can be of any use to easily manage health records while preserving the security.

To analyze the performance of the developed prototype and the underlying blockchain network was made using JMeter, an API testing tool maintained by the Apache Software Foundation and designed to load test and measure performance of web applications through

HTTP requests. The testing tool sends a batch of requests to the REST API implemented in Node.js. The API analyzes the request, checks the certificate using the HL Fabric SDK and then sends the transaction to the network through the SDK. However, this attempt fails to represent the real performance of the blockchain network because of the bottleneck caused by the REST API. To solve the problem, it was necessary to replace JMeter with a blockchain specific tool called Hyperledger Caliper (HL Caliper), which is the first to provide a means to test for different blockchain use cases, in fact, before HL Caliper, there was no general way to evaluate different blockchain implementations and projects.

7.1.3.1. Hyperledger Caliper:

HL Caliper is an open-source framework, under the Hyperledger umbrella and supported by various companies (IBM, Oracle, Huawei et al.), that provides users and developers with a tool to evaluate the performance of different blockchains. One of its declared purposes is to cope with the lack of source code of benchmark solutions that make it hard to both validate the results and to perform the same evaluation on different projects. It also attempts to cope with a lack of a common definition of performance indicators by relying on the definitions provided by the Performance & Scalability Working Group (PSWG).

In order to understand the validity of the results produced by HL Caliper, it is important to mention the modules that compose the benchmark tool. The architecture consists of three layers:

- **Adaptation Layer:** is used to integrate HL Caliper with different blockchain implementations and use cases. It contains a set of adaptors to interface with a particular use case.
- **Interface and Core Layer:** is formed by different modules that allow monitoring the resources, analyzing the performances, invoking and installing smart contracts on the blockchain, and generating the final report.
- **Application/Benchmark Layer:** contains scripts implemented by the user to test a particular blockchain use case. These scripts form the test suite that is used to measure the performances. This layer holds a benchmark engine that executes the tests using a master-workers strategy. The master node generates worker threads to which it assigns tasks. Each

task is in charge of one operation on the blockchain, that can be either smart contact initialization, execution or cleanup.

The produced final report contains different performance indicators:

- Success rate
- Transaction commit and state read Throughput (TPS)
- Transaction commit and read latency
- Resource consumption (CPU, Memory and network IO)

7.1.3.2. Test Environment and Results:

The tests were executed on a machine with the following characteristics:

• Operating system: Ubuntu 16.04

• HL Fabric components run in Docker container with the following resource allocation:

– 4 CPUs;

– 6GB of memory.

• The HL Fabric network setup was the following:

– HL Fabric version v1.4.2;

– The peers storage was the default LevelDB database;

– The ordering service based on Solo

Results:

Here the prototype presents the following characteristics:

3 organizations with 2 peers and one orderer each;

The transaction policy: at least 4 peers from different organizations have to endorse the transaction;

Write Operation:

Table 7.3: Average latencies of write operations

Send Rate (tps)	Average Latency (s)
5	0.32
10	0.52
15	0.54
20	0.77
30	5.31
40	9.35
50	10.13
100	14.37

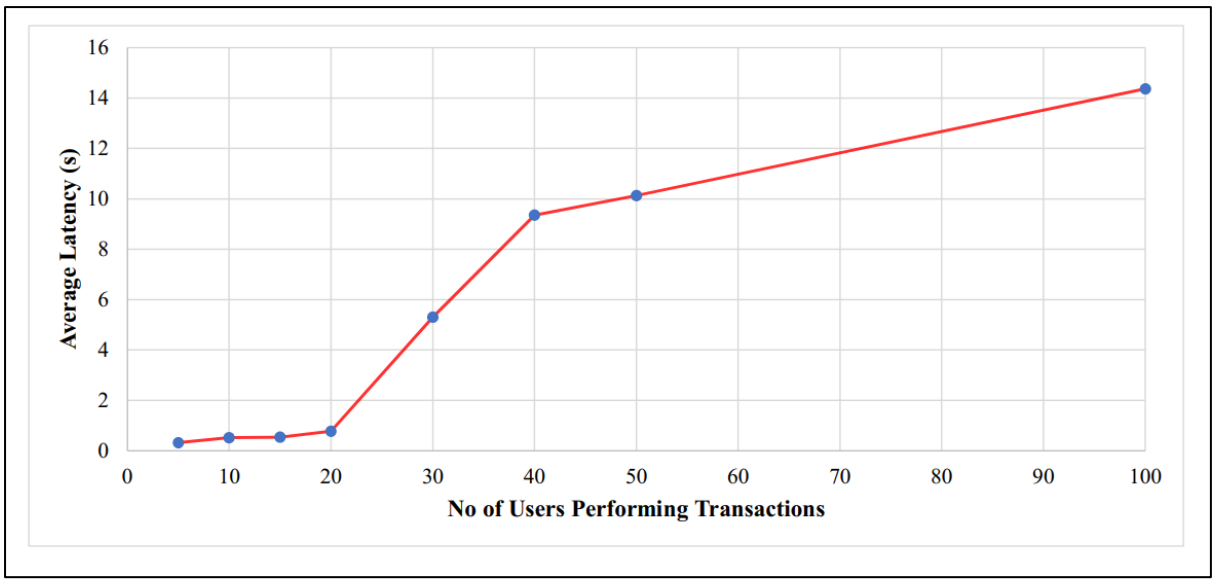

Figure 7.1: Average latencies of write operations

Table 7.4: Throughput of write operations

Transaction Per Second	Throughput (tps)
5	11.4
10	13.3
15	13.6
20	17.2
30	26.4
40	33.3
50	43.9
100	83.4

