



CHALMERS
UNIVERSITY OF TECHNOLOGY



UNIVERSITY OF GOTHENBURG

A Decentralised Application for Storing Electronic Health Records using Blockchain Technology

Bachelor's thesis in Computer Science and Engineering

EDENIA ISAAC
HAMPUS JERNKROOK
NILS JOHANSSON
CHRISTOPHER MOLIN
WENDY PAU
DAVID ZAMANIAN

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2022

BACHELOR'S THESIS 2022

**A Decentralised Application for Storing
Electronic Health Records using
Blockchain Technology**

EDENIA ISAAC
HAMPUS JERNKROOK
NILS JOHANSSON
CHRISTOPHER MOLIN
WENDY PAU
DAVID ZAMANIAN



UNIVERSITY OF
GOTHENBURG



CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2022

A Decentralised Application for Storing Electronic Health Records using Blockchain
Technology

EDENIA ISAAC HAMPUS JERNKROOK NILS JOHNSON CHRISTOPHER MOLIN
WENDY PAU DAVID ZAMANIAN

© EDENIA ISAAC, HAMPUS JERNKROOK, NILS JOHNSON,
CHRISTOPHER MOLIN, WENDY PAU, DAVID ZAMANIAN 2022.

Supervisor:

KARL BÄCKSTRÖM, Department of Computer Science and Engineering

Examiner:

ALEJANDRO RUSSO, Department of Computer Science and Engineering

Bachelor's Thesis 2022

Department of Computer Science and Engineering

Chalmers University of Technology and University of Gothenburg

SE-412 96 Gothenburg

Telephone +46 31 772 1000

Typeset in L^AT_EX

Gothenburg, Sweden 2022

Acknowledgements

We would like to express our gratitude to our supervisor, Karl Bäckström, for his guidance throughout the project; as well as to our examiner, Alejandro Russo, for the feedback and advice provided.

We also wish to acknowledge the help received from Chalmers Language and Communication Centre during the writing process of this paper.

Abstract

With the systems currently used in healthcare, medical records are often centrally stored in local databases or cloud services and managed by each individual healthcare provider. Due to this, the information exchange between healthcare providers is often limited. Consequently, upon visiting a new healthcare provider, the patient's past records may be inaccessible, potentially resulting in lower quality of the provided healthcare. This thesis proposes a decentralised web application using blockchain technology, for managing medical records, with the purpose of enabling data sharing between healthcare providers. Another purpose of the proposed system is to provide patients with data ownership, by allowing them to set access restrictions on their data. This empowers patients by giving them more control over their data. The proposed system is implemented using Ganache as a local Ethereum blockchain, IPFS for medical data storage, Firebase for storing application data and React Native for the GUI. The proposed system allows for several future improvements and expansions to be made. Improvements can include a more robust cryptographic scheme and using restricted storage of medical data, as well as a consortium blockchain. Possible expansions include logging read-operations and developing a way for access restrictions to be bypassed in case of an emergency.

Keywords: electronic health records, EHR, medical records, blockchain, Ethereum, IPFS, decentralised applications, Web3.Storage, information exchange

Sammandrag

I dagsläget använder aktörer inom sjukvården sig av system som lagrar patientjournaler centralt i lokala databaser eller molntjänster som hanteras av respektive vårdgivare. Detta begränsar möjligheterna till informationsdelning mellan vårdgivare. När en patient byter vårdgivare kan det leda till att patientens tidigare vårdjournaler är otillgängliga, vilket potentiellt resulterar i att kvaliteten av den givna vården försämras. Den här rapporten presenterar en decentraliserad applikation som nyttjar blockkedjeteknik för att hantera patientjournaler, i syfte att möjliggöra delning av data mellan vårdgivare. Ett ytterligare syfte med det föreslagna systemet är att förse patienter med äganderätt till sin data genom åtkomstbegränsning. Till följd av detta får patienter mer makt då de får mer kontroll över sin data. Det föreslagna systemet är implementerat med hjälp av Ganache som en lokal utvecklingsblockkedja, IPFS för lagring av patientjournaler, Firebase för lagring av applikationsdata och React Native för användargränssnittet. Det föreslagna systemet kan förbättras och byggas ut i flera avseenden. Några förbättringarna som föreslås inkluderar ett mer robust kryptografiskt system och användning av åtkomstbegränsad lagring av patientjournaler samt konsortium-blockkedja. Potentiella påbyggnader som föreslås inkluderar loggning av läs-operationer på blockkedjan samt möjlighet att kringgå åtkomstbegränsningar i nödsituationer.

Nyckelord: patientjournaler, blockkedja, Ethereum, IPFS, decentraliserade applikationer, Web3.Storage, informationsdelning

Contents

Abbreviations	1
1 Introduction	2
1.1 Purpose	3
1.2 Objectives	3
1.3 Related Work	4
1.4 Contributions	5
1.5 Thesis Outline	6
2 Preliminaries	7
2.1 Cryptography	7
2.1.1 Encryption and Decryption	7
2.1.2 Public- and Private-key Cryptography	8
2.1.3 Message Authentication Codes	8
2.1.4 Hash Functions	8
2.1.5 Key Derivation	9
2.1.6 Digital Signatures	9
2.2 Blockchains	9
2.2.1 Blockchain Architecture	10
2.2.2 Making and Validating Transactions	10
2.2.3 Types of Blockchains	11
2.2.4 Limitations of Blockchain Technology	12
2.3 Ethereum	12
2.3.1 Smart Contracts and Gas	13
2.3.2 Ethereum Virtual Machine	13
2.3.3 Networks	13
2.4 IPFS	14
2.4.1 Filecoin	14
2.4.2 Web3.Storage	14
3 Problem Description and Thesis Delimitations	15
3.1 Problem Description	15
3.1.1 Distributed Data Storage	15
3.1.2 Data ownership and Transparency	16
3.1.3 Data Confidentiality and Integrity	16
3.1.4 Authentication	16

3.1.5	Interactive Web Application	16
3.2	Delimitations	17
3.2.1	Security	17
3.2.2	Access Permissions	17
3.2.3	System Setup	17
3.2.4	Distributed File System Implementation	18
3.2.5	Blockchain Implementation and Assumptions	18
3.2.6	Network setup	18
4	Method	19
4.1	Workflow	19
4.1.1	Scrum	19
4.1.2	Trello	19
4.1.3	Code Collaboration	20
4.1.4	GitHub’s Issues	20
4.1.5	Testing	20
4.2	Frontend	21
4.2.1	Prototyping	21
4.2.2	Implementation	21
4.3	Backend	21
4.3.1	Utilised Frameworks	22
4.3.2	Implementation Process	23
5	Results	24
5.1	Application Requirements	25
5.2	Frontend	25
5.2.1	Logging In	25
5.2.2	Patient’s View	27
5.2.3	Medical Personnel’s View	27
5.3	Backend	29
5.3.1	Authentication and Database	29
5.3.2	Cryptographic Setup	31
5.3.3	Distributed Data Storage	32
5.3.4	Downloading and Displaying EHRs	32
5.3.5	Updating and Uploading EHRs	33
5.3.6	Handling Regions and Permissions	33
5.4	Smart Contract and Blockchain	34
5.4.1	Structs	34
5.4.2	Functions and Modifiers	34
5.5	Accomplished Objectives	35
6	Discussion	37
6.1	Limitations and Possible Improvements	37
6.1.1	Administration	37
6.1.2	Security	38
6.1.3	Consortium Blockchain and Restricted File System	39
6.1.4	Logging Read-Operations	39

Contents

6.1.5	Emergency Solution	40
6.1.6	Snapshots	40
6.2	Architectural Motivation	40
6.3	Decentralisation	41
6.4	Ethics	41
6.4.1	Privacy and Security Concerns	41
6.4.2	Environmental Impact	42
6.4.3	Economical Impact	42
7	Conclusions	44
	Bibliography	44

Abbreviations

CID - Content Identifier

dApp - Decentralised Application

EHR - Electronic Health Records

ETH - Ether

EVM - Ethereum Virtual Machine

GUI - Graphical User Interface

ID - Identifier

IPFS - InterPlanetary File System

MAC - Message Authentication Code

Nonce - Number used only Once

OS - Operating System

PBKDF2 - Password-based Key Derivation Function 2

PO - Product Owner

PoW - Proof-of-work

PoS - Proof-of-stake

UI - User Interface

SSN - Social Security Number

1

Introduction

Medical records have been in use since the turn of the 19th century when doctors understood how important it was to document their observations to better diagnose and treat patients [1]. However, the groundwork for the development of electronic health records (EHRs) was not set until the 1960s and 1970s. The usage of medical records was then transformed during the next few decades, as it progressed from paper-based to distributed computer-based department systems, and finally to fully centralised electronic health records [2]. Several technological advancements and the rise of the internet in the 1990s prompted the introduction of web-based EHRs, which allowed for data interchange between disparate systems [1].

EHRs are digital versions of a patient's records, which include personal contact information, medical history, diagnoses, allergies, prescribed medication, test results and the overall treatment plan [3]. It is estimated that 96 percent of hospitals in the United States have adopted it as of 2015. Even though EHRs have shown to improve efficiency and increase positive medical outcomes [3], several concerns have been raised regarding how medical records are stored [4].

Most hospitals and healthcare institutions have their own record management software. This software stores patient data locally in their databases or via a cloud service provider, isolating the data within the institution [4]. The centralised storage results in fragmented patient EHRs amongst various actors and difficulty with information exchange. This difficulty limits health professionals from providing the best care, as they are unable to view a patient's complete and accurate health record. Patients who switch healthcare providers lose access to past records due to the inability to transfer medical data from one institution to another. As a result, patients must undergo medical tests multiple times at separate institutions [4]. In the case of an emergency away from a patient's usual institution, the medical staff may be unable to access important medical information, putting the patient's life in danger [4].

Another concern raised regarding the centralised storage model is the manipulation or leaking of data by the centralised actors [4]. As artificial intelligence algorithms become more robust, voices have been raised regarding the necessity to transfer control over the data from centralised organisations to patients. As the British mathematician Clive Humby said, "*Data is the new oil*" [5]. Many large corporations, such as Google and Meta, currently generate their primary revenue by utilising user data to customise and deliver target advertisement [6]. Nowadays, the data of individuals, particularly medical data, is as valuable as gold.

The usage of blockchain technology for storing medical records could be a feasible solution to the concerns raised against the traditional centralised server architecture. Some companies are already developing alternative systems utilising blockchain technology for plenty of different purposes [7]. These range from keeping the medical data safely stored, to protecting patient's identity and employing blockchain for cybersecurity applications. Decentralising the storage of EHRs is thought to facilitate data exchange between healthcare providers by adding medical information to a distributed database [4].

1.1 Purpose

The purpose of this project is to develop a minimal viable application for accessing, managing and storing EHRs, using blockchain technology. The aim is to achieve a decentralisation and distribution of data to enable easier sharing of data between healthcare providers, compared to the traditional centralised server architecture. With a centralised server architecture, the server owner controls what happens with the patients' data. The proposed system will strive to put patients in control over their own data by allowing them a certain degree of discretion regarding who can access their EHRs.

1.2 Objectives

This section provides high-level objectives that should be achieved within the scope of the project. A more detailed analysis of the problem and delimitations appears in Chapter 3.

In order for the project to be considered a success, the following objectives should be achieved:

- (A) The application should have a user-friendly graphical user interface (GUI) for interaction between user and system.
 - (i) The design should be crafted through several iterations of prototyping.
 - (ii) The application should be easily navigated.
- (B) The system should have secure distributed storage of medical records.
 - (i) The medical records should be stored off-chain on the distributed file system IPFS.
 - (ii) The location of medical records on IPFS should be stored on a blockchain.
 - (iii) Medical records should be securely encrypted using a combination of private- and public-key cryptography.
- (C) All users should be able to authenticate themselves and log in.
 - (i) The application's GUI should enable authentication.
 - (ii) The backend should allow users to authenticate themselves.
- (D) All users should be able to read the medical records that they are authorised to access.
 - (i) The website's GUI should enable users to view medical records.

- (ii) The backend should allow users access only to the medical records that they are authorised to access.
- (E) Medical personnel should be able to create new records and add to existing ones.
 - (i) The application's GUI should enable medical personnel to create new records and add to existing ones.
 - (ii) The backend should allow medical personnel to create new records or add to existing ones.
- (F) Patients should be able to read and edit their basic contact information.
 - (i) The application's GUI should enable patients to view and edit their own contact information.
 - (ii) The backend should allow patients to read and edit contact information linked to their user.
- (G) Patients should be able to dictate who gets access to their medical records.
 - (i) The application's GUI should enable patients to restrict access to their medical records.
 - (ii) The backend should allow patients to restrict access to their medical records.

1.3 Related Work

Several projects have explored the possible applications of blockchain technology within the healthcare industry, and many have looked at how it can be utilised to solve the problems due to centralised EHR systems. McGhin [8] conducted a review of nine different blockchain implementations proposed in the literature and found clear potential for the technology to be used to address several existing issues in the industry. The existing applications analysed in this paper focused on security, authentication, decentralised storage, and patient empowerment. One of the main identified applications of using blockchain in healthcare, was storing EHRs. However, the researched work showed severe limitations when it comes to scalability, security, and key management.

Cao et al. [9] focused on tackling the security and data integrity issues, trying to implement a solution that would prevent malicious actors from tampering with the medical data. Cao et al. was motivated by the fact that once medical institutions generate and outsource EHRs to cloud servers, patients do not physically own their data anymore. A secure cloud-assisted eHealth system was proposed to protect outsourced EHRs from illegal modification by using blockchain technology. The key idea was that the EHRs only could be outsourced by authenticated participants, and each operation on outsourcing EHRs would be integrated into the public blockchain as a transaction.

