Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

Jens-Andreas Hanssen Rensaa

VerifyMed - Application of blockchain technology to improve trust in virtualized healthcare services

Master's thesis in Communication Technology

May 2020

NTNU
Kunnskap for en bedre verden

NTNU – Trondheim
Norwegian University of
Science and Technology

VerifyMed — Application of blockchain technology to improve trust in virtualized healthcare services

Jens-Andreas Hanssen Rensaa

Submission date: May 2020
Responsible professor: Prof. Danilo Gligoroski, IIK
Supervisor: Assoc. Prof. Katina Kralevska, IIK

Norwegian University of Science and Technology
Department of Information Security and Communication Technology

Title: VerifyMed — Application of blockchain technology to improve trust in virtualized healthcare services

Student: Jens-Andreas Hanssen Rensaa

Problem description:

Innovations within information and communication technology have changed the healthcare industry. Patients living in a digitized world can now interact with the healthcare system through online services. Patients can use these services to talk with medical professionals directly through chat applications, video conferencing or indirectly through consulting services. These applications surface some fundamental problems. Patients do not have any way to confirm that the person they are interacting with is actually a medical professional. Furthermore, they have no way to validate that the person has competence within the field in question.

This thesis aims to create and evaluate a proof-of-concept application for transparently validating the authorization and competence of medical professionals using blockchain technology. Such an application must be able to capture the trust relationships within the healthcare industry to validate the medical authorization of the person claiming it. Furthermore, we aim to capture the competence through storing metrics from outcomes reported by patients. We use design science to formally iterate through the process of capturing validation criteria, developing the technological artifact, and evaluating it.

We aim to implement this application using technology compatible with a large public blockchain. The application will be developed through using the Ethereum blockchain and by using open-source tools which are publicly available.

Responsible professor: Prof. Danilo Gligoroski, IIK

Supervisor: Assoc. Prof. Katina Kralevska, IIK

Abstract

The healthcare industry is moving towards increased usage of virtualized healthcare services. By using these, patients can interact with healthcare workers via online applications, such as video conferencing or chat. When patients first meet a healthcare worker through this setting, they must trust that the healthcare worker has the authority and competence to deal with their health problems. However, the patients do not have any tool to confirm that this is actually the case.

In this thesis, we design, implement and evaluate VerifyMed — A proof-of-concept platform built on the Ethereum blockchain. Our platform models trust relationships within the healthcare industry to validate professional clinical authorization. Furthermore, it enables a healthcare professional to build a portfolio of real-life work experience and further validates their competence by storing outcome metrics reported by the patients. By using VerifyMed, patients can verify the authorization, experience, and competence of a healthcare worker in a transparent, trust-less, and non-repudiable manner.

Our results show that the VerifyMed platform can address the problem of providing trust in healthcare at a moderate cost. However, throughput is limited by the Ethereum blockchain. Further work on alternative solutions is required to address these problems.

Sammendrag

Helsevesenet går mot økt bruk av digitale helsetjenester. Ved å bruke slike tjenester kan pasienter møte helsepersonell ved bruk av nettbaserte applikasjoner, for eksempel via digitale konsultasjoner på video eller chat. Når en pasient først møter helsepersonell gjennom slike tjenester, så må de stole på at helsepersonellet har autorisasjon og kompetanse til å håndtere deres helseplager. Pasienten har imidlertid ikke noe verktøy for å bekrefte at dette faktisk er tilfelle.

I denne masteroppgaven designer, implementerer og evaluerer vi VerifyMed — En prototype plattform bygget på blokkjeden Ethereum. I plattformen modellerer vi tillitsforhold innad i helsevesenet for å validere at autorisasjon eksisterer. Videre lar plattformen helsepersonell bygge en portefølje av ekte arbeidserfaring, og validerer kompetansen deres ved å lagre pasientrapporterte utfallsmål. Ved å bruke VerifyMed kan pasienter verifisere autorisasjon, erfaring og kompetanse hos helsepersonell på en transparent, tillitsfri og ikke-avviselig måte.

Resultatene våre viser at VerifyMed-plattformen kan benyttes for a skape tillit til helsepersonell, og dette med en moderat kostnad. Gjennomløpsraten er imidlertid begrenset av Ethereum blokkjeden. Det kreves derfor videre arbeid med alternative løsninger for å løse disse problemene.

Preface

This Master's Thesis concludes my Master of Science Degree in Communication Technology with specialization in Information Security at the Department of Information Security and Communication Technology (IIK) at the Norwegian University of Science and Technology (NTNU) in Trondheim. It was completed during spring 2020, and builds on a pre-project conducted during fall 2019.

Acknowledgements

Research is never a solitary task. This thesis would not have been possible without the invaluable input from experts within both the technical and healthcare domains. I would like to express my gratitude to Katina Kralevska, my supervisor, and Danilo Gligoroski, my responsible professor. They have both provided invaluable guidance and feedback during my work. I want to give a special thanks to Anton Hasselgren, who has provided immense insight on blockchain in healthcare and has participated beyond expectations in my thesis. I would also like to thank Arild Faxvaag for his excellent understanding of the healthcare domain.

I would like to give a special thanks to my family, who have provided their unbounded love and support during my education. A thank you is also due to all the people who have helped make my five years Trondheim unforgettable.

Although despised for shutting down society, I would like to acknowledge COVID-19 for dramatically increasing the relevance of virtualized healthcare platforms.

Contents

List of Figures

List of Tables

List of Symbols

$H(a) = b$	A cryptographic hash function with variable length input a and fixed length output b.
$KEC(a)$	The cryptographic hash function Keccak-256 used over a message a.
$RLP(x)$	Recursive Length Prefix encoding, for creating a serialization of the binary data x.
T	A complete blockchain transaction composed of both T_d and T_s.
T_d	The data component of a blockchain transaction.
T_s	The signature component of a blockchain transaction.

List of Acronyms

ABI Application Binary Interface.

API Application Programming Interface.

CLI Command Line Interface.

DAC Discretionary Access Control.

dApp Decentralized Application.

DSA Digital Signature Algorithm.

ECDSA Elliptic Curve Digital Signature Algorithm.

EHR Electronic Health Records.

ETH Ether.

EVM Ethereum Virtual Machine.