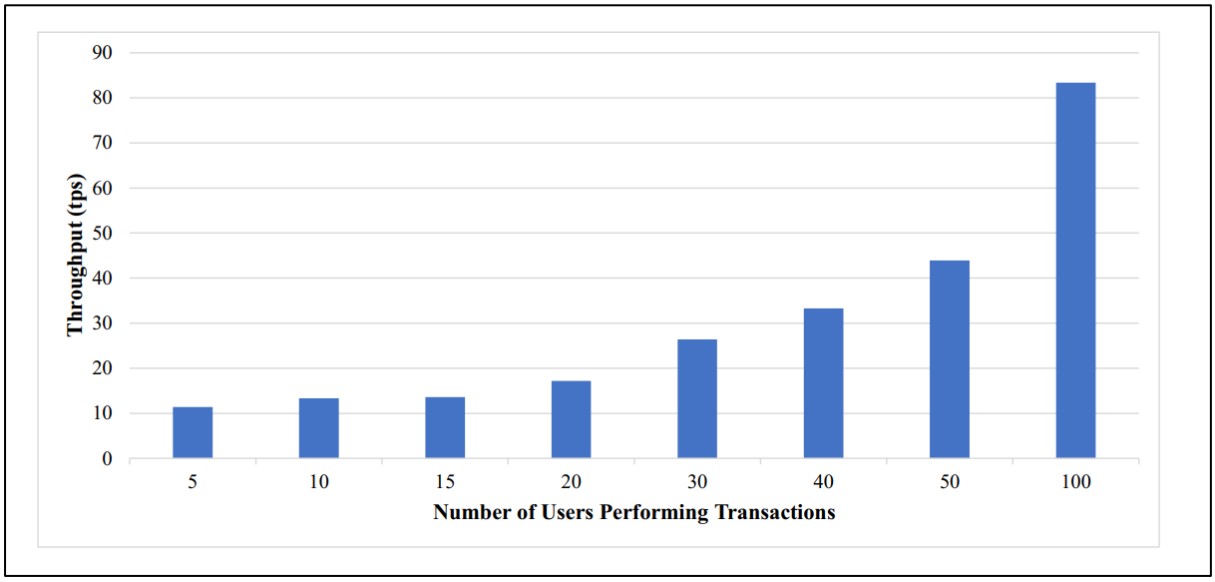

Figure 7.2: Throughput of write operations

Read Operation:

Table 7.5: Average latencies of read operations

No. of Users	Average Latency (s)
5	0.31
10	0.38
15	0.41
20	0.45
30	0.49
40	2.19
50	3.18
100	3.82

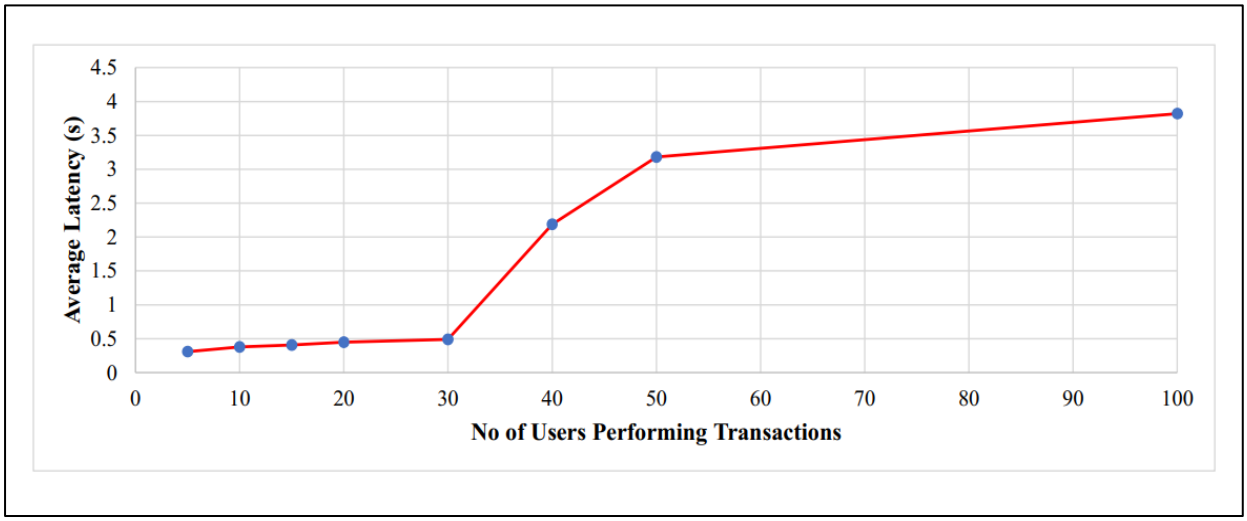

Figure 7.3: Average latencies of read operations

Table 7.6: Throughput of read operations

Transactions Per Second	Throughput (tps)
5	4.3
10	7.7
15	11.5
20	14.8
30	27.3
40	34.1
50	42.7
100	89