Chen et al. [10] presented a blockchain system in which the blockchain keeps only the record's search index, but the actual EHR is encrypted and stored on a public cloud server. The EHRs index was constructed through complex logic expressions, and the main aim of the paper was to facilitate the sharing of medical data. Later,

Nguyen et al. [11] identified the use of blockchain alone as an infeasible answer to the decentralisation of cloud-based systems. Instead, the InterPlanetary File System (IPFS) was employed as a decentralised storage. Demir and Kocak [12] evaluated the performance of such a system and came to the conclusion that the time of IPFS upload operations exhibited a linear relationship with file size. These results support that using IPFS will not cause any delays when storing records in the off-chain network.

The authors of MedAccess, the most similar project to this thesis, chose to focus mainly on reducing the cost by proposing an off-chain solution instead of saving EHRs directly on the blockchain [13]. A three-layer system architecture was presented in this case [13]. The initial layer was the blockchain implementation, which held the content identifier (CID) that indicated where the EHRs were saved on IPFS. The second layer was the smart contract, which was used to enforce the rules regulating the transactions that may be executed on the chain or in an off-chain environment. For example, when regulating which users had permission to retrieve an EHR file. The third layer consisted of the off-chain storage, using IPFS as the solution. The proposed approach could reduce the amount of data that needed to be stored directly on the chain, consequently lowering the amount of computing required to add a new block and diminishing the overall cost [13]. The distinguishing contributions of this thesis paper, in comparison to previous work, are highlighted in the following section.

1.4 Contributions

This project aims to develop a decentralised application for storage of EHRs, that allows for shareable medical records and empowers individuals to claim ownership of their data by having complete control over who has access to it.

The proposed system architecture is comparable to past work; yet, given IPFS' limitations on persistent data storage [14], Web3.Storage was adopted as the off-chain solution. In the IPFS network, a node's storage is finite, and nodes need to clear out some of their stored data to make room for new information. This process is known as garbage collection [14]. IPFS does not ensure that any content on the network will be available indefinitely, which can lead to the complete loss of crucial data. As a result, IPFS alone was not deemed suitable for long-term data storage, especially not in the case of EHRs.

By choosing Web3.Storage as the off-chain storage solution, some of the limitations from previous work were overcome. Web3.Storage simplifies the process of making content available through IPFS, while simultaneously ensuring long-term storage using Filecoin [15]. The Filecoin peer-to-peer network has built-in economic incentives to ensure that data is reliably stored over time. When these two systems' capabilities are combined, a complete solution for locating, storing, and retrieving data is provided.

1.5 Thesis Outline

This section provides the outline of the thesis' structure and contents. Chapter 2 gives the necessary background knowledge on technical areas used in the project. Chapter 3 details the identified problems that the project handles, as well as what issues it does not deal with. Chapter 4 explains what frameworks were used for the application development, the project's workflow and how the implementation was done. Chapter 5 describes the developed application and which objectives are accomplished as part of the proposed application. Chapter 6 analyses the limitations of the proposed system and highlights possible improvements. It further motivates the proposed system architecture and illuminates some ethical aspects to consider. Finally, Chapter 7 provides a summary and concluding remarks.

2

Preliminaries

This chapter details certain technical practices and systems pertinent to the thesis, as to provide necessary background knowledge. Included is information on cryptography, blockchains, Ethereum and IPFS to an extent sufficient for understanding the utilised methods.

2.1 Cryptography

Cryptography is a study mainly covering schemes used for encoding an original message (plaintext) into a secret, unrecognisable message (ciphertext) and schemes that allow authorised actors to decode the ciphertext into the plaintext [16].

This section gives the necessary background regarding cryptography and cryptographic primitives. It will explain encryption and decryption, public- and private key cryptography, as well as message authentication codes, hash functions, key derivation and digital signatures.

2.1.1 Encryption and Decryption

The operation for encoding a plaintext into a ciphertext is called *encryption* and the operation for decoding a ciphertext into a plaintext is called *decryption* [16]. Both encryption and decryption takes as input the plaintext/ciphertext to be encrypted/decrypted, as well as a *key* from the set of all possible keys for the specific cryptographic scheme being used [17]. Formally, the encryption function E takes as input an encryption key k_e and a plaintext m and outputs a ciphertext c such that

$$c \leftarrow E(k_e, m).$$

The decryption function D takes as input a decryption key k_d and a ciphertext c and outputs a message m , such that

$$m = D(k_d, c).$$

It is required that D is the inverse of E , i.e.,

$$D(k_d, E(k_e, m)) = m.$$

2.1.2 Public- and Private-key Cryptography

Public- and private-key cryptography differ in how they treat the keys used for encryption and decryption. In public-key cryptography, each participant has their own key pair (k_e, k_d) where k_e is the encryption key and k_d is the decryption key [16]. The encryption key is publicly available to everyone in the system, but the decryption key is private and should only be known by its owner. If a user Alice wants to send a message m to Bob, in a public-key cryptosystem where Bob has the key pair (k_{e_B}, k_{d_B}) , then Alice sends m encrypted using Bob's public key:

$$c \leftarrow E(k_{e_B}, m).$$

Bob can then decrypt the ciphertext c to get m , using his private key:

$$m = D(k_{d_B}, c).$$

In private-key cryptography on the other hand, a set of participants P exchanging messages all share the same key k . This key is used for both encryption and decryption of the sent messages, so every participant uses the functions $c \leftarrow E(k, m)$ for encryption and $m = D(k, c)$ for decryption [16]. The key k is called a *symmetric* key and should only be known to the participants in P .

2.1.3 Message Authentication Codes

While encryption is used for keeping data secret, so called *data confidentiality*, message authentication codes (MACs) are used to discover if data has been modified by an adversary [17]. This provides so called *data integrity*. A MAC system consists of two functions, whereof the first is the signing algorithm S [17]. S takes as input a key k and a message m and produces a tag t such that

$$t = S(k, m).$$

t is kept together with the message for readers of m to be able to check that m has not been modified. This is done using the other function of the MAC system, called the verification algorithm and is denoted V [17]. V takes as input a key k , a message m and a tag t , and outputs **accept** if t is a valid tag for m , or **reject** if t is invalid. It is required that V outputs **accept** if and only if t is the output of S , i.e.:

$$V(k, m, S(k, m)) = \text{accept}.$$

2.1.4 Hash Functions

A cryptographic hash function H is a mathematical algorithm which accepts a block of data x of variable length as input and produces a bit-string $y = H(x)$ of fixed length as output. This output is known as a *hash* or *digest* [16]. A cryptographic hash function is a hash function H which satisfies the following properties:

1. H should be efficiently computable [16], meaning it should be computable in polynomial time [18].

2. Given $y = H(x)$, computing x would take too much time, making it infeasible to compute x . This is called the *one-way property* [16].
3. It should be infeasible to find x_1 and x_2 such that $x_1 \neq x_2$ and $H(x_1) = H(x_2)$, with respect to time. This is called *strong collision resistance* [16].
4. The output of H should meet standard tests for being pseudorandom, i.e. it should not follow any clear deterministic pattern in relation to the input [16].

2.1.5 Key Derivation

As stated in [17], most cryptographic primitives require a cryptographic key as input. This key needs to be as random as possible and of a big enough size to be secure. Pseudorandom cryptographic keys can be derived from a non-random seed, for example a password, with the help of key derivation functions. An example of a password-based key derivation function is *PBKDF2* (Password-based Key Derivation Function 2), which is based on hash functions. This function takes as input a password, the number of iterations to run the algorithm for, and a salt. The salt is a random string that is hashed together with the password to prevent two of the same input seeds, passwords in this case, to produce the same output [17].

2.1.6 Digital Signatures

A digital signature has the exact same function as a handwritten signature, it shows a proof of identity and intent [19]. According to [16], a digital signature must have the following properties:

1. It must verify the author, the date and the time of the signature.
2. It must authenticate the contents at the time of the signature.
3. It must be verifiable by third parties, to resolve disputes.

There are a few different ways of creating and verifying a digital signature, but one simple way is to hash the document and then sign that hash by encrypting it with the private key [16]. This signature can later be verified by hashing the document again, decrypting the signature with the public key and then comparing the two hashes. This can be done by anyone that knows the public key.

2.2 Blockchains

A blockchain is a digital ledger of transactions that can be duplicated and distributed across multiple independent computer systems – or *nodes* – on a computer network [20]. A blockchain is divided into a sequence of blocks containing a number of transactions [21]. A transaction can contain everything from account balances and trust arrangements to random data; in short, anything that can currently be represented by a computer [22]. Each time a new transaction occurs, a record of that transaction gets added to every participant's ledger [20].

2.2.1 Blockchain Architecture

As illustrated in Figure 2.1 and summarised in [21], a blockchain is made up of a sequence of blocks. Each block uniquely references the one preceding it by the preceding block’s hash value. This holds for all blocks on the blockchain except the first block, which is called the *Genesis block*. With the only exception being the Genesis block’s lack of reference to any previous hash, each block on the blockchain contains:

- (i) The cryptographic hash of all transactions in the block. This hash value must meet some certain requirement, used to make the creation of new blocks computationally demanding.
- (ii) The timestamp of when the block was made.
- (iii) A “Number used only Once” (nonce), used for the consensus algorithm¹.
- (iv) The cryptographic hash of the previous block.
- (v) A number of transactions published on the given block.

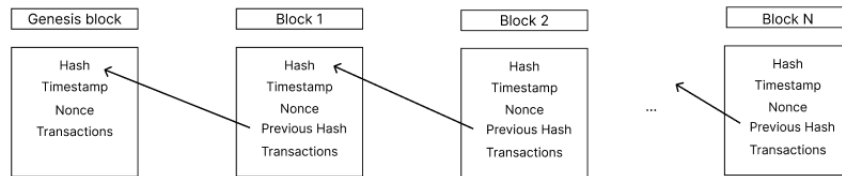


Figure 2.1: A simplified visualisation of a typical blockchain architecture.

According to [22], since every block, except the Genesis block, contains the hash of the previous block, altering the data on a block would be extremely difficult. It would require changing the block’s own hash and all the subsequent blocks’ hashes, which is infeasible given that the rest of the network consists of honest actors. This property makes the blocks on the chain immutable. New information can only be added as new blocks.

2.2.2 Making and Validating Transactions

Each participant in the blockchain network is associated with a public/private key pair. Participants who make a transaction must guarantee its authenticity by digitally signing the transaction with their private key [21]. This signature can then be verified by other participants in the network.

A block of transactions needs to be approved before it is added to the blockchain. Every decision to add a block to the blockchain is made by consensus, which means that the majority of the participants on the network need to agree that the block is valid [23]. This can be done with, for example, Proof-of-work (PoW) or Proof-of-stake (PoS) which are the two most common consensus mechanisms [24].

In PoW, all nodes in the network compete to be the first one to compute the hash value of the new block, using their own computing resources [21]. They do so by

¹Consensus algorithms are explained in Section 2.2.2.

iteratively changing the value of the block's nonce, to get a new hash value with each iteration until a valid hash is found. The hash value computed must be equal to or smaller than a certain given threshold. Once the hash is found, the winning node broadcasts the new block across the network, upon which all other nodes must validate that the new block is indeed valid. If the new block is accepted, the new block is appended to the blockchain [21] and the winning node is awarded with cryptocurrency [25][26]. This provides incentive for people to host nodes on their computers and take part in the validation process. The nodes taking part in calculating the hash are called *miners* and the computation process itself is called *mining* [21].

PoS uses randomly selected participants to validate transactions and thus removes the competitive aspect of validation that PoW is based on [27]. PoS is also more energy efficient compared to PoW. To become a validator of the network, a participant must deposit a part of their owned cryptocurrency [27]. This deposit is known as the *stake*. From the group of validators, a single validator V is chosen at random to create a new block. The validators that were not selected at a given time are responsible for confirming the block proposed by V . Confirming another validator's block is called *attesting*. Validators are awarded with cryptocurrency for both proposing new blocks and attesting to other validators' blocks. Attesting to malicious blocks leads to the validator losing their stake [27].

2.2.3 Types of Blockchains

Blockchain systems are generally divided into three categories: public, consortium and private blockchains [28]. On public blockchains, anyone can read and send transactions, as well as participate in the validation process. Such blockchains are decentralised to a large extent. A private blockchain constitutes a centralised network, where only participants within the organisation that is controlling the network are allowed to send transactions and participate in the validation process [21][28]. Read permissions may be either public, as for public blockchains, or restricted. Finally, there are consortium blockchains. These are like a hybrid between public and private blockchains and may be shared by multiple organisations, offering partial decentralisation [28]. Only a predefined set of nodes may participate in the validation process on consortium blockchains. Reading access may either be public or private.

In [28], some advantages of public and private blockchains respectively, are that private blockchains can be advantageous when a central organisation needs to be able to interfere with the system in some way; for instance, reverting transactions or changing the rules of the blockchain. Furthermore, in [28] it is stated that private blockchains allow for cheaper and faster transactions, as well as more privacy. Public blockchains on the other hand, can serve to protect users from the application developers, which may prove advantageous in certain scenarios [28]. Additionally, the potentially larger number of nodes on a public blockchain makes it more resistant to tampering of transactions, than private and consortium blockchains [21].

2.2.4 Limitations of Blockchain Technology

Vitalik Buterin, co-founder of Ethereum, states that blockchains inevitably need to make trade-offs between decentralisation, security and scalability. He states that “simple” blockchain technologies can only achieve two out of these three simultaneously [29] and even with sophisticated improvements, there are some trade-offs to be made [29][30]. Buterin calls this the *scalability trilemma* [29]. Common blockchains such as Bitcoin and Ethereum achieve high decentralisation and security, but struggle with scalability [29]. This manifests itself in the form of low transaction throughput and high latency [31]. The scalability problem arises as the blockchain grows and the number of transactions and blocks increase. This requires more effort from the nodes involved in storing and validating transactions [31]. Hence, the scalability of the blockchain, and the size of blocks, is bounded by individual nodes’ computing power, bandwidth and storage [30]. One suggestion to overcome this issue, and to some extent achieve all three parts of the scalability trilemma, is to use sharding [29][30].