NTNU Norwegian University of Science and Technology.

PHR Personal Health Records.

PREM Patient Reported Experience Measures.

PRO Patient Reported Outcomes.

PROM Patient Reported Outcome Measures.

RBAC Role-Based Access Control.

RLP Recursive Length Prefix.

SHA Secure Hash Algorithm.

UI User Interface.

Chapter 1
Introduction

1.1 Motivation

The healthcare sector is moving towards an increasingly virtualized world. Inspired by the Industry 4.0 initiative, Healthcare 4.0 [3] is a similar strategic concept for the healthcare domain. The aim of Healthcare 4.0 is to use modern technology such as next-generation networking, Artificial Intelligence (AI), and cloud to enable a virtualized healthcare environment where personalization is delivered in real-time for patients. A vital part of this process is a strong shift towards virtualized healthcare services, allowing routine tasks to either be automated or for patients to perform these tasks by themselves. Healthcare 4.0 will allow patients to access the healthcare system in more ways than before, resulting in a better and more accessible care.

Healthcare 4.0 aims to position the healthcare industry to tackle the socio-economic and demographic changes in the future. An increasingly elderly population will require the healthcare industry to be more efficient. Furthermore, socio-economic changes and globalization have led to trends such as an increased demand for specialized care, a strengthened patient choice of healthcare services, and an increased accessibility of healthcare services across borders and jurisdictions.

The changes within the healthcare industry will lead to some emerging problems. The critical problem we choose to focus on is providing trust within a virtualized healthcare environment. In the current European landscape, most patients physically encounter healthcare workers within authorized and well-known healthcare institutions. There is an inherent trust relationship from the patient to the healthcare worker rooted in this setting [4]. However, when meeting a healthcare worker within a virtualized environment, this inherent trust relationship is not present. Furthermore, building up such a trust relationship may be hard if the caregiver is an AI healthcare worker. The healthcare domain, therefore, needs new solutions for enabling trust to be established between patients and healthcare workers in a virtualized environment.

Blockchain [5] is a maturing technology with properties enabling it to provide trust within a virtualized healthcare domain. We can describe blockchain technology as an enabler of distributed platforms where data is stored in an immutable and fully distributed manner across organizations. The blockchain is maintained by self-organizing computers, which automatically coordinate themselves in a trustless environment to deliver this service. While the scope of blockchain traditionally was bound to the financial domain, Blockchain 2.0 [6] allows users to store arbitrary data and models on the blockchain through the means of smart contracts. Through creating and deploying smart contracts, we can build models that capture the authorization, experience, and competence of a healthcare worker directly on the blockchain. These models can then be used by patients to establish a trust relationship with the healthcare worker.

1.2 Goal and Methodology

The primary goal of this thesis is to design, implement, and evaluate a proof-of-concept application for transparently validating the authorization and competence of healthcare workers by using the blockchain technology. The thesis aims to do exploratory research into this problem and how to apply blockchain technology to solve it. The end-product itself may serve as a sound basis for further research or real-world implementation.

Our method is rooted in the principles of design science [7] and inspired by requirements engineering [8]. Figure 1.1 shows the iterative process, where we go from defining requirements to producing results. We first define some requirements based on our initial understanding of the problem. We work with a interdisciplinary team with backgrounds from both the technical and healthcare side. This setting allows us to validate if the requirements correspond to real-world problems and regulatory requirements. Next, we use our requirements to create an architectural model. We use the architecture to implement a proof-of-concept application, which is the main *artifact* produced. Finally, we test the application and evaluate it based on our requirements and collected metrics. If we ever find an assumption to be wrong, or if the process reveals unexpected problems, we go one step back to the previous step in our process.

The most critical outputs from our method are the observations of the artifact and metrics. The observation covers the artifacts' ability to solve the problem of trust in a virtualized healthcare environment. We will evaluate the artifact in the context of the defined requirements and other social needs. The metrics, on the other hand, are objective measures that we use for evaluating the artifact against other solutions. We collect these metrics from measurements during system testing. The most critical metrics of our interest are:

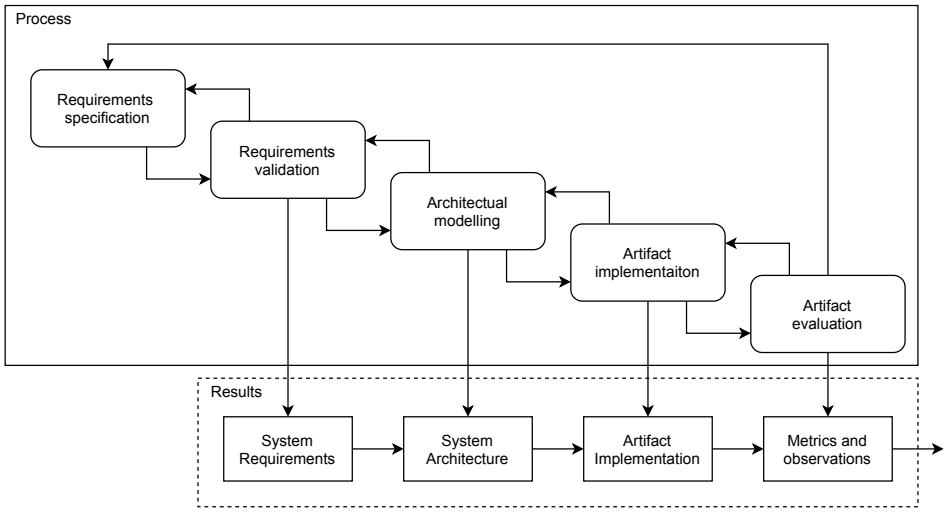


Figure 1.1: This figure shows a visualization of the overall method applied in this thesis. We iterate through a process where we define requirements, validate these, perform architectural modeling, implement the artifact, and finally evaluate it. Each of these steps creates outputs, forming our primary results.

– Cost: Public blockchain transactions cost an amount of cryptocurrency with monetary value. The cost scales with the complexity of the smart contract.

– Throughput: The maximum possible data rate which can be handled by the system.