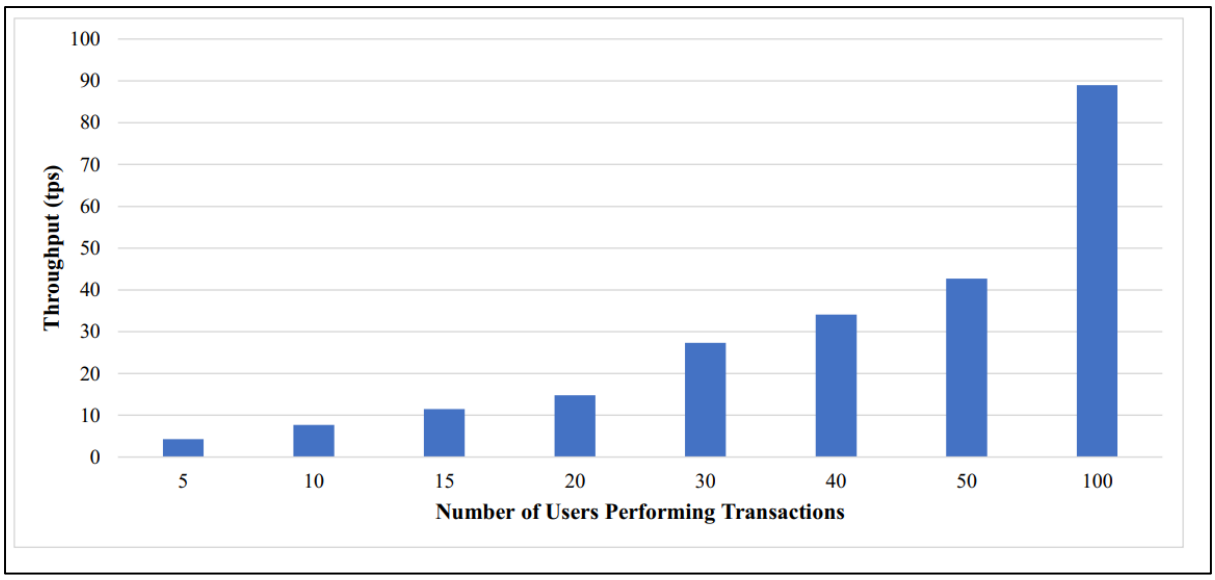

Figure 7.4: Throughput of read operations

Figure 7.5 below compares our EHR system with other existing blockchain based systems. The result shows that our proposed EHR framework is outstanding in scalability.

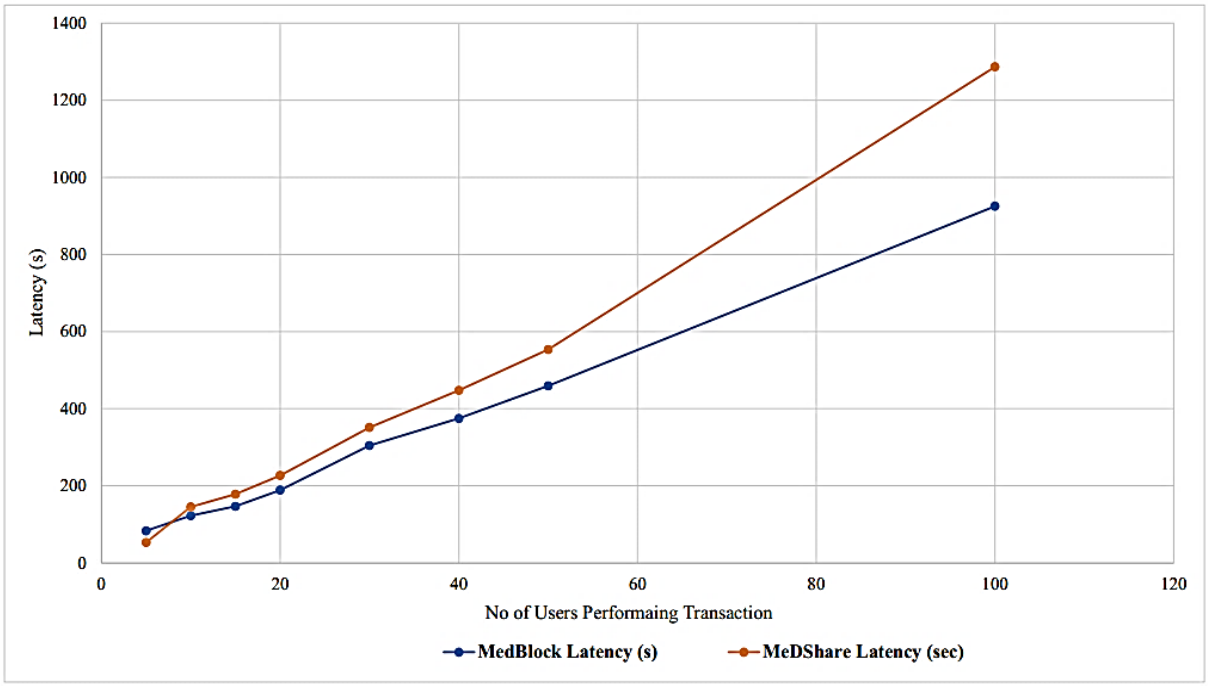

Figure 7.5: Average latencies of [MeDShare] [MedBlock].