Even blockchains which lean more towards decentralisation and security in the scalability trilemma can have some security and privacy risks. In [32], one security risk is that typical blockchains lack *transactional privacy*, meaning that the values and actions of smart contract² executions are publicly visible, as well as balances and transactions issued by user accounts. Even though users are anonymised behind a public key acting as their pseudonym [32], [33] shows how users can be de-anonymised on the Bitcoin network. This is done by linking user pseudonyms to the IP addresses where users have generated their transactions.

Blockchains suffer from additional security risks by a so-called *51% attack* [34]. In a 51% attack, an adversarial group of miners take control over a majority of the nodes on the blockchain network to set the rules of the system, e.g., reverting new honest transactions and maximising their own profit [21][35]. In [25], it is shown that the 51% attack is a problem in Bitcoin, even if the colluding group of miners initially control only a smaller fraction of the nodes. By attracting other profit-seeking miners into their group they could eventually form a majority.

2.3 Ethereum

The decentralised, open-source blockchain Ethereum is a platform primarily for cryptocurrency. The currency on the platform is called Ether (ETH) [36]. Compared to its precursors in cryptocurrency, such as Bitcoin, Ethereum uses far more advanced smart contract functionality [37]. As an open-source blockchain, Ethereum offers a way for users to create their own decentralised applications using Ethereum’s blockchain [36]. Ethereum uses Proof-of-work as its consensus mechanism but has plans to move over to Proof-of-stake [38].

²For an explanation of smart contracts, see subsection 2.3.1.

2.3.1 Smart Contracts and Gas

Programs can be run on the Ethereum blockchain. These programs are called smart contracts and are in essence a piece of code, as well as the data it uses, occupying a specific block on the blockchain. Smart contracts are mainly used for validating transactions on the blockchain. This can be done automatically due to their ability to self-execute. Once the criteria of the contract is fulfilled, the specified instructions will be executed. Solidity and Vyper are the programming languages supported by Ethereum for writing smart contracts [39].

As stated in [40], all transactions made on the Ethereum blockchain, whether it be a transfer of Ether from one account to another or the deployment of a smart contract, will cost a fee if the state is changed in any way. This means that retrieval of data, for example with getters, does not cost any fee. The fee is paid in a small amount of Ether and is normally referred to as *gas*. Supply and demand of computational power from Ethereum's Virtual Machine dictates the cost of gas for each transaction [40]. Transactions that are called can be dropped or fail for a number of reasons [41]. For instance, the provided transaction fee may be too low [41] or the conditions in the *require*-clause may not be met upon the functions execution [42]. The fee that is paid for the transaction to take place is covering the computing power needed to compute the transaction. This means that a fee will be collected regardless of whether the transaction succeeds or fails [43].

As with transactions, when the contract has been placed onto the chain, it can no longer be modified [44]. Modifying a contract can only be done by creating a new instance of it and redeploying to the chain, requiring another fee, but this will be a new contract as far as the Ethereum Virtual Machine is concerned [44].

2.3.2 Ethereum Virtual Machine

The platform used for developing decentralised applications using Ethereum is called the Ethereum Virtual Machine (EVM). All smart contracts and Ethereum accounts are stored in the EVM [45]. The EVM exists as one single entity maintained by thousands of connected computers running an Ethereum client. The purpose of the EVM is to define the rules for computing a new valid state from block to block [46].

2.3.3 Networks

As stated in [47], Ethereum is not only a public blockchain, but also a protocol for building blockchains. This means that there can be multiple different independent networks using this protocol. Ethereum networks can be both public and private. One example of such networks is the mainnet which is the primary public blockchain. There are also test nets, both public and private, that are used by developers for testing their smart contracts and protocols. Examples of such test nets are Ropsten and Rinkeby. Ethereum can also be used in consortium networks. All different networks use Ether as the currency but only mainnet currency has any real value [47].

2.4 IPFS

InterPlanetary File System (IPFS) is an open-source distributed hypermedia protocol [48]. It is used by various products and services, most commonly for content delivery [49]. It supports storing files and directories by splitting them into smaller chunks, and dynamically distributing files based on demand [48], [50].

Upon storing a file on IPFS, the file is separated into pieces, cryptographically hashed, and then given a unique fingerprint called *content identifier* (CID) [48]. The CID is used as a reference for the file's contents, which other nodes use when they need to look up the file. Modifying a file and uploading a new file to IPFS will result in a different CID referencing the new content. The original content can still be referenced via the original CID. This enables resistance to censorship and tampering. However, IPFS does not guarantee that the data is persistent forever, as IPFS cannot guarantee that the nodes are always available to cache and store the resources [51]. Therefore, IPFS in itself is not optimal for long-term storage.

IPFS may be used as a complement to using a blockchain for data storage. Storing large amounts of data on a blockchain can quickly clutter it, making it expensive and inefficient [52]. Therefore, IPFS may be used to provide off-chain storage, reducing the amount of data stored on the blockchain. IPFS can be used by blockchain developers, where the content addressing enables off-chain storage for large files and functions to put permanent and immutable links in transactions [48]. This enables securing and timestamping of the content, without putting the data itself on-chain. An example would be to store the files on the IPFS network, while storing the transaction links or CID to the files on the blockchain.

2.4.1 Filecoin

Filecoin is a decentralised file storage that is built upon IPFS and aims to provide a long-term distributed storage marketplace while also solving the data persistence problem [53]. Filecoin works by having users request and pay for storage of their files. Other users may provide storage of these files and be awarded with the monetary token Filecoin for doing so. Filecoin provides open markets for anybody to store or retrieve their files.

2.4.2 Web3.Storage

Web3.Storage is a free, decentralised file storage that is built upon the decentralised storage networks Filecoin and IPFS for data accessibility over the public IPFS network [54]. To solve the data persistence problem that arises from using IPFS, Filecoin is used to provide a certain amount of storage for an agreed period of time [15].

3

Problem Description and Thesis Delimitations

The purpose of this thesis is to develop an application for EHR management that enables easier sharing of data between healthcare providers, compared to the centralised server-architecture typically used today. The application should also equip patients with data ownership. Section 1.2 provides high-level objectives to achieve these goals. This chapter describes the identified issues to be solved in more detail and the delimitations set by the authors.

3.1 Problem Description

A number of sub-problems have been identified to achieve the general purpose of this project. Firstly, some kind of distributed data storage will be needed. Secondly, access to the data needs to be restricted despite being stored in a distributed setting. This includes the problems of data ownership, authentication and data confidentiality. Another issue is the immutability property of blockchains and how it can still be used for EHRs - which should be mutable. The system must also ensure that data is not modified by unauthorised actors, adding the problem of data integrity. Finally, the system must allow for user interaction via a user interface.

3.1.1 Distributed Data Storage

The application needs to provide a distributed solution for storing EHRs. Storing the EHR contents on a blockchain was first considered, but storing large amounts of data on blockchains can be inefficient³ [55]. However, using a blockchain adds advantageous properties with respect to security. The consensus mechanism prevents malicious actors from modifying published data on the blockchain and further prevents adversaries from establishing fake transaction histories as being true. Additionally, a blockchain adds transparency to the system by acting as a log. Blockchain technology will therefore be combined with a distributed file system to provide storage of medical data. In particular, the blockchain will contain references to each new version of a patient's EHR. Storing the EHRs in a system separate from the blockchain will allow for updates of the EHRs to be made by publishing new versions, while references to previous versions are safely recorded on the blockchain due

³See Section 6.2 for further discussion on using the blockchain for EHR storage.

to its immutable property. A smart contract will need to be implemented to provide the logic required to interact with the blockchain.

3.1.2 Data ownership and Transparency

The distributed data storage will be used by multiple different actors, possibly from widely different geographical areas. As a result, the question of data ownership and transparency becomes extra important, with respect to patients' privacy and integrity. The data storage will not be administered by each healthcare provider, as with the traditional client-server architecture. Subsequently, giving patients the ability to choose exactly which actors to allow access to the patient's EHR will be a central aspect of the system. Thus, the system needs to provide functionality for patients to set their permission settings and access control needs to be enforced throughout the application. This provides patients with data ownership.

To further equip patients with ownership over their data, the system will provide a certain degree of transparency of how the data is updated and used. The blockchain will not only add security benefits, but also serve as a log of function calls on the data. This allows patients to more transparently follow the life cycle of their data.

3.1.3 Data Confidentiality and Integrity

Since data will be stored distributively on a common network of nodes, it must be encrypted to prevent unauthorised actors from reading it. A combination of public- and private-key cryptography will be utilised for this purpose. A main aspect of this will be to use long enough keys for the data to be kept secret from adversaries for a foreseeable future.

Blockchain technology helps in providing data integrity via immutability and consensus [21]. This will however be supplemented by also storing the data along with MACs. This will allow an actor that accesses the data to verify that it has not been altered by some malicious adversary.

3.1.4 Authentication

To allow only authorised actors to access EHRs and interact with the system, authentication via a log-in service will be needed. The log-in process will uniquely identify users and give them the correct system permissions. For this purpose, a database with user information and permissions will be needed.

3.1.5 Interactive Web Application

To enable interaction with the system, a web-based user interface will be developed. This will allow users to log in, update their settings, read EHRs that they have access to and allow healthcare providers to write medical records.

3.2 Delimitations

This project is mainly meant to serve as a proof-of-concept proposal for how a blockchain-based, decentralised application for EHR management could be implemented. The main aspects considered for this are the questions of data storage and access permissions. Due to time constraints and limited expertise in certain areas, some relevant aspects will be considered out of scope for this particular project. These delimitations are discussed in the following subsections.

3.2.1 Security

The security of a system dealing with large amounts of private data is of course crucially important with respect to individual privacy and integrity. However, the security expertise of the authors is limited and best practises within the security field will not be researched, due to the limited time frame of this project. Thus, the security measures taken may not adhere to standard security practises nor be as robust nor secure as should be required for this kind of application. Despite this, a genuine attempt at implementing security measures will be done. Additionally, the security of the system will not be formally nor practically evaluated. For example, penetration testing will not be performed. Coming up with a robust and solid cryptographic scheme for the system will be deemed out of scope, meaning that the handling of things such as key distribution and key storage could most likely be improved upon. In addition, certain parts of the system will involve some kind of identifier (ID) of patients written in plain text. Where applicable, anonymity will be enforced by using the hash of a patient's social security number (SSN) as ID. However, further measures to enforce anonymity will not be taken.

3.2.2 Access Permissions

The initial idea regarding access permissions was that patients would only be able to limit which actors can read their EHR. All medical personnel were to be allowed writing permissions for all patients, in accordance with Objective (E). To facilitate the use of a simpler cryptographic scheme however, the permissions settings will apply to both reading and writing permissions. Thus, the application will allow healthcare providers to read and write EHR entries only for patients who have granted permission to the given provider. Implementing a solution that would allow unauthorised medical personnel to read and write EHR entries for patients in case of emergency was discussed but left out of scope due to time constraints.

3.2.3 System Setup

It will be assumed that some central authority exists to handle the registration of users and populate the EVM with all necessary instances of the contract's structs, as well as provide users with a required MetaMask account and Ethereum wallet. This registration process could be done via an admin-side of the application, but this will due to time constraints not be implemented. Instead, the migration file for deploying

the smart contract on the blockchain will provide an initial population of instances on the EVM, but no functionality for adding new ones will be implemented.

3.2.4 Distributed File System Implementation

Because of the limited time frame, a custom distributed file system will not be implemented. Instead, the project will rely on the public framework Web3.Storage for persistent storage on IPFS.

3.2.5 Blockchain Implementation and Assumptions

The project will not use a custom blockchain implementation due to time constraints. Instead, a local Ethereum blockchain will be used. The system will be developed with both a public blockchain and a public distributed file system in mind, but real implementations may be better off considering a consortium blockchain and restricted data storage. This is discussed in further detail in Section 6.1.3.

The blockchain may serve as a log to add transparency to the system regarding how patients' data is used. However, the project will due to a limited time frame not provide a way for users to inspect the blockchain and the transactions executed. If the system were to be deployed to the Ethereum mainnet or one of the common testnets, then Etherscan [56] may be used for this purpose. For a custom network, a custom solution would be needed.

3.2.6 Network setup

Due to lack of time and knowledge, the system will not be tested on any custom network consisting of multiple machines, nor will it be considered how the network would be set up and configured in a real-world adoption of it. The project will thus not evaluate how the system fares with storage, accessibility nor security in terms of computer networking.

4

Method

This chapter describes how the results of this thesis were achieved. Furthermore, it explains how the various technologies and concepts mentioned in Chapter 2 were utilised to achieve said goals.

4.1 Workflow

The project's workflow relied on a strategy called agile software development (Agile). Agile refers to a group of software development methodologies whose goal is to deliver incremental improvements through iterative development [57].

4.1.1 Scrum

Scrum is a framework for agile development [58], which was used throughout the project. Generally this process would demand two individuals with special roles, a product owner (PO) and a scrum master. As the group members themselves were the only stakeholders in the project, no PO was elected. Development began with only a scrum master with the purpose of making sure each team member understood and followed the intended process.