1.3 Contribution

This thesis is the product from exploring the problems of trust in the healthcare domain and implementing a proof-of-concept application by using the Ethereum blockchain. Our main contribution is the proof-of-concept application itself, showing how blockchain technology can solve real problems in the healthcare domain. The application is released as open-source software and is freely available on GitHub[1]. Furthermore, we provide requirements, architectural models, and metrics for the application. Finally, we provide an assessment of our application. We uncover the conditions in which blockchain is suitable for the healthcare domain and provide suggestions for further applications of the technology.

[1]https://github.com/jarensaa/transparent-healthcare, commit *34c09d9*

In addition to the thesis, the work has also resulted in two papers:

- One technical paper on the VerifyMed architecture and implementation [9], accepted for presentation at the *2nd Blockchain and Internet of Things Conference (BIOTC 2020)*, to be held in Singapore from 8th to 10th of July 2020. The complete paper is attached in Appendix D.
- One work in progress paper on blockchain for trust in healthcare [10], to be submitted to the journal *Information Processing & Management*, for their special issue *Blockchain for Information Systems Management and Security.*

1.4 Outline of Thesis

This thesis is organized into nine chapters. We have presented our motivation, goal, methodology, and contribution in Chapter 1. Chapter 2 presents essential background material and concepts which are relevant to the thesis. We give a brief introduction to the healthcare domain, cryptographic principles, blockchain, smart contracts, and Ethereum. Chapter 3 presents a novel framework for categorizing evidence for trust in the healthcare domain. Chapter 4 defines a set of requirements for our platform. Chapter 5 presents our architectural models and system design. Chapter 6 describes our implementation of the platform. Chapter 7 presents our test results in the form of system testing, requirements validation and metrics. Chapter 8 includes a discussion and summary of our main findings. Finally, we conclude our work in Chapter 9.

Chapter 2
Background and related literature

This chapter presents essential background material for the remainder of this thesis. We will provide background on the healthcare domain, foundational cryptographic methods, and relevant context for blockchain. Finally, we present prior art on applying blockchain in the healthcare domain.

2.1 The Health Domain

The healthcare domain is inherently complex. Hospitals, insurance providers, clinics, educational institutions, virtual healthcare platforms, and a range of other stakeholders together compose the domain. In this section, we present some essential background on relevant trends in the healthcare domain and background on evaluating patient outcomes.

2.1.1 The Case for Virtualized Healthcare Services

The world is going through an overall increase in global life expectancy combined with a decrease in the global birth rate. This trend is causing a demographic shift where a smaller number of people in the workforce must support an increasing number of people outside of it. This trend is reflected in the age dependency ratio, measuring the ratio between dependants and the working-age population. This ratio has decreased globally from 74% in 1960 to 54% in 2018 [11], and from 58% to 53% in Norway. However, the growth in global life expectancy leads us to believe that these numbers will increase over time.

A growing dependency ratio demands increased capacity within the healthcare system. Otherwise, health services may become unavailable to patients, leading to worse patient outcomes. One way to grow this capacity is through technological innovations for increasing the efficiency of the existing healthcare infrastructure. Initiatives such as Healthcare 4.0 [3] facilitate for a strong shift towards virtualized healthcare services to increase efficiency. Such platforms see steadily increasing usage, and

we expect an accelerated growth as the world struggles to fight a global pandemic. Additionally, policy focus from regulatory organizations such as WHO Europe [12] further strengthens a shift towards such services.

2.1.2 Relevant Trends in Healthcare

Besides the technological and demographic changes mentioned in Section 2.1.1, we can notice other trends relevant to trust in the healthcare domain. These trends show how changes in behavior for how patients and healthcare workers interact, both between each other and with the healthcare system in general. These changes may have a sizeable structural impact on the future organization of the healthcare system. In our further work [10], we define the following significant trends of relevance:

1. **Virtualization of healthcare:** Due to initiatives such as Healthcare 4.0 and changes in demographic needs, the healthcare system will become increasingly virtualized. Physical care will increasingly become the last resort for specialized care. Routine tasks such as consultations with practitioners will increasingly be done through virtual healthcare platforms, where both people and AI healthcare workers can serve as a practitioner. Furthermore, new technology such as personal sensors and devices enable automated and personalized healthcare services outside of a traditional consultancy setting.

2. **Healthcare across boarders and jurisdictions:** An increase in the globalization caused by improved communication and transportation infrastructure will continue to trigger changes in the healthcare system. These changes lead to increased healthcare worker mobility across jurisdictions and borders. Patient mobility due to long-term movement or short-term search for cheaper or better medical care will also continue.

3. **Increased specialized care:** Continuing innovation in health sciences will result in the healthcare domain becoming larger in scope. Healthcare workers have to handle increasingly advanced domains within the healthcare industry. Such patterns lead to requirements for the increased specialization of competence. Therefore, patients seeking healthcare services should be able to verify that the healthcare worker has competence in the field relevant to the care they seek.

4. **Patient empowerment:** The healthcare domain is increasingly giving patients the ability to choose their healthcare services and providers. This change is triggered by greater availability due to easier mobility and increased technological advances, enabling personalized care. However, healthcare systems are not currently suited for a patient-controlled environment, as it requires advanced access control mechanisms and increased data sharing.

2.1.3 Patient Reported Outcomes

One way to measure health outcomes from treatments and clinical recommendations are through Patient Reported Outcomes (PRO) [13]. There are two standardized approaches for measuring PROs: Patient Reported Outcome Measures (PROM) and Patient Reported Experience Measures (PREM) [14]. They, among other factors, can measure the functional status associated with a treatment or the healthcare which the patients have received. PROMs and PREMs are currently a critical metric for evaluating the quality of a healthcare system. The metrics are collected and compared on national levels for comparing healthcare systems, where they indicate the quality of care within a country. We have chosen PREMs to capture the patient experience metrics related to the virtual interaction with the caregiver. These can, for example, be satisfaction rates for patients' experience with their treatment, the healthcare worker, or the virtual setting of the healthcare institution.

2.2 General Cryptographic Context

The proof-of-concept in this thesis uses the Ethereum blockchain to store data about trust relationships, treatments, and evaluations within the healthcare domain. To achieve this, we rely on many cryptographic primitives. Some of these concepts are used directly within our proof-of-concept, while others are vital pieces to understand the underlying workings of the Ethereum blockchain and the tools used to interact with it. This section will explain the needed background on these topics.