7.2. Validating the System

The HLF Network package presented in Chapter 5 is a functional blockchain network. The integrity of the system has been verified during testing with the Node.js application package. Testing with the Node.js application package comprises the following tests:

- Add a new record to the blockchain
- Reading the record stored on the blockchain
- Giving permission to HP to access and add new record
- Revoking permission from HP
- Requesting patient to share key
- Registering users

No signs of data corruption or other malfunctions were discovered during testing with the JavaScript application.

7.2.1. Block Creation and Policies

The framework proposes a status field indicating if the record is successfully added to the blockchain to be included in the ledger. The embedded three-step transaction flow of Hyperledger Fabric ensures that a transaction that is not endorsed correctly is marked as invalid when peers place the block on the blockchain.

The three chaincodes developed for this thesis set an endorsement policy that requires four peers out of six peers (here we have set a network of three organization each having 2 peers and on orderer) to endorse a transaction. The peer within the organization invoking the transaction is used to verify that the invoking peer is not faulty, just as endorsers from other organizations also do. Meanwhile, the peers selected from the other organization ensures that no organization is trying to spoof the network.

The chosen endorsement policy is tolerant to misbehaving organizations, as long as two or more of the organizations do not perform a coordinated effort to spoof the network. If two coordinating organizations select each other's peers for endorsement, they can successfully propose and thereafter endorse any transaction they would like to. However, in a network composed of essentially trusted organizations, it is not expected that two organizations would operate in such a way.

It is possible to restrict peers of a specific role to endorse transactions on behalf of an organization. These roles are specified during channel configuration. Such roles could denote the various physical locations of peers or the physical location of the clients assigned with connecting to that specific peer.

7.2.2. Resilience to Fault and Misuse

The Raft ordering service is crash fault tolerant. New leaders are elected when the current leader node goes offline, e.g. with three nodes in the ordering service, the network can

tolerate to lose one node and still be operational with the two remaining nodes. For a five-node ordering service, the network can withstand the loss of two nodes. In other words, if the majority of orderer nodes are still active, the network can withstand to lose a node. The Kafka ordering service is also a crash fault tolerant service.

The Raft ordering service will serve as a starting point for the implementation of an official byzantine fault tolerant (BFT) ordering service for Hyperledger Fabric. Some unofficial BFT ordering services have been developed, but none are currently included in the official Fabric releases.

The lack of BFT in the current ordering services offered with Hyperledger Fabric means that the system does not sustain the robustness to handle malicious responses from compromised nodes in the network. However, for the closed healthcare system use case targeted with this thesis, the lack of BFT does not impose an immediate threat to the system. The combination of strict access control and network monitoring of any node in the network of a healthcare provider and the fact that the Fabric blockchain is permissioned means that the appearance of malicious nodes in the network is unlikely.

7.3. Performance at Scale

In versions prior to v1.1.0, Hyperledger Fabric voting-based consensus performed worse when scaling for an increased number of nodes than comparable blockchain implementations. However, for versions after v1.1.0 there are no indications that the framework has problems with scaling [67].

Performance in the Fabric blockchain has two potential bottlenecks:

- Peer node endorsements
- Ordering service throughput

Our chosen endorsement policy of requiring only one external node to endorse our transaction ensures that the endorsements bottleneck is minimized. The peer node endorsements bottleneck would only increase if we add too many clients compared to peer nodes in the network. If this happens, we must introduce additional peer nodes to handle the endorsements bottleneck.

As the ordering service is a queuing system that batches transactions into blocks, increasing the batch size might help for throughput issues. As each query of the blockchain will result in a log transaction, we might end up with a large amount of transactions in the

system as the number of clients increase. However, the workload involved in creating a block is small, as no validation of the data is performed at the ordering service. The computationally intensive tasks are solely executed at peer nodes. The potential bottleneck imposed by the ordering service should in most cases be negligible.

Testing and measurements obtained in regard to the implementation created for this thesis are limited by physical constraints in the testing environment. The testing simulates virtual nodes with the use of containers running on a single physical machine. Each container presents an isolated environment where software can operate as if it was running on a separate physical machine and communicate with other containers through loopback network interface.

7.3.1. Increasing Peer to Orderer Ratio

The configuration files supplied with the HLF Network package are initially configured with two peer nodes and a single orderer node for each of three organization. This gives a 2:1 peer to order ratio, which for larger networks means that we will end up with way too many orderer nodes in the network. For larger networks, a 10:1 peer to orderer ratio would be a more reasonable configuration.

No more than two or three orderer nodes provided by each organization are required. The exact number of orderer nodes per organization will depend on the number of peers per organization in the network. In a network with a large number of peers but few organizations, each organization will have to provide more orderers than in a network with fewer peers and a higher number of organizations. In general, a significantly higher number of endorsing peers than orderers are needed in the network.