4.1.2 Trello

A Scrum-board was made on the website *Trello* [59], which features virtual bulletin boards. *Trello* was used to organise and distribute tasks within the team. The Scrum-board facilitated the organisation of each task, by placing each task into one of the following categories:

- **Product backlog.** General features, enhancement and requirements to be achieved eventually.
- **Sprint backlog.** More focused and concrete tasks that were scheduled to be worked on during the upcoming sprint.
- **Current Sprint.** Tasks that were in-progress or under development. These were scheduled to be completed during the current sprint.
- **Pending Review.** Although not always applicable, tasks that involved submitting code were to be reviewed prior to merging into the main code base.
- **Done Sprint.** For tasks that were considered completed, which is useful as it conveys when team members could be delegated new tasks.

With exceptions around the exam- and re-exam periods, the sprints were a week long.

4.1.3 Code Collaboration

In order to simplify collaboration, storing and tracking of the code, the online software development platform GitHub was used. GitHub has the feature to create separate branches of a program's code, enabling the group to develop code in parallel to one another in a simple way. Using pull requests, which is a GitHub feature, the team members were able to get feedback and approval on their code before merging it into the master branch. This ensured good quality of code and also helped to keep all members up to date on what was being done and how it was all coming together.

4.1.4 GitHub's Issues

The previously mentioned Scrum-board was used throughout the implementation phase, but as it often was overarching milestones rather than specifics, another documentation method was needed. GitHub's *Issues* [60] was used to document encountered bugs, or missing functionality, with the option to elaborate on the issues with further detail and useful images.

GitHub *Issues* was gradually used more and more throughout the project, and was used both for the development of the frontend and the backend. It also served as a much better channel for communication regarding a specific issue.

4.1.5 Testing

During the planning stages of the project, the group was considering using test driven development (TDD). However, this proved difficult and somewhat ineffective. While unit testing was done in as wide of a capacity as the group was capable of, it was not TDD. The major contention was the testing of real-time systems and frontend-components, which was not trivially done. Thus, unit testing was done as much as the authors' expertise allowed for, while the rest was tested manually.

There were occasions where unit testing could not reliably find a bug. Thus, a *Quick-Check*-inspired testing library, named *Fast Check* [61], was used. It provided a way to test the code using property-based testing, which was able to automatically generate test inputs that may not have been tested while manually creating unit tests.

Finally, GitHub Actions [62] enabled the project to have Continuous Integration (CI). This meant that the code was continuously checked to be functional on several Operating Systems and Node.js distributions. These checks were exceptionally useful for ensuring the code was functional for everyone in the team. Moreover, these checks were done automatically when committing code to GitHub. However, in the later part of the development, a Truffle dependency was introduced as the

blockchain was being added to the project. This led to the CI-checks failing for certain Windows distributions, and the CI-checks were thereafter not as reliable of an indicator of issues as they were initially.

4.2 Frontend

The frontend enables the interaction between users and the system, and takes the form of a web application. This allows the application to be independent of the user's Operating System and will not require users to download and install any additional software. Nowadays, JavaScript is present in many, if not most, websites [63] and comes supported by all major web browsers [64]. Due to previous experience among the authors, the frontend was developed with the JavaScript library *React Native* [65] - originally created by Facebook (now Meta) [66].

This part of the application is responsible for the parsing and validation of inputs from the users, and the interaction with the various Backend-systems, via an accessible graphical user interface (GUI).

The frontend was never evaluated nor tested using any *user testing*-technique, nor was it designed for a targeted user group. However, the frontend was designed with user experience in mind and followed several suggestions found in [67].

4.2.1 Prototyping

The appearance and a rough outline for the behaviour of the frontend was prototyped using Figma [68], over a duration of roughly two weeks. During these two weeks, the frontend was iteratively improved upon with consideration for the feedback from within the team, as well as the requirement for simplicity due to the time limitation. The prototyping done is considered to have solved Objective (A)(i).

4.2.2 Implementation

Once the design for the prototype was finalised, the process of implementing it could begin. With a newly created *React Native* project, the screens were implemented in the order in which the user would normally access them. Hence, the log-in screen was developed first.

4.3 Backend

The backend, also called the data access layer, is hidden from the user, handling business logic and data, displayed by the frontend. To achieve the goal of a distributed storage of medical records, the aim was to develop the project using decentralised frameworks.

4.3.1 Utilised Frameworks

The backend was developed using:

Node.js. An open-source and cross-platform JavaScript runtime environment that runs in a single process. Node.js enables developers to simultaneously write server-side and client-side code in the same language [69]. In the project, Node.js' own cryptography module, named *Crypto* [70], was used for all cryptographic functions, e.g. to encrypt and decrypt EHR-data.

Web3.js. A collection of libraries for interacting with local or remote Ethereum nodes using IPC, WebSocket, or HTTP [71]. Web3.js was used for interacting with the blockchain from the source code. It provided a way to call functions on the smart contract.

Web3.Storage acted as an intermediate during the uploading process of EHRs to IPFS. It makes data persist on IPFS by the use of FileCoin, as well as provides the CID of the newly uploaded EHR-archive on IPFS.

Firestore. An application development software, backed by Google, that helps developers to build and run apps [72]. Firestore offers a number of services, where *Firestore Realtime Database* [73] and *Authentication* [74] is used in the project. The Realtime Database was used for storing the cryptographic keys and user contact information. Meanwhile, the Authentication was used for enabling secure and easy authentication of users.

Truffle. A development environment for blockchains using the EVM for providing a testing framework and asset pipeline [75]. Truffle provides, among other things, compilation, testing and deployment of smart contracts, as well as network configurations for interacting with the blockchain.

Ganache is a local blockchain used for Ethereum distributed application development. It facilitates development, testing and deployment of decentralised applications in a deterministic and safe environment [76]. Ganache was used to provide a local blockchain for storing the CIDs returned from Web3Storage, and patients' permission settings, among other things.

Solidity. A high-level, object-oriented programming language for creating smart contracts that runs on the EVM [77]. Solidity was used to create the smart contract used for storing application data, such as references to the IPFS location of each patient's EHR.

MetaMask. A crypto wallet [78] and a tool for handling account management and user connection to the Ethereum mainnet and several testnets [79], as well as local networks [80]. MetaMask is available as both a browser extension for Google Chrome, Firefox, Brave and Microsoft Edge as well as a mobile application for iOS and Android [81].

4.3.2 Implementation Process

The authentication via Firebase was the first to be set up, as it would be needed immediately in the frontend. Firebase's real time database was soon thereafter used to test the fetching and uploading of contact details shown in the patient overview screen, presented in section 5.2.2.

The uploading and downloading of files to and from IPFS, via Web3Storage, was implemented next. Initially in Node.js, and soon after, Web3Storage was used by the frontend to upload new files and download old ones.

Next, the functionality for encrypting and decrypting EHRs was added. All cryptographic functionality was provided by the aforementioned Node.js-library Crypto. The library has several methods that were utilised. Both private keys and public keys, as well as password-derived keys, were generated using Crypto, to be used for encrypting and decrypting data. The data that was encrypted and decrypted were EHRs and some of the keys themselves.

The private-key cipher used in the application is *AES* in *GCM mode*. Encryption with this cipher outputs both a ciphertext and a MAC [17]. For the encryption, an *initialisation vector* (IV), is also given as input, along with the plaintext to encrypt and the secret key. The IV is a random byte sequence of fixed size generated for each encryption.

The public-key cipher used in the application is that of Crypto's `publicEncrypt(·)` and `privateDecrypt(·)`. Although [70] does not explicitly state what cipher these functions use, it implicitly seems to state that the public-key cryptosystem RSA is the one used.

The smart contract was continuously improved upon and adjusted to the ever-changing needs and requirements. Although the smart contract was being planned and prepared beforehand, it was not utilised until Truffle - and Ganache - was used in the project. Truffle and Ganache were together used to set up the local blockchain. Web3.js enabled the interaction between the application and the blockchain and MetaMask was used to connect to the blockchain as well as import and handle Ethereum accounts provided by Ganache. The migration script that populated the blockchain added three medical personnel and two patients in total.

5

Results

This chapter gives details on each part of the proposed application and how they interact with each other, as well as requirements needed to use the application. An overview of the interaction between different parts of the system is given in Figure 5.1. The source code for the application can be found in [82]. Details on how to download and use the source code, as well as full resolution-versions of all images in this chapter, is given in the `README.md`. The application is mainly divided into two parts: frontend and backend. The frontend provides a user interface for interaction with the system via a web application. The backend consists of several parts: regular JavaScript source code, a central database for authentication and storage of application data, as well as a smart contract to be deployed to an Ethereum network and a distributed data storage for storing medical data.

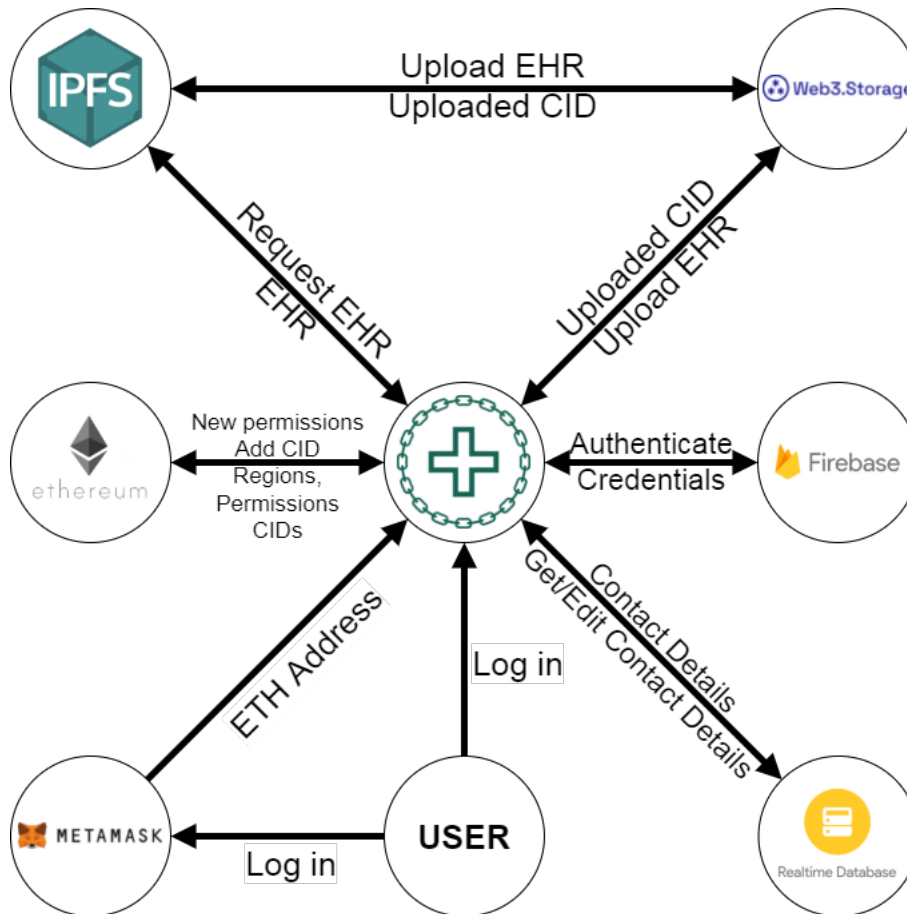


Figure 5.1: An overview of the system's interaction with other services.

5.1 Application Requirements

To use the application, users will need to have installed MetaMask in one of the browsers or mobile applications with MetaMask-support⁴. The authors have however only used browser-versions. Having the MetaMask extension installed on their chosen platform, the user will need to create a MetaMask account and import their Ethereum wallet, as well as select this wallet as the current active one in MetaMask. This is required to be able to log in to the application, as each user is associated with a unique Ethereum address and cannot interact with the system without it. Of course, the user's Ethereum wallet must also contain a sufficient amount of Ether to be able to fully interact with the system.

As the application is still in development and serves only as a prototype, there are some additional requirements that current users will need to adhere to. This includes several of the tools and frameworks described in Sections 4.2 and 4.3.1. This would of course not be required if the application went into production, but the requirements listed in the first paragraph of this section would still be firmly in place.

5.2 Frontend

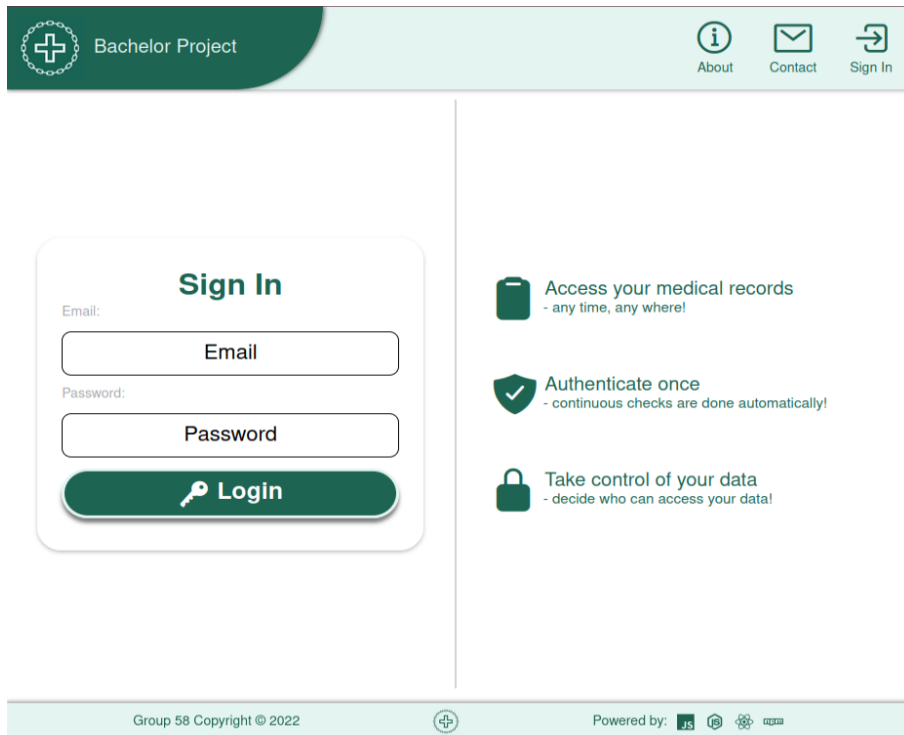
The frontend provides the UI via a website, through which users can interact with the system. The UI supports the login process for patients and medical staff alike, as well as the interfaces for viewing and adding EHR-related content to the system. Additionally, patients can edit their contact details and permission settings.