2.2.1 Cryptographic Hash Functions

Hash functions are a family of functions which take binary data of arbitrary length as input and output a fixed-sized output. The input is called a *message*, while the output is called a *digest*. Furthermore, hash functions should have the following properties [15]:

- **Collision-Resistance:** It should be computationally infeasible to find a message M and M' such that $H(M) = H(M')$.
- **Pre-image Resistance:** Hash functions should not be computationally feasible to invert. Given $H(M)$ it should be computationally infeasible to find M.
- **2nd Pre-image Resistance:** Given any message-digest pair $(M, H(M))$, it should be computationally infeasible to find another message M' such that $H(M) = H(M')$.

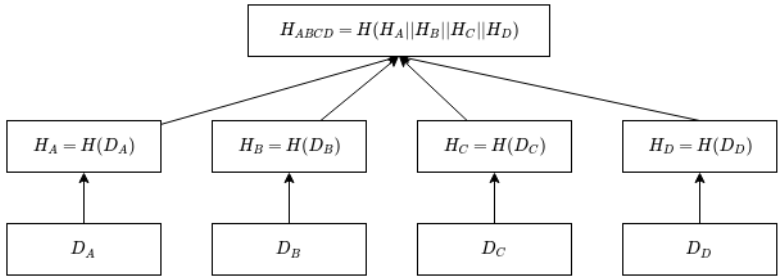


Figure 2.1: A hash list for calculating a root hash over a set of data

Cryptographic hash functions are the workhorses of modern cryptography, and by deduction blockchain. The most widely used hash functions fall within the Secure Hash Algorithm (SHA) specifications. Within the Ethereum platform, SHA-3 (using the KECCAK family of hash functions [16]) is most widely deployed [2].

2.2.2 Keccak Hash Functions

Keccak hash functions [17] are a family of modern cryptographic hash functions. They are the latest generation of the NIST standardized SHA cryptographic hash functions, known as SHA-3. The different variants of the functions have different input and output sizes. The function family addresses concrete cryptoanalysis attacks on the previous generations of SHA functions while improving speed and hardware performance. This is achieved by using a sponge construction [18] combined with Keccak-f [17] functions.

2.2.3 Merkle Trees

We can use cryptographic hash functions to prove the inclusion and integrity of data within a set. One way to achieve this is by calculating a cryptographic hash, often called a root hash, as a function over the full set of data. Given that the root hash is known, any party who has access to the data can verify its presence and inclusion by recalculating the hash and compare it to the root hash. A trivial way to calculate such a hash is through a hash-list, as shown in Figure 2.1. In a hash-list, we first calculate the cryptographic hash of each data item. We then calculate the root-hash by hashing the concatenation of all the produced hashes. A verifier who wants to ensure that they have the full dataset and verify its integrity may recalculate the hash and compare it with the root-hash. If equal, they can believe that the two underlying datasets were equal.

As described in Merkels' 1979 patent [19], an *authentication tree*, *hash-tree* or simply *Merkle tree* is also used for calculating a root-hash over a dataset. It has the same

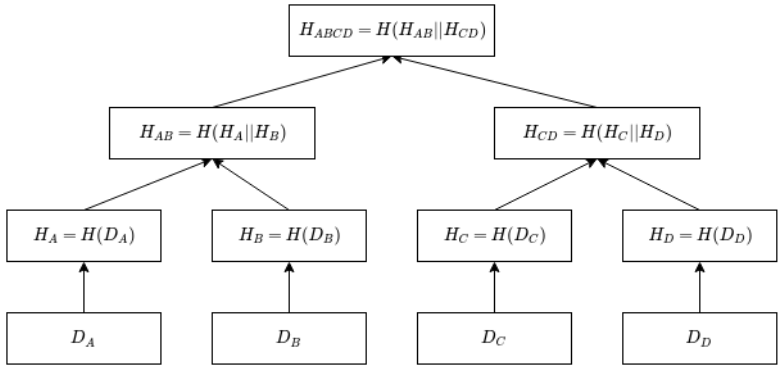


Figure 2.2: A Merkle tree for calculating a root hash over a set of data

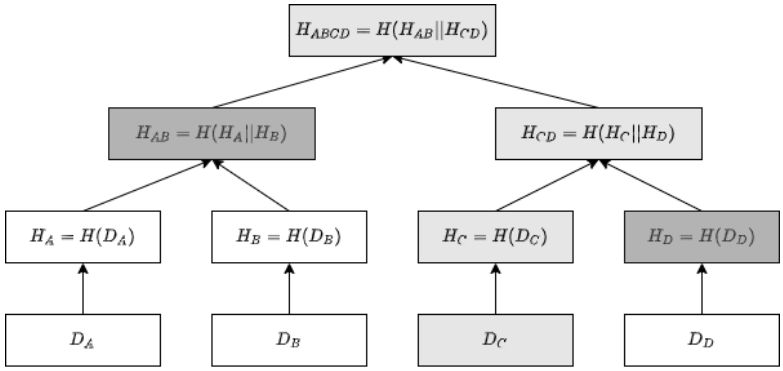


Figure 2.3: Verifying the inclusion of a data-item through Merkle-tree traversing.

intent as hash-lists but enables more efficient verification of data by calculating the root-hash through a tree structure. Figure 2.2 shows a Merkle tree over a set of 4 data elements. Each leaf-node in the hash component of the tree is found by hashing the corresponding data with a cryptographic hash function. We find parent node hashes by hashing the concatenation of the children hashes. We repeat this process until we reach a root hash, which serves as an integrity hash for the whole dataset.

With Merkle trees, verifiers can calculate hashes starting bottom-up from the leaf, ending with the root. Figure 2.3 shows an example where we verify the presence of D_C. Here, we want to prove the presence of the data D_C by recalculating all hashes leading to the root-hash H_{ABCD}. Given the hashes H_D and H_{AB}, we can calculate the hashes H_C, H_{CD} and finally H_{ABCD}. We compare the calculated root hash with the expected root-hash. In this proof, we only need a subset of the tree to verify the presence of data. This stands in contrast to hash-lists, where we must know all hashes to verify integrity.