The final decision on how many peer nodes and orderer nodes are required in the network will always depend on the expected connectivity of the nodes in the network. If nodes in the network have low connectivity and regularly experience connection issues, more redundant nodes must be added to the network.

The number of nodes feasible for testing is bounded by the constraints induced by the machine running the containers. For a realistic full-scale test, nodes should be placed in physical disparate locations as to simulate organizational setup, administration and overhead.

7.3.2. Raft vs Kafka Ordering Service

The initial version of the HLF Network package used a Kafka ordering service in Hyperledger Fabric v1.4.0. As of v1.4.1, the new Raft ordering service was introduced as an

option. The final version of the HLF Network package will use the Raft ordering service. In terms of operational and administrative complexity, using the Raft service over Kafka reduces the complexity significantly, as discussed in section 3.3.3. This is especially noteworthy for large systems spanning multiple organizations. The Raft service requires significantly less inter-node ordering communication and system overhead.

Another drawback of the Kafka implementation is that the Kafka cluster must be run as a single organization in the network. This means that all orderer nodes will communicate with the same centralized cluster. This also introduce implications when scaling the network to support a large number of nodes.

Table 7.7: The average required setup times for the Kafka and Raft ordering services in the development environment.

# of Nodes	Setup time for Kafka	Setup time for Kafka
2	44 seconds	9.2 seconds
3	53 seconds	11.7 seconds
4	65 seconds	12.2 seconds

For a system running on the setup described in Appendix A, the initial time for all specified nodes to be up and ready to accept communication is reduced from 1-2 minutes to only 10-15 seconds when applying the Raft ordering service over Kafka. This means that the Raft setup is about six times quicker than the Kafka setup. This is bound to the fact that there is no need for any Kafka or ZooKeeper nodes to be initialized. The average time measured for the Docker Compose setup to finish for the environment described in Appendix A is shown in Table 7.7. The result is an average of three attempts for each configuration.

Chapter 8

Conclusion

CHAPTER 8: CONCLUSION

8.1. Concluding Statement

The thesis introduces a cost effective and adaptable blockchain implementation for improving security, privacy and scalability of EHR systems. The proposed blockchain-based EHR framework has been successfully implemented in Hyperledger Fabric. The implementation is verified to work according to the description provided in this thesis and it can be concluded that the framework can draw benefits from the implementation in Hyperledger Fabric.

In the thesis we have described chaincode implementations for the proposed smart contracts. The implementation has been tailored to the three-step transaction flow embedded by Hyperledger Fabric, which means that instead of selecting an orderer to create the block as would typically be done in a traditional transaction flow, the mechanism selects peers for the more computational heavy task of endorsing a transaction.

The proposed chaincode implementations place logs of events happening to an EHR on the blockchain, which make for immutable records capabilities. Chaincodes also embed access control features directly on the blockchain. The access control lists (ACLs) governing which entities are authorized to access a record is placed on the blockchain to avoid malicious edits to an ACL, while the process of authenticating a client for access to a Record is conducted in chaincode checking for entries in the ACLs. This makes for strict enforcement of access control policies, improving privacy of the EHR system.

The Hyperledger Fabric blockchain implementation does, however, impose some constraints on the framework. One of the imposed constraints is that every transaction must be deterministic, which means we cannot generate non-deterministic values within chaincodes, and that a transaction must be executed at every peer that validates it, which imposes some computational load. A discussion of potential future improvements of the two proposed software packages are discussed in the following section.

8.2. Further Development

The HLF Network package currently comprises the features described in the proposed EHR framework. The framework introduces new features of encrypting the Record log and using a collective authority for distributing the secret key. Currently the there is no secure

mechanisms to share keys with user that they use for login to the system. Hyperledger Fabric already provides support for encrypting objects and it should be a feasible task to implement a secure mechanism for key distribution as well. Implementing these features should be the next step in any further development of the package.

As mentioned in section 3.3.3, a new Byzantine fault tolerant ordering service for Hyperledger Fabric is in development. As this new service is being built on top of the Raft ordering service, which is already used in the HLF Network package, it is likely that changing to the new service will only be a minor task.

References

[1] Office of the National Coordinator for Health Information Technology, “What is an electronic health record (EHR)?.” https://www.healthit.gov/faq/ what-electronic-health-record-ehr. Retrieved April 25, 2019.

[2] J. L. Fernández-Alemán, I. C. Señor, P. n. O. Lozoya, and A. Toval, “Security and privacy in electronic health records: A systematic literature review,” Journal of Biomedical Informatics, vol. 46, no. 3, pp. 541–562, 2013.

[3] Roberts, Lucien W. and Towey, Emily W. G., “Risk management: Medical records.” https://www.physicianspractice.com/pearls/risk-management-medical-records. Retrieved March 1, 2019.

[4] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system.” https://bitcoin.org/bitcoin.pdf. Retrieved March 25, 2019.

[5] Peterson, K., Deeduvanu, R., Kanjamala, P., & Boles, K. (2016b). A Blockchain-Based Approach to Health Information Exchange Networks. Mayo Clinic.