5.2.1 Logging In

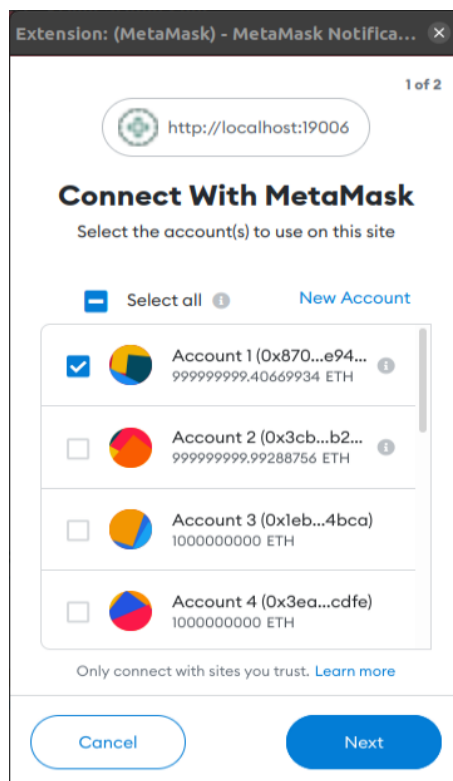
Upon launching the application and visiting the website, users are first met with the login-screen. At the login-screen, the user's MetaMask extension will prompt the user to connect their Ethereum account to the website, provided this has not been done during a previous visit to the site. If the user rejects the request, then the page will prompt the user to accept the prompt to be able to use the website. After that the page will automatically be reloaded. Having accepted the MetaMask connection, the user must enter their email and password to authenticate themselves and be redirected into the system. This functionality solves (C)(i) from the Objectives. Figure 5.2 below shows the UI for this application flow.

⁴See Section 4.3.1 for details regarding what platforms that MetaMask is available on.

5. Results



(a) The login page.

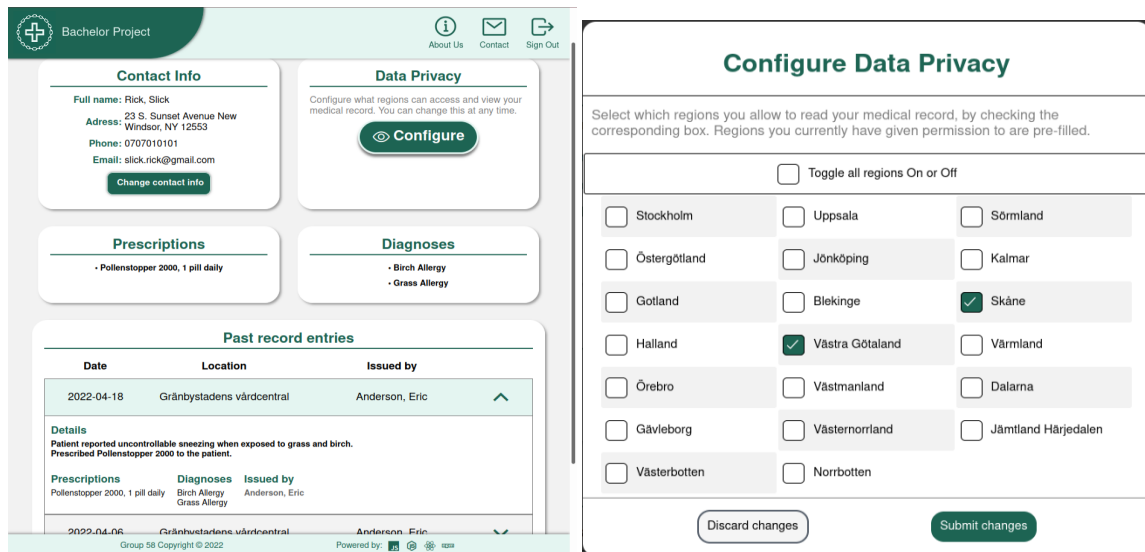


(b) The MetaMask prompt.

Figure 5.2: The login view.

5.2.2 Patient's View

After the login-screen, the application flow differs depending on the role of the user. If the user is a patient, then they are redirected to the patient-overview-screen. Here, patients may read their previous EHR entries, change their contact information as well as view and change their permission settings. This part of the frontend solves Objectives (F)(i) and (G)(i), as well as parts of (D)(i). Permission is granted on a regional level, meaning that if a patient grants access to the region of, for example, Stockholm, then all medical personnel at all healthcare institutions in Stockholm will have read and write permissions with respect to that patient's EHR. The UI presented at the patient-overview-screen is shown in Figure 5.3.



(a) Patient's overview page.

(b) Interface for configuring which regions can access the patient's data.

Figure 5.3: The available views for patients.

5.2.3 Medical Personnel's View

If the user is part of the medical personnel, they are redirected to the patient-search-screen after the login-screen. Here, medical personnel may search for a patient using the patient's SSN. If the medical personnel has access to the given patient, i.e., they work for a healthcare institution within one of the regions that the patient has granted access to, then they are redirected to the doctor-overview-screen. On this screen they are presented with information about the patient searched for. The doctor-overview-screen is similar to the patient-overview-screen in all regards, except the part detailing the patient's set of permitted regions. The medical personnel will not be able to view the patient's permission settings. Instead, at this spot in the UI is a button for medical personnel to be redirected to the new-entry-screen, where a new EHR entry for the given patient can be written. If the access control at the search screen finds that the medical personnel is not permitted by the patient searched for, then the search will be rejected and the redirection will be cancelled.

The access control enforced here is part of solving Objective (D)(i) and does so via the backend. Thus, Objective (D)(ii) is also solved via the enforced access control. The UI presented at the patient-search-screen and doctor-overview-screen is shown in Figure 5.4.

(a) Part of the patient-search-screen.

(b) The patient overview from the perspective of medical personnel.

Figure 5.4: The application-flow for medical personnel.

Entry information that may be given by the medical personnel on the aforementioned new-entry-screen include issued diagnoses and prescriptions, as well as free-text notes (details) from the appointment. The entry may either be submitted to be added to the patient's EHR, or discarded. This part of the frontend solves Objective (E)(i). Upon submitting a new EHR entry, the user is redirected back to the patient-search-screen. The UI presented on the new-entry-screen is shown in Figure 5.5.

The screenshot shows a web interface for adding a new EHR entry. At the top, there's a header with a cross icon and 'Bachelor Project', and navigation links for 'About', 'Contact', and 'Sign Out'. Below the header is a 'Cancel & Return' button. The main section is titled 'Add Entry'. It features a 'Patient ID' field with the value '9801011111' and a yellow warning box that says 'Ensure this is the intended patient.' Below this is a 'Details' section with a text area containing the text: 'Patient reported uncontrollable sneezing when exposed to grass and birch. Testing indicate the patient is allergic to birch. Prescribed Pollenstopper 2000 and Nose-Clear Spray to the patient.' To the right, there are sections for 'Prescriptions' and 'Diagnoses'. The 'Prescriptions' section lists 'Pollenstopper 2000' (1 pill daily) and 'Nose-Clear Spray' (1 dosage per nostril per day), each with a 'Remove' button. Below these are input fields for 'Name of prescription' and 'Dosage', with an '+ Add' button. The 'Diagnoses' section lists 'Birch Allergy' with a 'Remove' button, and an input field for 'Diagnosis' with an '+ Add' button. At the bottom of the form is a large green 'Complete' button with a checkmark icon. The footer contains 'Group 58 Copyright © 2022' and 'Powered by: JS, React, Node, MongoDB'.

Figure 5.5: The screen for writing and submitting new EHR entries.

5.3 Backend

The backend provides the logic of the application and consists of multiple different parts that are interconnected to constitute the overall system. The central parts of the backend are the smart contract, the distributed data storage, the database and the source code connecting and adding to all these parts. This section explains the role of each part and the application flow as it is run on the backend, with respect to user input and actions taken from the frontend. The details of the smart contract implementation and the blockchain network used is given in Section 5.4.

5.3.1 Authentication and Database

Upon logging in, users are authenticated to be who they claim to be, solving Objective (C)(ii). This is handled via Firebase's Authentication service, which uses users' email and password for authentication and maps each user to a unique identifier (UID). The UID is in turn mapped to the user's SSN in Firebase's Realtime Database.

The mapping from UID to SSN is stored in the database tree `mapUser`. `mapUser` also contains which role each user has within the system; `patient` or `doctor`⁵. The

⁵'Doctor' and 'Medical personnel' are used interchangeably within the source code of the project.

role of the user is checked when the user has been successfully authenticated and determines the UI presented by the frontend.

In addition to the things mentioned, `mapUser` includes information essential to the cryptographic scheme used throughout the application. In particular, `mapUser` contains for each user:

- (i) Their public key used for encrypting symmetric keys within the system.
- (ii) A random salt used in key derivation.
- (iii) Their private key, encrypted with a secret key derived from the user's password and salt upon logging in. An IV is also stored along with the private key.

Section 5.3.2 explains the cryptographic scheme that this information is used within. The schema of `mapUser` is given in Figure 5.6.

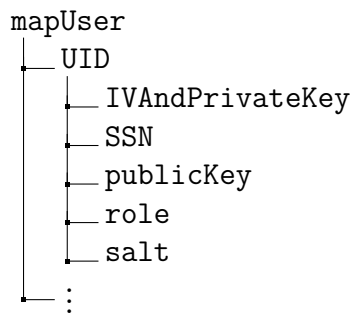


Figure 5.6: The schema of database tree `mapUser`.

Opposite to `mapUser`, there is also a database tree for mapping user SSN to UID. The tree mapping SSN to UID is called `mapUserSNNtoUID`. The schema of `mapUserSNNtoUID` is given in Figure 5.7.

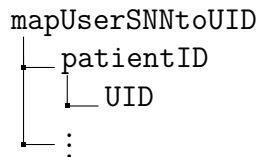


Figure 5.7: The schema of database tree `mapUserSNNtoUID`.

Two other trees in the Realtime Database are `Doctors` and `Patients`. Since they have the same schema, these will, for the sake of brevity, be jointly denoted as `Users`. `Users` contains users' names and contact details, including address, phone number and email. The schema of `Users` is given in Figure 5.8. Objective (F)(ii) is solved by connecting functionality for reading and updating `Patients` to the UI described in Section 5.2.2.

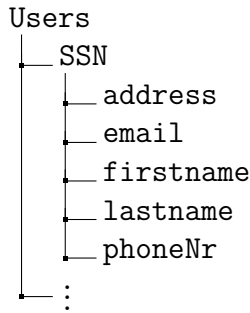


Figure 5.8: The schema of database tree `Users`.

Finally, the Realtime Database contains the trees `PatientToRecordKey` and `DoctorToRecordKey`. The schema of these two trees differ but they essentially contain the same thing; a symmetric key denoted `record_key`, encrypted for a given user. Before the patient has granted access to any medical personnel, only the patient’s encrypted version of `record_key` exists in the database. The patient’s version exists in `PatientToRecordKey`, which schema is given in Figure 5.9. The versions used by medical personnel are stored in `DoctorToRecordKey`, which schema is given in Figure 5.10.

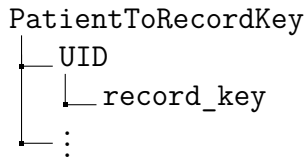


Figure 5.9: The schema of database tree `PatientToRecordKey`.

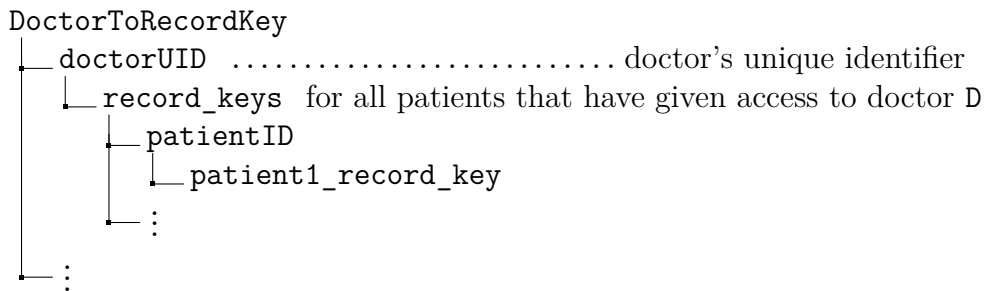


Figure 5.10: The schema of database tree `DoctorToRecordKey`, with clarifying comments given in non-bold text.

5.3.2 Cryptographic Setup

The application makes use of both public- and private-key cryptography. Thus, each user needs to have their own public/private key-pair. The public and private keys are created externally from the application and stored in the database. The public key is stored in plaintext, whereas the private key is encrypted with a key `tmp_key`

before being stored in the database. `tmp_key` is a symmetric key derived from the user's password upon logging in, by passing the password and a user-associated salt through the key derivation function PBKDF2. Upon derivation, `tmp_key` is used to decrypt the user's private key, which is then stored in runtime for use within the application.