2.2.4 Public Key Cryptography

Public key cryptography provides authentication or confidentiality of data through the means of asymmetric key-pairs. These key-pairs are composed of two main components, a secret key and a public key, where the latter is calculated based on the secret key. As the name implies, the secret key is never shared, while we share the public key with other parties. How we use these keys depends on the cryptographic context in which we apply them. If we seek confidentiality, we apply public key encryption. If authentication is required, we use the keys for digital signatures. We first briefly explain digital signatures and public key encryption from a high level. We then dig onto the Elliptic Curve Digital Signature Algorithm (ECDSA) cryptosystem in detail, as this is essential for understanding the underlying workings of Ethereum.

Digital Signatures

Banks want to verify the sender of a bank transfer. Email recipients want to know if a mail is legitimate or a phishing attempt. Blockchains want to verify the origin of a transaction. Digital signatures are the enabling technology for making these processes secure. We can view them as the improved, virtualized, and cryptographically secured counterpart to physical signatures. In general, a signer uses their secret key along with the message they wants to sign to create a signature that is sent along with the message. To verify that the signature is correct, the verifier can use the signer's public key along with the message, thus proving that the secret key indeed signed the message. If the verifier can be certain that only the expected sender possesses the secret key, the verifier can be sure that the message originates from that specific sender. This system also brings *non-repudiation* - once a message with a signature is known, the secret key holder cannot deny that they signed it.

Public Key Encryption

In other settings, we may want to ensure that only the intended receiver can read a message. For example, when we send an email with confidential information, send a secret message to someone, or publish some confidential data on a blockchain. In this case, we can apply public key encryption. Here, the sender can use the public key of the receiver to encrypt the message through a one-way function. The only way to recover the message is through another function that relies on the secret key. Thus, the sender can ensure that only the holder of the secret key can see the message.

2.2.5 The ECDSA Cryptosystem

Digital signatures are an enabling technology used in blockchains. The Ethereum blockchain uses the ECDSA cryptosystem [20], which is a modern alternation of the DSA cryptosystem designed to work with Elliptic curves over finite fields.

The DSA Cryptosystem

The Digital Signature Algorithm (DSA) cryptosystem is a set of key-generation, signature generation and signature verification algorithms. DSA is a variant of the ElGamal signature scheme [21], which is secure due to the difficulty of computing discrete logarithms.

Shortcomings of Classical Digital Signature Schemes in Blockchain

One of the problems of using a classical digital signature such as DSA within the blockchain domain is the key length of at least 1024 bits, and often longer. Public keys are often used as the identifiers in most blockchain systems. Therefore, we often include a public key in most messages to a blockchain network. As these messages, in general, are small, the large identifier will result in significant overhead. Furthermore, the computational complexity of the verification process associated with long keys has a high cost when these computations are repeated frequently across a large network of independent nodes.

Introducing Elliptic Curves over Finite Fields

We can use cryptosystems with shorter keys to address the two problems of key-length and computational complexity. One alternative for this is digital signature schemes based on elliptic curves. In general, these cryptosystems also apply the discrete logarithm problem but doing it over points on an elliptic curve over a finite field. We refer the reader to Koblitz [22] for further information on both finite fields and elliptic curves.

ECDSA is the elliptic curve equivalent to the DSA cryptosystem. By using elliptic curves, the ECDSA cryptosystem can deliver the same level of security, but with dramatically lower parameter and key sizes, increasing the level of security per key-bit extensively. A 3072-bit DSA public key has the same security level as a 256-bit ECDSA key. The later is of a size where it is better suited as a tool for blockchains.

2.2.6 Zero Knowledge Proofs

Zero-knowledge protocols are cryptographically backed methods allowing a prover to show that they know some information x without revealing anything about the information x itself. We can use these proofs in multiple contexts, where many of these proofs are useful for either the implementation of Blockchains themselves or applications using blockchains. Examples of common uses for zero-knowledge-proofs include:

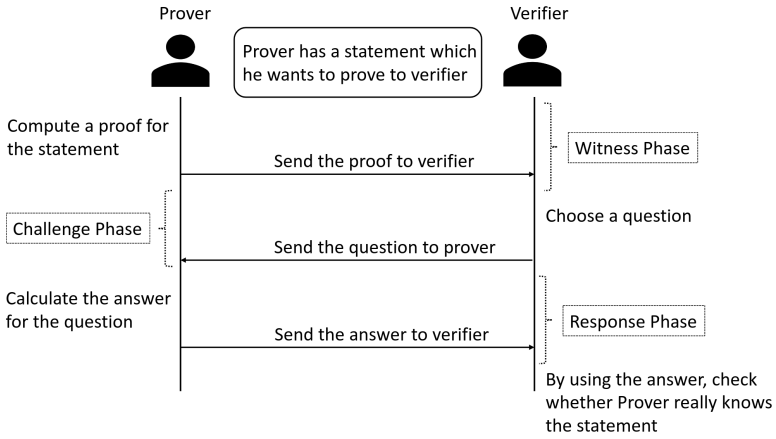


Figure 2.4: Interactive variant of a general zero-knowledge protocol [1]

1. Identification: A prover aims to prove they know x, which is associated with their identity. The zero-knowledge proof can thus be used as an identification mechanism, where the identifier shows they know x. An example of this is the Schnorr protocol [23], a simple zero-knowledge protocol for showing ownership of a secret key related to a given public key, without revealing it.

2. Group inclusion: Given a group $g \in G$ where $g_i = f(x_i)$, a prover may show that they know a x_n resulting in a g_n which is included in the group, without neither revealing x_n or g_n. Such proofs are the cornerstone of systems such as ZeroCoin [24] and ZeroCash [25].

The general process of an interactive zero-knowledge protocol is shown in Figure 2.4. The first phase is the *commitment phase*, or *witness phase*. Here, the prover creates an initial commitment based on the fact which they try to prove. The following phase is called the *challenge phase*. Here, the verifier will generate a question and send this to the prover. The prover can only confidently answer the question if they know the underlying committed fact. In the last phase, the *response phase*, the prover sends its answer to the verifier, who checks it for correctness. However, there may be a realistic chance for the prover to guess the correct answer. Thus, the prover and verifier can repeat the challenge and response phase. We repeat this until the probability of guessing all the correct answers is below an acceptable threshold.