[6] Commission, E. (2012). eHealth Action Plan 2012-2020. Innovative healthcare for the 21st century. https://doi.org/SWD(2013) 527

[7] Prakash, R. (2016). Adoption of block-chain to enable the scalability and adoption of Accountable Care. NIST Workshop on Blockchain & Healthcare, (August).

[8] Krawiec, R., Housman, D., White, M., Filipova, M., Quarre, F., Barr, D., … Tsai, L. (2016). Blockchain: Opportunities for Health Care.

[9] Gartner. (2016). Hype Cycle for Emerging Technologies. Retrieved from https://www.gartner.com/newsroom/id/2575515

[10] Gartner, I. (2017). Top Trends in the Gartner Hype Cycle for Emerging Technologies, 2017 - Smarter with Gartner. Retrieved from https://www.gartner.com/smarterwithgartner/top-trends-in-the-gartner-hype-cyclefor-emerging-technologies-2017/

[11] Ethereum Community, “White paper.” https://github.com/ethereum/wiki/wiki/White-Paper. Retrieved May 20, 2019.

[12] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukoli´c, S. W. Cocco, and J. Yellick, “Hyperledger fabric: A distributed operating system for permissioned blockchains,” in Proceedings of the Thirteenth EuroSys Conference, EuroSys ’18, (New York, NY, USA), pp. 30:1–30:15, ACM, 2018.

[13] Hyperledger, “Node.js SDK for Hyperledger fabric 1.4.” https://fabric-sdk-node.github.io/release-1.4/index.html. Retrieved May 4, 2019.

[14] Tapscott, D., & Tapscott, A. (2016). Blockchain Revolution. Blockchain Revolution. https://doi.org/10.1515/ngs-2017-0002

[15] Xu, X., Pautasso, C., Zhu, L., Gramoli, V., Ponomarev, A., Tran, A. B., & Chen, S. (2016). The blockchain as a software connector. Proceedings - 2016 13th Working IEEE/IFIP Conference on Software Architecture, WICSA 2016, (August), 182–191. https://doi.org/10.1109/WICSA.2016.21

[16] TechTarget. What is DDBMS (distributed database management system)? - Definition from WhatIs.com. 2005. url: http://searchsqlserver.techtarget.com/definition/DDBMS (visited on 01/15/2018)

[17] T. T. Kuo, H. E. Kim, and L. Ohno-Machado. "Blockchain distributed ledger technologies for biomedical and health care applications." In: Journal of the American Medical Informatics Association 24.6 (2017), pp. 1211–1220. issn: 1527974X. doi: 10.1093/jamia/ocx068.

[18] R. Krawiec, D. Barr, J. Killmeyer, M. Filipova, F. Quarre, A. Nesbitt, K. Fedosova, L. Tsai, and A. Israel. "Blockchain: Opportunities for Health Care." In: NIST Workshop on Blockchain & Healthcare August (2016), pp. 1–12.

[19] L. A. Linn and M. B. Koo. "Blockchain For Health Data and Its Potential Use in Health IT and Health Care Related Research." In: U.S. Department of Health and Human Services (2016), pp. 1–10. url: https://www.healthit.gov/sites/ default/files/11-74-ablockchainforhealthcare.pdf.

[20] T. Mcconaghy, R. Marques, A. Müller, D. De Jonghe, T. Mcconaghy, G. Mcmullen, R. Henderson, S. Bellemare, and A. Granzotto. "BigchainDB: A Scalable Blockchain Database (DRAFT)." In: BigchainDB (2016), pp. 1–65. url: https: //www.bigchaindb.com/whitepaper/bigchaindb-whitepaper.pdf.

[21] IBM Global Business Services Public Sector Team. "Blockchain: The Chain of Trust and its Potential to Transform Healthcare – Our Point of View." In: NIST Workshop on Blockchain & Healthcare (2016). url: https://bit.ly/2oBJDLw.

[22] M. Rouse. What is data lake? 2015. url: http://searchaws.techtarget.com/definition/data-lake (visited on 01/19/2018).

[23] A. Ekblaw, A. Azaria, J. D. Halamka, A. Lippman, I. Original, and T. Vieira. "A Case Study for Blockchain in Healthcare: "MedRec "prototype for electronic health records and medical research data MedRec: Using Blockchain for Medical Data Access and Permission Management." In: IEEE Technology and Society Magazine (2016), pp. 1–13. doi: 10.1109/OBD.2016.11. url: https://bit.ly/2mrsp5O.

[24] Alexaki, S., et al. Blockchain-based Electronic Patient Records for Regulated Circular Healthcare Jurisdictions. in 2018 IEEE 23rd International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD). 2018.