Private-key cryptography is used in the application for encrypting and decrypting EHRs. Each user is associated with a symmetric 256 bit-key, denoted `record_key`. The `record_key` is used when both uploading and downloading the EHR content of a patient. Thus, each actor that should have access to the patient's EHR must be able to access and use the `record_key`. This is what public-key cryptography is used for. A `record_key` encrypted and stored for a user `U` in either of the two database trees, `DoctorToRecordKey` or `PatientToRecordKey`, will be encrypted with `U`'s public key before it is stored in the database. Sections 5.3.4, 5.3.5 and 5.3.6 detail the application flow and give more information on how the cryptographic scheme is used throughout the system.

5.3.3 Distributed Data Storage

`Web3.Storage` is used for persistent and distributed storage of patients' EHRs. Upon submitting a new EHR entry, the entry is added to the set of previous entries within the patient's EHR. The resulting new EHR is then uploaded to IPFS via `Web3.Storage` and the CID returned from this operation is recorded on the blockchain to uniquely reference the patient's EHR archive for future access. Sections 5.3.4, 5.3.5 and 5.3.6 gives further details on how `Web3.Storage` and the blockchain are used and fit into the application.

5.3.4 Downloading and Displaying EHRs

When a user enters the overview screen of a given patient, the CID associated with the patient is retrieved from the blockchain. The CID is then used to download all files constituting the patient's EHR from IPFS. The content of each file consists of the encrypted EHR entry contents along with an IV and a MAC. To be able to display the content of each entry in the frontend, a number of operations are carried out:

- (i) The `record_key` encrypted for the given user is retrieved from the database and decrypted using the user's private key.
- (ii) For each file, the content is split into a tuple (`MAC`, `IV`, `file_content`).
- (iii) The decrypted `record_key` is used as input along with the IV for decrypting the `file_content`. Once the content is decrypted, the MAC is verified and if successful, the decrypted content is then displayed in the frontend.

The operations described above work to fulfil Objective (D)(ii), together with the operations for access control described in Sections 5.2.3 and 5.4.2.

5.3.5 Updating and Uploading EHRs

When a medical personnel submits a new EHR entry for a patient, a new EHR archive is created on IPFS and a new CID to reference this archive is recorded on the blockchain. Submitting a new entry is thus, in the project’s terminology, equivalent to issuing an *EHR update*. A series of operations are performed when issuing such an update:

- (i) The entry’s content is parsed from the frontend and encrypted using AES in GCM mode. This is done with the entry as the plaintext input and the previously decrypted `record_key` as the symmetric key input. A new IV is generated before encryption is performed. The generated IV, as well as the resulting encrypted contents and MAC are parsed into a string of the form `MAC||IV||encrypted_file_content` and piped into a `File` object.
- (ii) The `File` object is together with the patient’s previous EHR content, which is encrypted since before, uploaded to a new IPFS archive via `Web3.Storage`. This operation returns a CID that uniquely references the IPFS archive.
- (iii) The CID of the IPFS archive is recorded on the blockchain together with information on which medical personnel it was who issued the EHR update, which healthcare institution the personnel works at and the ID of the given patient. The patient ID used on the blockchain is the hash of the patient’s SSN, produced with the hash function *SHA-256*. The ID of the medical personnel is an arbitrary SSN-like sequence, uniquely identifying each personnel but not tied to the person’s actual SSN.

The above described operations provide the functionality for updating and uploading EHRs to solve Objective (B), together with the previously described cryptographic setup and database structure, as well as Objective (E)(ii).

5.3.6 Handling Regions and Permissions

Retrieving the list of all regions and the list of permissioned regions for a given patient is rather straightforward. These two operations simply involve calling the smart contract’s functions `getRegions(·)` and `getPermissionedRegions(·)`, respectively. Updating a patient’s set of permissioned regions however, is a bit more intricate. When the patient `P` submits an update of their permission settings, by either adding or deleting any access permission, the following operations take place:

- (i) `P`’s new list of permissioned regions, `permissionedRegions`, is recorded on the blockchain, using the smart contract function `setPermissions(·)`.
- (ii) For each new region `R_new` in `permissionedRegions` and for each medical personnel `MP` in `R_new`: `record_key` is encrypted into `encrypted_record_key` using `MP`’s public key. The tuple `(P, MP, encrypted_record_key)` is then stored in the database, as explained in Section 5.3.1.
- (iii) For each removed region `R_rem`, that was in the patient’s previous version of `permissionedRegions` but not in the new one, and for each medical personnel `MP` in `R_rem`: the tuple `(P, MP, encrypted_record_key)` is deleted from the database.

The result after all iterations is that all medical personnel in all permissioned regions

have an `encrypted_record_key` available for use, until the patient retracts access permission from the region. The functionality for patients to set access permissions described here solves Objective (G)(ii).

5.4 Smart Contract and Blockchain

A central part of the backend is the smart contract. It is in the smart contract that a major part of the application’s business logic resides. The smart contract allows for storing data on the blockchain and process functions on the EVM. The blockchain serves to store important application data, but it does not store any EHR directly. Instead, the blockchain contains references to where the EHR of a given patient is stored on IPFS. The following subsections detail what data is stored on the blockchain and the functionality provided by the smart contract.

5.4.1 Structs

The smart contract contains several different custom structured data types (structs) and functions for object initialisation and data storage. Table 5.1 gives an overview over each defined struct and its attributes. The structs are used for permanent storage of structured data on the blockchain.

EHR	Region	HealthCareInst	MedicalPersonnel	Patient
healthcareInst medPersonnel patient cid	id name	id name region	id healthcareInst addr	id permissionedRegions addr

Table 5.1: Smart contract structs with their respective attribute names column-wise.

The application involves actors in the form of regions, healthcare institutions, medical personnel and patients. These are all represented as structs on the smart contract. Functions named as `addX(·)` are implemented to provide a way of adding objects of type `X` to the blockchain, for `X` in `{EHR, Region, HealthCareInst, MedicalPersonnel, Patient}`. During development, these functions are only used from within the migrations script, which deploys the smart contract to the blockchain, to populate the blockchain with a few example objects. The `addX(·)` functions could be used by an admin to register new actors into the system, but no admin side of the application has been developed apart from this.

5.4.2 Functions and Modifiers

The smart contract contains a multitude of functions, both internal and public ones. Two public functions include `setPermissions(·)` and `updateEHR(·)`. These are the only two functions that cost Ether to invoke. Access to invoking some public functions is restricted. To which degree depends on the function. This is done using Solidity’s *modifier* functionality. Any function call that does not adhere to the

condition set by the respective modifier is reverted. Table 5.2 gives an overview over all public functions, except those of the form `addX(·)`, and their interface, as well as who has permission to invoke them by the function-specific modifier. The access restrictions set for the functions are part of enforcing the access control throughout the application. In particular, the modifier used for `getEHRcid(·)` helps to solve Objective (D)(ii).

<code>f([, (param:type)]):returnType</code>	Restriction
<code>getRegions():Array<Object></code>	None
<code>getRegionPersonnel():Array<Object></code>	None
<code>getInstitutionName(instId:String):String</code>	None
<code>getHealthCareInstitution(medicalPersonnelId:String):Object</code>	None
<code>hasPermission(patientId:String):bool</code>	None
<code>updateEHR(patientId:String, cid:String):void</code>	PMP
<code>getEHRcid(patientId:String):String</code>	PMP, The patient
<code>setPermissions(patientId:String, regionIds:Array<String>):void</code>	The patient
<code>getPermissionedRegions(patientId:String):Array<String></code>	The patient

Table 5.2: Smart contract function interfaces and access restrictions. The left column gives the function interface as `functionName(param1 : type1, ...) : returnType`. The right column gives the access restriction, with PMP standing for ‘Permissioned Medical Personnel’.

Functions of the form `addX(·)` currently have no restrictions attached to them. This means that anyone in the network could freely add new objects and, with the current implementation, overwrite existing objects. One way of preventing this could be to add an `admin` attribute to the smart contract and initialise this as the account that deploys the smart contract to the blockchain. Functions of the form `addX(·)` could then be restricted to only be callable by the contract admin. This would however mean that only one account is able to add objects to the blockchain. More sophisticated approaches could allow for a selected group of admins to be in charge of registrations instead.

5.5 Accomplished Objectives

Section 1.2 lists the high-level objectives that this project set out to accomplish. From the results described in this chapter, most of these are considered accomplished, but some leave room for more considerable improvements than others. In particular, Objective (B)(iii) is considered accomplished, but as discussed in Section 6.1.2, the security of the system needs thorough evaluation and probably a completely new implementation.

The perhaps most difficult objective to evaluate is Objective (A)(ii), since it is rather subjective and perhaps not something that should be evaluated by developers themselves. However, Objective (A)(ii) is considered accomplished within the scope of this project.

All Objectives are considered accomplished except one, which is considered only partially accomplished, namely Objective (E). As explained in Section 3.2.2, this

Objective was relaxed to only allow medical personnel authorised by a given patient to issue EHR updates for that patient. The reason for this relaxation was to allow for a simpler cryptographic scheme to be used.

6

Discussion

The purpose of the project was to develop an architecture and a minimal viable application for decentralised storage and management of EHRs. This purpose had two main goals; enabling sharing of medical records between healthcare providers and equipping patients with ownership over their own data. The proposed system fulfils these goals, but leaves room for improvements within several aspects, as well as some of the defined objectives not fully solved. This chapter provides discussions on these matters, detailing limitations and possible improvements of the system. It also provides a motivation for the proposed system architecture, in particular why off-chain storage of EHRs was used instead of on-chain storage. Moreover, a motivation for using a centralised database for storing certain application data is given. Finally, a number of ethical aspects of the projects are discussed.

6.1 Limitations and Possible Improvements

This section describes several limitations of the proposed system which have been identified, as well as possible improvements that could be made to the system. These include administrative issues, security aspects, the practical network setup, among others. Smaller issues and improvements not mentioned in this chapter can be found on the Issues-page of the project's Github repository, in [82].

6.1.1 Administration

If the system were to be implemented on a large scale, some central authority would be necessary to handle user registration. As mentioned, the functions for adding actors to the blockchain are currently only called when the smart contract is deployed, and no admin-side UI has been implemented. In a future implementation, these functions could be callable by a group of admins within the central authority via an admin-side UI.

An additional functionality currently missing in the application is the ability to delete data from the system. This needs to be addressed in future versions as well as similar projects. The proposed system has a major drawback with respect to the deletion of data, in that it uses Web3.Storage for EHR storage. Deleting a file from Web3.Storage is not guaranteed to completely delete the file from IPFS, since some node might still have the file pinned [83]. Thus, deletion cannot be guaranteed even if functionality for it was implemented in the application. This may be an issue both ethically and legally.

Related to the issue of data deletion is the issue that the current system has no expiration date on prescriptions and no way of removing previously made diagnoses and prescriptions. Even though deletion from files on IPFS cannot be guaranteed, deletion of diagnoses and prescriptions could be implemented to be reflected in the new EHR archive of a patient, created upon an EHR update.

Another part of the administration that would need to be solved is the question of how the system would be funded. In particular, if the network requires gas fees to be paid upon blockchain interaction, users' Ethereum wallets need to be supplied with Ether. In the case of Sweden, for example, one solution could be to have either a governmental or regional authority act as system administrators and supply patients' Ethereum wallets with Ether regularly for use within the system. Another solution could be to remove the required gas fees to make all blockchain interaction free, if possible both technically and practically with respect to, for example, the network setup and security.

6.1.2 Security

The security methods used in the application were kept simple and have not been evaluated. Given the type of data being handled, a more cryptographically solid scheme would most likely be needed if this system were to be used in production. In particular, the scheme used for key distribution and storage may be both insecure and inefficient. Since patients grant access permission on a regional level, all medical personnel with a system account within one of the permitted regions will get an encrypted version of the patient's `record_key`. The number of medical personnel within one region may be very large, so this might be computationally demanding. Additionally, only a small subset of the medical personnel may at any point interact with the patient's EHR, in which case the current scheme results in a large set of redundant keys being distributed.

Another improvement that could be made with respect to key management, is to regularly rotate `record_keys` and supply patients with new fresh keys. This prevents adversaries from reading new EHR updates by using old compromised keys. Furthermore, symmetric and private keys are currently stored in runtime during the session. This is not secure and needs to be considered if proceeding with development of the application.

With regards to the security of the system, additional efforts should be made to enforce anonymity of patients, to prevent adversaries from performing directed attacks against specific individuals. The current system tries to achieve a level of anonymity by using hashes of patients' SSNs as IDs on the blockchain. The anonymity of patients may however, be subject to unsuspected attacks, especially considering that information about who makes the EHR updates and what regions a patient has authorised are viewable on the blockchain.

The proposed system relies on the public IPFS for storing EHRs and has been developed with a public blockchain in mind. Thus, references to patients' EHR-archives on IPFS are publicly available and possible to access by anyone. However,

the contents of each archive is encrypted using AES with a 256 bits long key. In theory, such an encryption would take many millions of years to crack with brute force [16]. Even though the encryption is theoretically secure, the public storage of EHRs could both be ethically questionable and unsecure if a cryptographic key were to be compromised. This is a major disadvantage of using public storage solutions as in this project. Further research needs to be done on the possibility of using both file systems and blockchain solutions with restricted access, to prevent the security flaws identified here.

Moreover, the use of a public blockchain introduces its own security risks to be aware of, for example the so-called *51% attack*. 51% attacks are unlikely to generate enough reward to be worthwhile to the attacker [34], but this danger should still be taken into consideration in further research.