Within the topic of blockchain, Zero-knowledge proofs are useful in a range of different use cases. Examples include proving identity, showing an ability to spend coins, or showing membership inclusion. However, blockchains are often immutable and offer no means of direct interaction between provers and verifiers. Thus, advances in non-

interactive variants zero-knowledge proofs have been the main driver of usage within blockchain platforms. Protocols such as ZK-SNARKs [26] enable non-interactive proofs of knowledge where the proofs are of size linear with the secret size.

2.3 Blockchain

We can describe a Blockchain as a distributed data store managed by a network of trustless computers organized in a peer-to-peer network. Data in a blockchain should be immutable, meaning it cannot be changed once added. We can achieve this behavior by using a wide range of *cryptographic primitives* combined with mechanisms to reach *consensus* in the trustless network. The result is a shared data-store which can be updated and shared across different organizations and stakeholders, without strict trust assumptions between these.

In general, we can add data to a blockchain by creating a *transaction*. Transactions contain the data which we want to add to the blockchain, along with some metadata describing the transaction. Transactions also contain a signature from a cryptographic key. By using the signature, we can link a transaction to the owner of the given key-pair in a non-repudiable manner. The signature serves as an identification and authentication mechanism.

Blockchain designs that neither use *blocks* or *chains* exist [27]. However, we will only focus on the classical *blocks and chain* based blockchain architecture as this is closely related to the system implementation in this thesis. We will present blockchain from a classical perspective, by using the Bitcoin blockchain as an example. Bitcoin offers an understandable and straightforward system design while showing the most critical patterns used in advanced blockchain designs.

2.3.1 Clarifying Terminology

Conversations about blockchain often become confusing because of the missing distinction between blockchain ledgers and the network of underlying computers. Figure 2.5 shows the overall framework for our terminology. In this thesis, we will use the following concepts consistently:

A blockchain ledger is the data structure that stores immutable data. The ledger stores the transactions themselves. As new transactions arrive, we add new blocks containing these to the ledger. In this context, principles like smart contracts and cryptographic hashes are of significant relevance.

A blockchain network is the network of nodes that cooperate to maintain a shared blockchain ledger and propagate transactions. All nodes in a blockchain

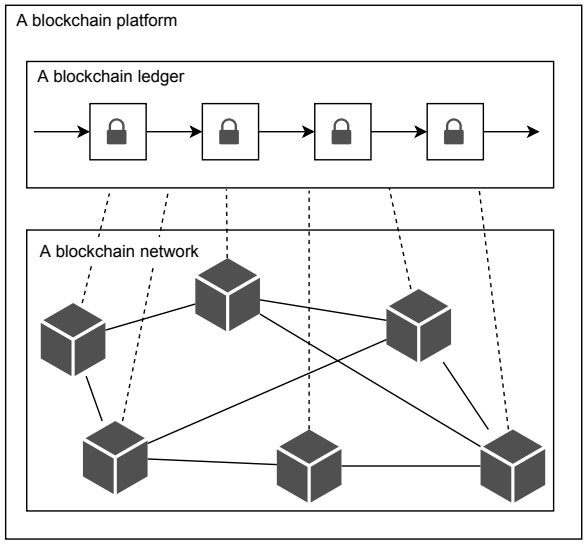


Figure 2.5: A simple illustration showing the concept of a blockchain platform

network are generally not assumed to trust each other, and they, therefore, rely on *consensus algorithms* to reach an agreement on how they build up the blockchain ledger. Applications that want to interact with the blockchain ledger will do so by interacting with any node in the network. The node can receive transactions, which are propagated onwards in the network. We can also query nodes for the current state of the ledger.

A blockchain platform is the composition of a network and the ledger it maintains. When we conceptually talk about adding transactions to the blockchain, we refer to the integrated process of sending the transaction to a node, the propagation of the transaction in the network, the consensus procedures performed, and finally adding the transaction to the ledger within a new block.

2.3.2 The Blocks and Chain of Blockchain

Blockchain ledgers are a data structure composed of a series of bundles with transactions. We call these bundles blocks, and they both contain transactions and some other metadata. The blockchain network nodes will create blocks based on the transactions they have observed. The network nodes will apply some consensus mechanism, which will eventually append (referred to as *mining*) the block to the blockchain ledger, linking it back to a previous block. In this section, we dig into what transactions are, how blocks are composed, and how they are linked together.

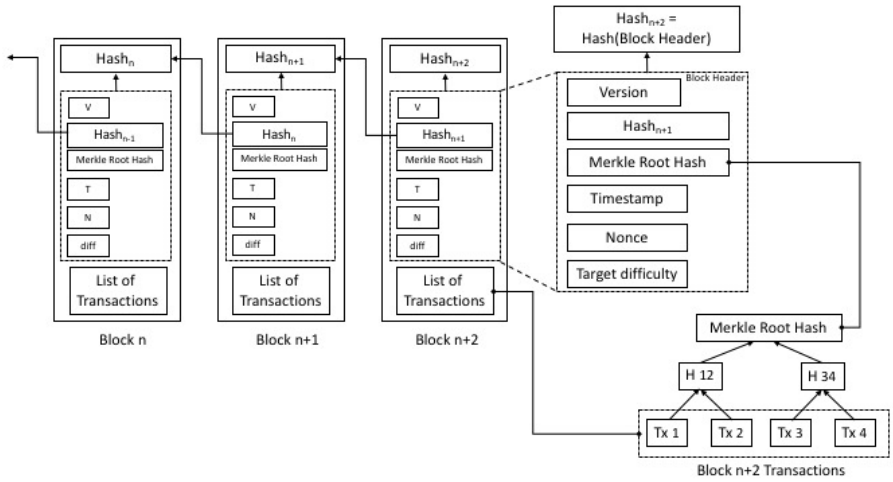


Figure 2.6: The structure of the bitcoin blockchain [1].

We use the Bitcoin blockchain ledger, as presented in Figure 2.6 as the context of our examples.

Transactions are a data structure used for interacting with a blockchain platform. Users of the platform will create a transaction that indicates the process they want to invoke on the blockchain. Within the bitcoin ledger, this process is typically sending some currency from one account to another. We encode our process invocation as a payload within the transaction, add metadata including the receiver, and finally sign it with a key-pair. The signature links the transaction to the sender. In addition to sending currency, we can invoke a large set of predefined processes. The payload in our transaction indicates which of these actions to invoke.