[25] Li, H., et al., Blockchain-Based Data Preservation System for Medical Data %J J. Med. Syst. 2018. 42(8): p. 1-13.
[26] T. F. Xue, F.U.Q., C. Wang, and X. Y. Wang,. A Medical Data Sharing Model via Blockchain. in Acta Automatica Sinica. 2017.
[27] H. Oh, C. Rizo, M. Enkin, and A. Jadad, "What is ehealth (3): A systematic review of published definitions," J Med Internet Res, 2005.
[28] HealthIT.gov, "What is an electronic health record (EHR)?." https://www.healthit.gov/faq/what-electronic-health-record-ehr. Retrieved May 25, 2019.
[29] J. L. Fernández-Alemán, I. C. Señor, P. n. O. Lozoya, and A. Toval, "Security and privacy in electronic health records: A systematic literature review," Journal of Biomedical Informatics, vol. 46, no. 3, pp. 541–562, 2013.
[30] L. Beard, R. Schein, D. Morra, K. Wilson, and J. Keelan, "The challenges in making electronic health records accessible to patients," Journal of the American Medical Informatics Association, vol. 19, no. 1, pp. 116–120, 2012.
[31] C. Paar and J. Pelzl, Understanding Cryptography: A Textbook for Students and Practitioners. Springer, 2nd ed., 2010.
[32] Yaga, Dylan and Mell, Peter and Roby, Nik and Scarfone, Karen, "Blockchain technology overview." https://doi.org/10.6028/NIST.IR.8202. Retrieved May 30, 2019.
[33] M. Castro and B. Liskov, "Practical byzantine fault tolerance," in OSDI '99 Proceedings of the Third Symposium on Operating Systems Design and Implementation, pp. 173– 186, USENIX Association Berkeley, 1999.
[34] Nxt community, "Nxt whitepaper." https://www.dropbox.com/s/cbuwrorf672c0yy/NxtWhitepaper_v122_rev4.pdf. Retrieved March 25, 2019.
[35] Hyperledger, "Hyperledger fabric SDKs." https://hyperledger-fabric.readthedocs.io/en/release-1.4/fabric-sdks.html. Retrieved May 4, 2019.
[36] Hyperledger, "Glossary." https://hyperledger-fabric.readthedocs.io/en/release-1.4/glossary.html. Retrieved May 7, 2019.
[37] The Linux Foundation, "Hyperledger." https://www.hyperledger.org. Retrieved March 25, 2019.
[38] Docker Inc., "What is a container? a standardized unit of software." https://www.docker.com/resources/what-container. Retrieved May 1, 2019.
[39] K. Olson, M. Bowman, J. Mitchell, S. Amundson, D. Middleton, and C. Montgomery, "Sawtooth: An introduction." https://www.hyperledger.org/wp-content/uploads/2018/01/Hyperledger_Sawtooth_WhitePaper.pdf. Retrieved May 25, 2019.
[40] The Linux Foundation, "About hyperledger." https://www.hyperledger.org/about. Retrieved March 1, 2019.

[41] Hyperledger, “Chaincode for operators.” https://hyperledger-fabric.readthedocs.io/en/release-1.4/chaincode4noah.html. Retrieved May 15, 2019.
[42] Hyperledger, “Gossip data dissemination protocol.” https://hyperledger-fabric.readthedocs.io/en/release-1.4/gossip.html. Retrieved May 7, 2019.
[43] Hyperledger, “The ordering service.” https://hyperledger-fabric.readthedocs.io/en/release-1.4/orderer/ordering_service.html. Retrieved May 25, 2019.
[44] Apache Software Foundation, “Introduction.” https://kafka.apache.org/intro.html. Retrieved April 7, 2019.
[45] Apache Software Foundation, “Apache zookeeper.” https://zookeeper.apache.org. Retrieved April 7, 2019.
[46] D. Ongaro and J. Ousterhout, “In search of an understandable consensus algorithm,” in 2014 USENIX Annual Technical Conference, pp. 305–319, USENIX Association, 2014.
[47] Hyperledger, “Ledger.” https://hyperledger-fabric.readthedocs.io/en/release-1.4/ledger.html. Retrieved May 25, 2019.
[48] I. MongoDB, “NoSQL databases explained.” https://www.mongodb.com/nosql-explained. Retrieved May 6, 2019.
[49] Ecma International, “The JSON data interchange syntax.” https://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf. Retrieved May 30, 2019.
[50] Hyperledger, “v1.4.1 release notes - april 11, 2019.” https://github.com/hyperledger/fabric/releases/tag/v1.4.1. Retrieved April 25, 2019.
[51] Hyperledger, “A blockchain platform for the enterprise.” https://hyperledger-fabric.readthedocs.io/en/release-1.4/. Retrieved May 1, 2019.
[52] Hyperledger, “Activity.” https://jira.hyperledger.org/projects/FAB/summary. Retrieved May 1, 2019.
[53] Hyperledger, “Prerequisites.” https://hyperledger-fabric.readthedocs.io/en/release-1.4/prereqs.html. Retrieved May 1, 2019.
[54] Hyperledger, “cryptogen.” https://hyperledger-fabric.readthedocs.io/en/release-1.4/commands/cryptogen.html. Retrieved May 6, 2019.
[55] Ben-Kiki, Oren and Evans, Chris and döt Net, Ingy, “Yaml ain’t markup language (YAML™) version 1.2.” https://yaml.org/spec/1.2/spec.pdf. Retrieved March 1, 2019.
[56] Hyperledger, “configtxgen.” https://hyperledger-fabric.readthedocs.io/en/release-1.4/commands/configtxgen.html. Retrieved May 6, 2019.
[57] Docker Inc., “Overview of docker compose.” https://docs.docker.com/compose/overview/. Retrieved May 1, 2019.
[58] Hyperledger, “Fabric CA user’s guide.” https://hyperledger-fabric-ca.readthedocs.io/en/release-1.4/users-guide.html. Retrieved May 10, 2019.