6.1.3 Consortium Blockchain and Restricted File System

As mentioned in Section 6.1.2, the use of public data storage poses a security risk for the system. It may thus be preferable to use a consortium blockchain, managed by a selected group of organisations, e.g. a group of healthcare providers or external authorities. This could somewhat decrease the risk of external adversaries trying to access patients' data. If patients are to be able to view the blockchain however, to provide transparency to the system, then patients may still access other patients' encrypted data. To prevent this, a custom restricted file system may be used. This would however, require a degree of centralisation of the data storage – of both the blockchain and the file system. This would probably increase the security of the system, but at the cost of transparency and patient empowerment, since the central authorities may misuse their admin access in undesirable ways.

One potential advantage of using a consortium blockchain is that if a governmental or regional authority acts as the system administrator, then the question of funding may be more easily solved, since the rules concerning gas prices may be set by the authority implementing the blockchain.

6.1.4 Logging Read-Operations

The proposed application only logs updates of data, or write-operations, on the blockchain. Calling read-operations to read data from the blockchain is not logged as a transaction. The reasoning behind this is that there will potentially be many more read-operations called than write-operations. Not logging the read-operations is thus a way to prevent cluttering of the blockchain, as well as to not have read-operations be associated with an Ether-cost. However, also logging read-operations would increase the transparency of the system and allow for a more holistic view of the lifecycle of data than the current approach does.

6.1.5 Emergency Solution

The current system does not implement a solution for emergency cases, i.e. allowing medical personnel to access the EHR of a patient who has not granted them permission, even if the patient is unconscious or in a critical condition. This part of the system may need to be built upon, since it may be indispensable in situations when the patient is unable to grant permission.

6.1.6 Snapshots

The proposed system architecture supports the implementation of a snapshot system. Every time a new EHR entry is submitted, this entry is added to the list of previous entries, resulting in the most recent version of the patient's EHR. The CID returned after uploading the modified EHR is recorded on the blockchain. This procedure is repeated every time the patient's EHR is updated, and because of the blockchain's immutability property, a record of all previous versions of the patient's EHR exists. In the current implementation, only the most recently added CID can be accessed via the smart contract; providing a unique reference to the most recent version of a patient's EHR. This functionality can be expanded to provide a system that allows patients to view different iterations of their medical records at any given time. This extension can be made by extending the smart contract functionality to, for example, mapping the patient's ID to a list of all its EHR updates instead of just to its latest EHR update, as it is currently. The UI also needs to be extended to allow the users to interact with this system, letting the patient view the EHR version at a given date. This adds additional transparency by allowing the patient to more easily inspect the exact details of modifications made to their EHR.

6.2 Architectural Motivation

At the initial stages of the project, a system architecture built on on-chain storage of EHRs was investigated. This was, however, finally discarded for the use of off-chain storage on IPFS instead. This architectural decision was mainly based on the scalability limitations of blockchain technology. This section describes different limitations that were considered in coming to the conclusion to use an off-chain storage for EHRs.

New transactions are added within new blocks on the blockchain. Old blocks cannot be modified. Thus, to keep records of old transactions, the blockchain will always grow bigger and bigger. This growth of the blockchain is more rapid if the sizes of transactions are large, as would be the case if storing entire EHRs or even only individual EHR entries. Hence, such a solution would lead to inefficiency issues due to the scalability problem of blockchains.

It was concluded from the initial research and analysis, that the inefficiency problems remain no matter if the entire EHR is stored as a single entity, or if individual EHR entries are stored separate from each other. If EHR entries were to be stored separately, then the EHR data will be fragmented across multiple blocks. To lookup

a patient's EHR would then require an aggregation of multiple individual transactions from potentially many different blocks. This was deemed computationally inefficient.

Due to limitations of the hosts constituting the network, the size of blocks must be limited [30]. If storing all EHR data together, for each individual EHR, then this would mean that at some point the EHR might out-grow the size of a single block. Upon such a scenario, the EHR would need to be fragmented across several new blocks. Hence, this would further clutter the blockchain, more rapidly increasing its size and adding to the inefficiency problems.

Due to the inefficiency problems identified with on-chain EHR storage, combined with inspiration from related work, the proposed system architecture with off-chain storage of EHRs and on-chain content references was decided. Further research and comparisons should however be done between possible alternative system architectures.

6.3 Decentralisation

The proposed application is mostly built on frameworks that are or could be decentralised. As discussed in subsection 6.1.3, full decentralisation of the data storage may not be desirable with respect to security. The current implementation uses centralised storage for some application data, since the database used is provided by Firebase. As such, it is currently both centralised and dependent on a third party. Authentication and storage of cryptographic keys were deemed to be best handled by a central server for security reasons. Additionally, potentially sensitive information such as contact information was also considered more appropriately stored in the database rather than on IPFS or the blockchain. Out-sourcing the database to a third-party as done in this project may however be unadvised and an in-house solution should probably be used.

6.4 Ethics

It is important to consider the ethical aspects of any new development. This is especially critical given the sensitive nature of the data that this application is developed to handle. This section discusses the ethical elements connected to this thesis, focusing on privacy, security, environment and economy.

6.4.1 Privacy and Security Concerns

With the suggested system's core concept being the distribution of patient data across multiple healthcare providers, it is inevitable to question its vulnerabilities, particularly out of privacy concerns. Hence, a system for storing EHRs must be sufficiently secure in order to protect the patients' private data from being exposed, for example by using adequately secure encryption schemes and ensuring the anonymity

of patients. The proposed system’s cryptographic scheme needs to be improved upon in this regard.

Another point of concern in regards to patient privacy is the transparency related to who can actually access the EHRs and to what degree patients can restrict this access. Giving patients the ability to adjust this at will, grants the assurance that they are in control of their sensitive data and privacy. This of course needs to be combined with adequate security measures, as mentioned, to provide both data confidentiality and integrity, so that adversarial modifications of data are noticed. This may also be combined with the use of a consortium blockchain and a file system with restricted access, to prevent adversaries from accessing even encrypted data.

An additional security aspect of the system is its availability. The availability of EHRs may be critical for medical personnel to be able to carry out their job. Thus, it is important that the inefficiencies that may come with using blockchain technology does not hamper the availability of the system.

Another aspect central to the privacy aspect of the system is that of data deletion. The proposed system is lacking in this regard, since even if this functionality would be added, deleting files from Web3.Storage does not guarantee that the files are deleted from IPFS [83]. Additionally, since the files are stored publicly, they could remain for many millions of years, in theory. This could leave enough time for an attacker to brute force the encryption of a patient’s EHR and read its content. Even if it is highly theoretical and presumably long after the patient’s death, this raises ethical concerns regarding for how long a system must ensure data confidentiality and individual privacy.

6.4.2 Environmental Impact

To ensure that no data has been manipulated, many blockchains, including Ethereum, use PoW as consensus mechanism. This introduces an additional energy consumption beyond what is required for storing data. How large of a proportion of the total cost this makes up differs case by case. In Bitcoin’s case, this is argued to be the primary source of its estimated yearly energy consumption of over 120 TWh - surpassing that of many countries [84]. Ethereum has acknowledged that PoW has led to higher energy consumption, so much so in fact, that it currently considers it “too high and unsustainable” [85]. To reduce the negative environmental impact of a system such as this, using alternative consensus mechanisms rather than PoW may thus be needed. For example, Ethereum will soon be moving over to using PoS instead of PoW and estimates that this will reduce Ethereum’s energy consumption by 99.95% [85].

6.4.3 Economical Impact

If the system increases the healthcare providers’ energy consumption due to the Proof-of-work consensus mechanism for example, then this will not only have negative environmental effects, but also lead to higher economic costs for the healthcare providers. Whether or not the energy consumption would go up though is uncertain.

The cost of a system which relies on redundant data could be assumed to be higher than its centralised counterparts. Especially considering that the blockchain as a whole would be copied and distributed many times over. On the other hand, the traditional data storing solutions commonly use backup servers to keep an extra copy of the data in case of a failure at the main server. It is therefore difficult to conclude which is more expensive, as it can for the most part only be speculated on.

The proposed system may have economical advantages due to the facilitation of medical record exchange between healthcare providers. This may have economic benefits for both patients and healthcare providers, as well as society as a whole. The facilitated information exchange may lead to a more efficient healthcare by saving its time and resources. For example, patients may not have to undergo redundant health examinations upon moving. In countries where patients pay for healthcare, this may reward the patients directly, since they could have fewer healthcare appointments to pay for. A more efficient healthcare due to easier information exchange may also lead to healthcare at large becoming cheaper and thus more affordable. In countries with tax-funded healthcare, a more efficient healthcare may also require less governmental funding.

7

Conclusions

Several actors within the healthcare industry have raised concerns about how EHRs are stored and managed. The current centralised model has shown its limitations when it comes to fragmented medical records, difficult information exchange between healthcare providers, data integrity, and data ownership. In the face of these challenges, blockchain technology has been presented as a feasible alternative to the traditional server architecture.

This project aimed to develop a minimal viable application for decentralised storage and management of EHRs. The proposed system sought to facilitate sharing of medical records between healthcare providers and to empower patients to claim ownership of their data.

The system architecture presented, focused on decentralisation using blockchain technology. The main goals were reached by developing a layered architecture and utilising decentralised frameworks. By choosing Web3.Storage as an off-chain solution to store the actual EHRs files, blockchain scalability limitations and IPFS' data persistence issues were overcome. An Ethereum blockchain was used for storing references to EHR storage locations on IPFS. A smart contract was implemented to primarily provide the logic for the storage of these references and access permissions set by each patient.

Even though most of the main objectives of the project were fulfilled, there is room for improvement when it comes to several aspects of the proposed system. One important improvement identified is the need for a more solid cryptographic scheme to ensure both confidentiality and integrity of data, as well as ensuring the anonymity of patients. The proposed system does not implement a solution for facilitating access to patients' EHRs from an unauthorised region in emergency cases. This part of the system may need to be built upon, given that it may be indispensable in life-threatening situations when the patient may be unconscious and unable to grant permission. The snapshot functionality can also be extended to provide a system that allows patients to view different iterations of their medical records at any given time. Further research should also be done regarding what type of blockchain and distributed storage to use. A consortium blockchain combined with restricted access to the file storage would present potential advantages, but it would introduce a certain degree of centralisation. A thorough review of alternative network solutions should be considered for future work.

Bibliography

- [1] T. I. C. OpenText. “The history of health information management – from then to now.” (May 2, 2017), [Online]. Available: <https://blogs.opentext.com/history-health-information-management-no/> (visited on 02/03/2022).
- [2] R. S. Evans. “Electronic health records: Then, now, and in the future.” (May 20, 2016), [Online]. Available: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5171496/> (visited on 02/03/2022).
- [3] C. Kruse, A. Stein, and H. Thomas. “The use of electronic health records to support population health: A systematic review of the literature.” (Sep. 29, 2018), [Online]. Available: <https://link.springer.com/article/10.1007/s10916-018-1075-6> (visited on 02/03/2022).
- [4] Thimmaiah, D. S, D. Nayak, D. B. B, and G. H. L. “Decentralized electronic medical records.” (Mar. 2019), [Online]. Available: <https://www.ijrar.org/papers/IJRAR1ADP055.pdf> (visited on 02/03/2022).
- [5] J. BUTCHER. “Data is the new oil of the 21st century.” (Dec. 18, 2021), [Online]. Available: <https://blog.s4rb.com/data-is-the-oil-of-the-21st-century> (visited on 02/08/2022).
- [6] T. I. TEAM. “How do internet companies profit with free services?” (Jul. 29, 2021), [Online]. Available: <https://www.investopedia.com/ask/answers/040215/how-do-internet-companies-profit-if-they-give-away-their-services-free.asp> (visited on 02/08/2022).
- [7] B. Sam Daley. “How using blockchain in healthcare is reviving the industry’s capabilities.” (Jul. 30, 2021), [Online]. Available: <https://builtin.com/blockchain/blockchain-healthcare-applications-companies> (visited on 02/08/2022).
- [8] T. Mcghin, K. Choo, C. Liu, and D. He. (2019), [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1084804519300864#bib10> (visited on 04/25/2022).
- [9] S. Cao, G. Zhang, P. Liu, X. Zhang, and F. Neri. (2019), [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0020025519301471> (visited on 04/25/2022).
- [10] L. Chen, W. Lee, C. Chang, K. Choo, and N. Zhang. (2019), [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0167739X18314134> (visited on 04/25/2022).
- [11] D. Nguyen, P. Pathirana, M. Ding, and A. Seneviratne. (2019), [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/8717579> (visited on 04/25/2022).