Blocks are in essence a bundle of transactions. By using a consensus mechanism, network participants create blocks by selecting a set of received transactions and confirming their presence through some kind of proof. The Bitcoin blockchain platform, as shown in Figure 2.6, uses a Merkle tree to create a root-hash over all the transactions contained within the block. If a prover wants to show that a transaction was indeed present in a given block, they can use a Merkle tree proof (see Section 2.2.3) to convince the verifier of this. Thus, once we have added a transaction to a block, its presence cannot be disputed. By applying Merkle trees, we can also provide non-repudiation of transaction ordering, allowing us to construct a complete

transaction history. This ordering allows all transactions to have a relative ordering, either occurring before or after any other transaction.

Chains are a link from a block to the previous one in the chain, typically represented by including the hash of the previous block within the next block's data. Figure 2.6, shows how a chain materializes in the Bitcoin ledger. Here, the block header contains the hash of the previous block. The main intent of this chain is to create a fixed ordering of blocks, allowing us to form a strict global ordering of transactions. We can use this ordering to ensure that future transactions are valid.

2.4 Privacy and Security for Blockchain

Using a public blockchain platform poses some key challenges from a privacy and security perspective. The strict regulations for patient privacy and data ownership makes the health domain hard to combine with blockchain unless specific steps are made towards ensuring these requirements. Traditional centralized storage structures can be protected through firewalls, encryption, and local access control mechanisms. This set of mechanisms is not possible to apply in the same manner to a blockchain-based application. However, there are tools available to us which are possible to use on the blockchain. These are Cryptographic tools, and they can be used in multiple different contexts, both for data published on the blockchain ledger itself or outside of it.

2.4.1 Signature Schemes

Digital signatures schemes are a fundamental building-block for blockchain platforms. We have previously described the ECDSA cryptosystem for key generation and signatures. These cryptosystems address the fundamental need of blockchains where we need to create keys and sign transactions by using these keys. However, if the blockchain platform or contract requires quality attributes such as privacy, anonymity or unlinkability, then different and more complex signature schemes may be required.

Multi-Signatures

Multi-signatures is a signature scheme where a group of participants uses individual keys to collectively sign a message, as illustrated in Figure 2.7. A single stakeholder may use multi-signatures to increase the security of their credentials. Alternatively, a group of signers may use multi-signatures to indicate a multi-party agreement. An alternative to the latter is for each of the participants to sign a message individually. While functionally similar, multi-signature schemes result in a fixed-size joint signature regardless of the number of participants. Another variant of this scheme is N-of-M

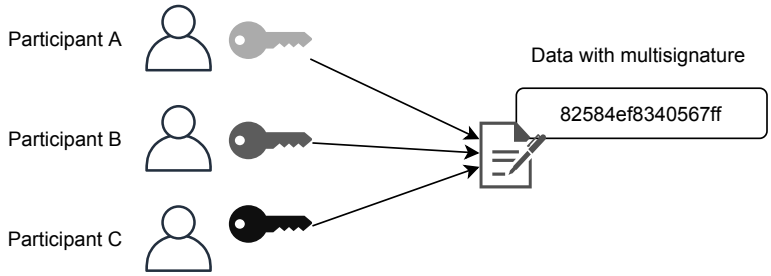


Figure 2.7: A figure showing the concept of multi-signatures

multi-signatures. Here, we set a limit n of required signatures from a group of known signers $s \in M$. If $|s| >= n$ then the multi-signature is valid.

Blind Signatures

Blind signatures is a scheme where the message creator (C) and signer (I) are different entities. Generally, blind signatures are created through a process where C has a message m. This message is *blinded* by combining it with a secret component s, creating the *blinded message* $\overline{m}$. C sends $\overline{m}$ to I who signs it, thus creating the *blinded signature* $\overline{\sigma}$ which is valid for the public key of I, p_I. The signature is sent to C, who can use the same secret element s with $\overline{\sigma}$ to extract a signature σ which is valid for the message m with the public key p_I. The resulting pair (m, σ) is a valid signed message from the signer I, where I does not know the message m.

We can use blind signatures to ensure message integrity and authenticity while preventing the sender from being associated with the message itself. We call this property unlinkability, and it is essential for providing anonymity to message creators in an environment where privacy is crucial.

2.4.2 Access Control

Access control dictates who can access data and in which way. Access Control Policies [28] describe general models for access control. These general models can be applied within the blockchain context to ensure that data written to it is legitimate. We can apply access control policies for multiple purposes within a blockchain platform. By incorporating access control schemes into the platform itself, we limit the entities who can access it. Alternatively, we can implement access control schemes to control the individual process invocations on a blockchain. In a fundamental sense, all blockchains apply a simple access control mechanism where only users with a given key-pair can get access to the account related to this.

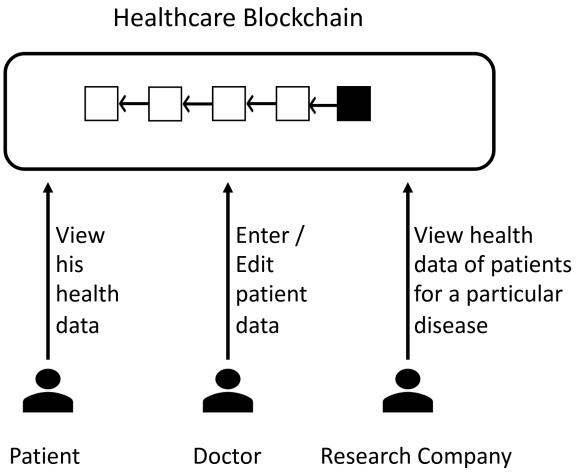


Figure 2.8: Generic Role-Based Access Control scheme for healthcare [1]

Discretionary Access Control (DAC)

DAC is an access control scheme centered around direct access to specific data objects. In this scheme, each data object a specific owner. Read and write access to the data is given to users or groups of users through actions from the owner. Limiting the access of an account on a blockchain to the owner of the related key-pair is DAC. Additionally, we may use blockchains for enforcing DAC elsewhere. Data stored at a location outside of the blockchain can use state on the blockchain to determine who has access to the data. In this case, the owner of the data can give access to the given piece of data by invoking a process on the blockchain.