[59] Hyperledger, “Service discovery.” https://hyperledger-fabric.readthedocs.io/en/release-1.4/discovery-overview.html. Retrieved May 7, 2019.
[60] Apache Software Foundation, “Apache license, version 2.0.” https://www.apache.org/licenses/LICENSE-2.0.html. Retrieved May 7, 2019.
[61] Rescorla, E., “The transport layer security (TLS) protocol version 1.3.” https://tools.ietf.org/pdf/rfc8446.pdf. Retrieved May 30, 2019.
[62] w3school, “JavaScript” https://www.w3schools.com/jsref/jsref_parse_json.asp Retrieved May 1, 2019.
[63] Code, V. (2019). Visual "Studio Code." https://code.visualstudio.com/ Retrieved April 16 2019.
[64] Docker 0.7.0, “Docker for VS Code.” https://code.visualstudio.com/docs/azure/docker. Retrieved May 5, 2019.
[65] Composer 0.19.20, “composer-vscode-plugin.” https://github.com/hyperledger/composer-vscode-plugin/. Retrieved May 4, 2019.
[66] Angularjs.org., "AngularJS — Superheroic JavaScript MVW Framework." https://angularjs.org/. Retrieved May 4 2019.
[67] C. Ferris, “Does hyperledger fabric perform at scale?.” https://www.ibm.com/blogs/blockchain/2019/04/does-hyperledger-fabric-perform-at-scale/. Retrieved April 15, 2019.
[68] Hyperledger.github.io. (2019). Installing | Hyperledger Composer. https://hyperledger.github.io/composer/latest/installing/installing-index.html Retrieved April 16 2019.
[69] Q Xia, E B Sifah, K O Asamoah, J Gao, X J Du, and M Guizani,, “MeDShare: trust-less medical data sharing among cloud service providers via blockchain,” IEEE Access, vol.5, 2017, pp.14757-14767.
[70] D. Vujičić, D. Jagodić and S. Ranđić, "Blockchain technology, bitcoin, and Ethereum: A brief overview," 2018 17th International Symposium INFOTEH-JAHORINA (INFOTEH), East Sarajevo, 2018, pp. 1-6. doi: 10.1109/INFOTEH.2018.8345547
[71] Kuo, T., Kim, H., & Ohno-Machado, L. (2017). Blockchain distributed ledger technologies for biomedical and health care applications. Journal Of The American Medical Informatics Association, 24(6), 1211-1220. doi: 10.1093/jamia/ocx068
[72] Z. Shae and J. J. P. Tsai, "On the Design of a Blockchain Platform for Clinical Trial and Precision Medicine," 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), Atlanta, GA, 2017, pp. 1972-1980.
[73] Michael Dufel. A new paradigm for health information exchange. Technical report, Peer Health, 2016. http://peerhealth.io/download-whitepaper/.

[74] Kefa Rabah. Challenges & opportunities for blockchain powered healthcare systems: A review. Mara Research Journal of Medicine & Health Sciences-ISSN 2523-5680, 1(1):45–52, 2017.

[75] Jesse Damiani. Simplyvital health is using blockchain to revolutionize healthcare, 2017. https://www.forbes.com/sites/jessedamiani/2017/11/06/simplyvital-health-blockchain-revolutionize-healthcare/#f4a1fbd880a0

[76] M. G. Kim, A. R. Lee, H. J. Kwon, J. W. Kim, and I. K. Kim, "Sharing Medical Questionnaries based on Blockchain," Proc. - 2018 IEEE Int. Conf. Bioinforma. Biomed. BIBM 2018, pp. 2767–2769, 2019.

[77] Fan, K., Wang, S., Ren, Y., Li, H., & Yang, Y. (2018). MedBlock: Efficient and Secure Medical Data Sharing Via Blockchain. Journal of Medical Systems, 42(8). doi: 10.1007/s10916-018-0993-7

Appendix A

Development Setup

The system used for development and testing of the HLF Network package and the JavaScript Application package is configured with the following software setup:

- Ubuntu 16.04
- Docker 18.09.6
- Docker Compose 1.13.0
- Node 8.16.0
- Npm 6.9.0
- Git 2.21.0
- Python 2.7.12
- composer-cli 0.19
- composer-rest-server 0.20
- hyperledger-composer 0.20
- Hyperledger Fabric v1.4.1
- AngualJS

Further information regarding the installation Hyperledger Fabric and Composer can be found here [68]

Appendix B

Setup Guides

To set up the Hyperledger Fabric blockchain network provided by the HLF Network package, follow these steps in the order they are listed below:

1. Place the HLF Network package in the Fabric binaries root directory
2. Edit the configuration files if necessary:
   - compose-with-couchdb.yaml
   - compose-with-raft.yaml
   - configtx.yaml
   - crypto-config.yaml
   - base/compose-base.yaml
   - base/peer-base.yaml
3. Remove any existing cryptographic material or artifacts
4. Run generate.sh to create new cryptographic material and channel artifacts
5. Run start.sh to start the docker containers and configure the peers and channel
6. Check that all containers have started successfully and are running