- [12] O.Demir and B.Kocak. (2019), [Online]. Available: https://www.researchgate.net/publication/335232056_A_Decentralized_File_Sharing_Framework_for_Sensitive_Data (visited on 04/25/2022).
- [13] M. Misbhauddin, A. AlAbdulatheam, M. Aloufi, H. Al-Hajji, and A. Al-Ghuwainem. (2020), [Online]. Available: <https://ieeexplore.ieee.org/document/9257720/authors> (visited on 04/18/2022).
- [14] I. doc. “Persistence, permanence, and pinning.” (2022), [Online]. Available: <https://docs.ipfs.io/concepts/persistence/#persistence-versus-permanence> (visited on 04/18/2022).
- [15] Web3.Storage. “Decentralized storage in brief.” (n.d.), [Online]. Available: <https://web3.storage/docs/concepts/decentralized-storage> (visited on 04/14/2022).
- [16] W. Stallings, *Cryptography and network security : principles and practice*. Pearson, 2017, ISBN: 9781292158587. [Online]. Available: <https://search.ebscohost.com/login.aspx?direct=true&db=cat07470a&AN=clc.519d5579.642d.4186.9877.a7a76096e652&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds>.
- [17] D. Boneh and V. Shoup, *A Graduate Course in Applied Cryptography*, English. Jan. 2020. [Online]. Available: https://crypto.stanford.edu/~dabo/cryptobook/BonehShoup_0_5.pdf.
- [18] J. Kleinberg and É. Tardos, *Algorithm design. [electronic resource]*. Pearson, 2014, ISBN: 9781292037042. [Online]. Available: <https://search.ebscohost.com/login.aspx?direct=true&db=cat07472a&AN=clec.DAWVLE26905510&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds>.
- [19] Unknown. “A graduate course in applied cryptography.” (Mar. 21, 2018), [Online]. Available: <https://www.njordlaw.com/handwritten-signatures-vs-e-signatures> (visited on 04/20/2022).
- [20] —, “What is blockchain?” (N.d.), [Online]. Available: <https://www.euromoney.com/learning/blockchain-explained/what-is-blockchain> (visited on 02/03/2022).
- [21] Z. Zheng, S. Xie, H.-N. Dai, X. Chen, and H. Wang, “An overview of blockchain technology: Architecture, consensus, and future trends,” Jun. 2017. DOI: [10.1109/BigDataCongress.2017.85](https://doi.org/10.1109/BigDataCongress.2017.85).
- [22] G. WOOD. “Ethereum: A secure decentralised generalised transaction ledger.” (n.d.), [Online]. Available: <https://ethereum.github.io/yellowpaper/paper.pdf> (visited on 02/03/2022).
- [23] Unknown. “How does a transaction get into the blockchain?” (N.d.), [Online]. Available: <https://www.euromoney.com/learning/blockchain-explained/how-transactions-get-into-the-blockchain> (visited on 02/03/2022).
- [24] J. Frankenfield. “Consensus mechanism (cryptocurrency).” (Nov. 30, 2021), [Online]. Available: <https://www.investopedia.com/terms/c/consensus-mechanism-cryptocurrency.asp> (visited on 02/11/2022).
- [25] I. Eyal and E. Sirer, “Majority is not enough: Bitcoin mining is vulnerable,” vol. 8437, Nov. 2013. DOI: [10.1007/978-3-662-45472-5_28](https://doi.org/10.1007/978-3-662-45472-5_28).

- [26] Ethereum. (n.d.), [Online]. Available: <https://ethereum.org/en/developers/docs/intro-to-ether/> (visited on 04/25/2022).
- [27] @. (github). “Proof-of-stake (pos).” (Jan. 26, 2022), [Online]. Available: <https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/> (visited on 02/08/2022).
- [28] V. Buterin. “On public and private blockchains.” (Aug. 7, 2015), [Online]. Available: <https://blog.ethereum.org/2015/08/07/on-public-and-private-blockchains/> (visited on 04/11/2022).
- [29] V. Buterin. “Why sharding is great: Demystifying the technical properties.” (Apr. 7, 2021), [Online]. Available: <https://vitalik.ca/general/2021/04/07/sharding.html> (visited on 04/13/2022).
- [30] —, “The limits to blockchain scalability.” (May 23, 2021), [Online]. Available: <https://vitalik.ca/general/2021/05/23/scaling.html> (visited on 04/13/2022).
- [31] D. Khan, T. Low, and M. Hashmani, “Systematic literature review of challenges in blockchain scalability,” *Applied Sciences*, vol. 11, p. 9372, Oct. 2021. DOI: [10.3390/app11209372](https://doi.org/10.3390/app11209372).
- [32] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, “Hawk: The blockchain model of cryptography and privacy-preserving smart contracts,” in *2016 IEEE Symposium on Security and Privacy (SP)*, 2016, pp. 839–858. DOI: [10.1109/SP.2016.55](https://doi.org/10.1109/SP.2016.55).
- [33] A. Biryukov, D. Khovratovich, and I. Pustogarov, “Deanonymisation of clients in bitcoin p2p network,” in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS ’14, Scottsdale, Arizona, USA: Association for Computing Machinery, 2014, pp. 15–29, ISBN: 9781450329576. DOI: [10.1145/2660267.2660379](https://doi.org/10.1145/2660267.2660379). [Online]. Available: <https://doi.org/10.1145/2660267.2660379>.
- [34] J. A. Kroll, I. C. Davey, and E. W. Felten, “The economics of bitcoin mining, or bitcoin in the presence of adversaries,” 2013.
- [35] C. Ye, G. Li, H. Cai, Y. Gu, and A. Fukuda, “Analysis of security in blockchain: Case study in 51%-attack detecting,” in *2018 5th International Conference on Dependable Systems and Their Applications (DSA)*, 2018, pp. 15–24. DOI: [10.1109/DSA.2018.00015](https://doi.org/10.1109/DSA.2018.00015).
- [36] Unknown. “Intotoether.” (Apr. 5, 2022), [Online]. Available: <https://ethereum.org/en/developers/docs/intro-to-ether/> (visited on 04/16/2022).
- [37] Shinobi. “Introduction to smart contracts.” (Nov. 21, 2021), [Online]. Available: <https://bitcoinmagazine.com/technical/why-bitcoin-smart-contract-platform> (visited on 01/31/2022).
- [38] M. contributors. “Consensus mechanisms.” (Jan. 29, 2022), [Online]. Available: <https://ethereum.org/en/developers/docs/consensus-mechanisms/> (visited on 04/19/2022).
- [39] Unknown. “Introduction to smart contracts.” (Jan. 10, 2022), [Online]. Available: <https://ethereum.org/en/developers/docs/smart-contracts/> (visited on 01/31/2022).
- [40] J. Frankenfield. “Gas and fees.” (Mar. 9, 2021), [Online]. Available: <https://ethereum.org/en/developers/docs/gas/> (visited on 04/16/2022).

- [41] K. Choi. “Transaction dropped and replaced?” (2020), [Online]. Available: <https://info.etherscan.com/transaction-dropped-replaced/> (visited on 04/19/2022).
- [42] Unknown. “Expressions and control structures.” (n.d.), [Online]. Available: <https://docs.soliditylang.org/en/v0.8.13/control-structures.html#error-handling-assert-require-revert-and-exceptions> (visited on 04/19/2022).
- [43] M. Support. “Why did i pay a fee for a failed transaction?” (Jan. 2022), [Online]. Available: <https://metamask.zendesk.com/hc/en-us/articles/360045439051-Why-did-I-pay-gas-fees-for-a-failed-transaction-> (visited on 04/19/2022).
- [44] G. McCubbin. “How to build blockchain app - ethereum todo list 2019.” (Apr. 20, 2022), [Online]. Available: <https://www.dappuniversity.com/articles/blockchain-app-tutorial> (visited on 05/12/2022).
- [45] B. Nibley. “What is the ethereum virtual machine? how does it work?” (Nov. 4, 2021), [Online]. Available: <https://www.sofi.com/learn/content/what-is-ethereum-virtual-machine/> (visited on 02/04/2022).
- [46] m. c. Unknown. “Ethereum virtual machine (evm).” (Apr. 11, 2022), [Online]. Available: <https://ethereum.org/en/developers/docs/evm/> (visited on 04/16/2022).
- [47] —, “Networks.” (Mar. 28, 2022), [Online]. Available: <https://ethereum.org/en/developers/docs/networks/> (visited on 04/16/2022).
- [48] IPFS. “Ipfs powers the distributed web.” (n.d.), [Online]. Available: <https://ipfs.io/> (visited on 02/10/2022).
- [49] —, “Ecosystem directory.” (n.d.), [Online]. Available: <https://ecosystem.ipfs.io/> (visited on 02/10/2022).
- [50] —, “How ipfs works: Directed acyclic graphs (dags).” (n.d.), [Online]. Available: <https://docs.ipfs.io/concepts/how-ipfs-works/#directed-acyclic-graphs-dags> (visited on 05/12/2022).
- [51] —, “Persistence, permanence, and pinning.” (n.d.), [Online]. Available: <https://docs.ipfs.io/concepts/persistence/#pinning-services> (visited on 04/19/2022).
- [52] M. Steichen, B. Fiz, R. Norvill, W. Shbair, and R. State, “Blockchain-based, decentralized access control for ipfs,” in *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, 2018, pp. 1499–1506. DOI: [10.1109/Cybermatics_2018.2018.00253](https://doi.org/10.1109/Cybermatics_2018.2018.00253).
- [53] Filecoin. “What is filecoin.” (n.d.), [Online]. Available: <https://docs.filecoin.io/about-filecoin/what-is-filecoin> (visited on 04/18/2022).
- [54] Web3.Storage. “Better storage. better transfers. better internet.” (2022), [Online]. Available: <https://docs.web3.storage/> (visited on 02/02/2022).
- [55] J. Eberhardt and S. Tai. “On or off the blockchain? insights on off-chaining computation and data.” (Sep. 1, 2017), [Online]. Available: https://link.springer.com/content/pdf/10.1007/978-3-319-67262-5_1.pdf (visited on 02/11/2022).

- [56] EtherScan. (n.d.), [Online]. Available: <https://etherscan.io/> (visited on 04/21/2022).
- [57] cprime. “What is agile? what is scrum?” (N.d.), [Online]. Available: <https://www.cprime.com/resources/what-is-agile-what-is-scrum/> (visited on 02/02/2022).
- [58] C. Drumond. “Scrum.” (n.d.), [Online]. Available: <https://www.atlassian.com/agile/scrum> (visited on 02/02/2022).
- [59] Trello. (n.d.), [Online]. Available: <https://trello.com/en> (visited on 04/22/2022).
- [60] GitHub. “Github issues: Project planning for developers.” (n.d.), [Online]. Available: <https://github.com/features/issues> (visited on 04/22/2022).
- [61] N. Dubien. “Github repository for fast-check.” (n.d.), [Online]. Available: <https://github.com/dubzzz/fast-check> (visited on 04/25/2022).
- [62] GitHub. “Github actions: Automate your workflow from idea to production.” (n.d.), [Online]. Available: <https://github.com/features/actions> (visited on 04/26/2022).
- [63] W3Techs. “Usage statistics of javascript as client-side programming language on websites.” (Feb. 8, 2022), [Online]. Available: <https://w3techs.com/technologies/details/cp-javascript/> (visited on 02/08/2022).
- [64] T. Podmanicki. “How to enable javascript in your browser.” (Apr. 6, 2022), [Online]. Available: <https://www.enable-javascript.com/> (visited on 04/06/2022).
- [65] React.org. “React - a javascript library for building user interfaces.” (n.d.), [Online]. Available: <https://reactjs.org/> (visited on 02/08/2022).
- [66] Meta. “React | meta open source.” (n.d.), [Online]. Available: <https://opensource.fb.com/projects/react> (visited on 02/08/2022).
- [67] J. Preece, H. Sharp, and Y. Rogers, “Interaction design: Beyond human-computer interaction,” in Wiley, 2015, pp. 65–84, 158–198, ISBN: 978-1-119-02075-2.
- [68] Figma. (n.d.), [Online]. Available: <https://www.figma.com/ui-design-tool/> (visited on 04/22/2022).
- [69] Node.js. “Introduction to node.js.” (2022), [Online]. Available: <https://nodejs.dev/learn/introduction-to-nodejs> (visited on 02/02/2022).
- [70] Unknown. “Node.js v17.9.0 documentation.” (n.d.), [Online]. Available: <https://nodejs.org/api/crypto.html> (visited on 04/19/2022).
- [71] O. B. Samson. “What is web3.js.” (Mar. 11, 2022), [Online]. Available: <https://web3.hashnode.com/what-is-web3js-an-introduction-into-the-web3js-libraries> (visited on 04/19/2022).
- [72] L. Rosencrance. “Google firebase.” (Apr. 2019), [Online]. Available: <https://www.techtarget.com/searchmobilecomputing/definition/Google-Firebase> (visited on 04/18/2022).
- [73] Firebase. “Firebase realtime database: Store and sync data in real time.” (n.d.), [Online]. Available: <https://firebase.google.com/products/realtime-database> (visited on 04/26/2022).

- [74] —, “Firebase authentication: Simple, no-cost multi-platform sign-in.” (n.d.), [Online]. Available: <https://firebase.google.com/products/auth> (visited on 05/12/2022).
- [75] T. Suite. “Truffle overview.” (n.d.), [Online]. Available: <https://trufflesuite.com/docs/truffle/> (visited on 04/18/2022).
- [76] Truffle Suite. “Overview.” (n.d.), [Online]. Available: <https://trufflesuite.com/docs/ganache/> (visited on 02/02/2022).
- [77] Ethereum. “Solidity.” (2021), [Online]. Available: <https://docs.soliditylang.org/en/v0.8.11/> (visited on 02/02/2022).
- [78] (N.d.), [Online]. Available: <https://metamask.io/> (visited on 05/05/2022).
- [79] MetaMask. “Introduction.” (Jan. 14, 2022), [Online]. Available: <https://docs.metamask.io/guide/> (visited on 05/05/2022).
- [80] Unknown. “Getting started.” (n.d.), [Online]. Available: <https://docs.metamask.io/guide/getting-started.html#basic-considerations> (visited on 05/05/2022).
- [81] (N.d.), [Online]. Available: <https://metamask.io/download/> (visited on 04/18/2022).
- [82] D. Zamanian, N. Johnsson, W. Pau, E. I. Galan, H. Jernkrook, and C. Molin. (n.d.), [Online]. Available: <https://github.com/DavidZamanian/Bachelor-thesis-blockchain-for-medical-records>.
- [83] Web3Storage. (n.d.), [Online]. Available: <https://web3.storage/faq/> (visited on 04/23/2022).
- [84] D. A. P. at Cambridge Centre for Alternative Finance. “Cambridge bitcoin electricity consumption index.” (Feb. 1, 2022), [Online]. Available: <https://ccaf.io/cbeci/index> (visited on 02/02/2022).
- [85] Ethereum.org. “Ethereum energy consumption.” (Feb. 1, 2022), [Online]. Available: <https://ethereum.org/en/energy-consumption/> (visited on 02/02/2022).