Role-Based Access Control (RBAC)

RBAC is a general model for access control where users assume different roles. Data can also be configured to accessible by different roles. Thus, users can only access or modify data if they can assume a role with such permissions. Figure 2.8 shows an example where we use role-based access control in the context of patients, healthcare workers, and medical research institutions. In this case, only the stakeholder of a particular role may add new data to the blockchain.

2.5 Smart Contracts and the Ethereum Blockchain

The main intention of the Ethereum blockchain platform is to extend the blockchain paradigm from the financial domain into a platform where we can create general-purpose applications. From the top level, we can view the Ethereum platform as a

state-machine where transitions are performed based on valid transactions. Each transaction submitted to the blockchain alters the state through the function:

$$\sigma_{t+1} = \Upsilon(\sigma_t, T) \tag{2.1}$$

σ is the global state for the Ethereum blockchain platform, often described as the world state. Υ is the Ethereum state transition function, which produces a new world state based on the current world state and a transaction T. To ensure that all nodes participating in the blockchain network can deduce the same world state σ, they must all agree to a fixed ordering of transactions $S = [T_0, T_1, T_2, ..., T_t]$. If all nodes can agree upon such an ordering, they can deduce a common world state. They achieve this by using the transition function over the transaction set:

$$S = [T_0, T_1, \dots, T_t] \tag{2.2}$$
$$\sigma_t = \Upsilon(\dots \Upsilon(\Upsilon(\Upsilon(\sigma_0, T_0), T_1), T_2) \dots, T_t) \tag{2.3}$$

The purpose of the blockchain ledger and consensus mechanisms is to allow the nodes in the network to agree to such a transaction order. The ledger follows the same general structure as described in Section 2.3.2, where each block contains a set of ordered transactions which are cryptographically bound to the block via a root hash. By introducing blocks, we must alter the world state update function:

$$B_b = (\dots, (T_{b0}, T_{b1}, T_{b2}, \dots), \dots) \tag{2.4}$$
$$\sigma_b = \Omega(B_b, \Upsilon(\dots \Upsilon(\Upsilon(\Upsilon(\sigma_{b-1}, T_{b0}), T_{b1}), T_{b2}) \dots)) \tag{2.5}$$

Where σ_b is the world state after block B_b is processed. The block B_b contains the transaction set and the remaining data bound to the block. The Block transition function Ω combines the state changes from transactions and the block (e.g., rewards given to the miner of the block) and generates a new world state σ_b

The world state in the Ethereum blockchain platform is universal and expressive enough to handle a wide range of possible states. This expressiveness allows us to use it for general purpose applications, where we can define custom procedures and state to be stored on the platform. We can manage this via smart contracts, which can be uploaded by any user to the platform. Smart contracts act as an additional state-machine on top of the existing infrastructure, with their own set of valid transaction types. We can draw an analogy from smart contracts to object

instances, as they can store internal state, and users can interact with them through calling functions via transactions.

The Ethereum blockchain platform's composition follows the same fundamental principles as Bitcoin, as previously described in Section 2.3.2. The Ethereum platform expands on these principles by introducing additional complexity and concepts. The remainder of this section will explain accounts, smart contracts, transactions, costs, and the Ethereum ledger construction.

2.5.1 Ethereum Contextual Terminology

$KEC(x)$ refers to the Keccak-256 hash over a variable-length message x. Refer to Section 2.2.2 for an explanation of the hash function.

$RLP(x)$ refers to Recursive Length Prefix encoding, a serialization method for binary data. Refer to the Ethereum yellowpaper [2] for further details.

$\sigma[s]$ refers to the state of an account with address s

$\sigma[s]_n$ refers to the latest used nonce for the account with address s

$\sigma[s]_b$ refers to the balance in Ether for the account with address s

$Ether(ETH)$ is the cryptocurrency used in the Ethereum blockchain platform

Wei is the atomic unit of Ether. One Ether is defined as 10^{18} *wei*

Gas is a unit for the cost of transactions

2.5.2 Ethereum Accounts

Ethereum accounts are a fundamental building block in the Ethereum blockchain platform. All state in the Ethereum blockchain ledger is stored in the context of a given account with their public identifiers which are called *addresses*. The world state σ is a union of all account states denoted $\sigma[s]$ where s is the account address.

An account in Ethereum is, in essence, a ECDSA key-pair, as described in Section 2.2.5. Ethereum uses the elliptic curve SECP-256k1 [29] with 256-bit private keys, offering security equivalent to a 128-bit symmetric key. Secret keys p_r are created by selecting a random positive integer in the domain of the curve. By using a ECDSA key-pair, we can perform three important operations:

$$p_u = \text{ECDSAPUBKEY}(p_r) \tag{2.6}$$
$$v, r, u = \text{ECDSASIGN}(m, p_r) \tag{2.7}$$
$$p_u = \text{ECDSARECOVER}(m, v, r, u) \tag{2.8}$$

These functions are related to the definitions, as found in the ECDSA specification [20]. `ECDSAPUBKEY`, is part of the standard key generation algorithm and creates the public key p_u from the secret key p_r. `ECDSASIGN` [2] is a variant of the standard signature generation algorithm. In addition to the standard signature (r,u), an additional *recovery identifier* byte v is included. This identifier specifies the parity and finiteness for the point on the SECP-256k1 curve, where the signature component r is the x-value. The inclusion of the recovery parameter v allows us to recover the public key from a signed message with the `ECDSARECOVER` function, which is a variant of the standard ECDSA signature verification algorithm.

Address Generation

We can create the Ethereum address s of an account by compressing the ECDSA public key. This is done by taking the last 160-bits of the Keccak-256 hash function over the public key. However, as we do not persistently store the ECDSA public key, we use the following procedure to generate the address directly from our private key:

$$s = \beta_{96..255}(\text{KEC}(\text{ECDSAPUBKEY}(p_r))). \tag{2.9}$$

Signature Verification

We have previously stated that Ethereum does not use the standard ECDSA signature verification algorithm. This is caused by the choice of using addresses instead of public keys as the general identifier for accounts due to their shorter length. Therefore, we typically do not have the public key available to us during the signature verification process. Signature validation in Ethereum is performed by using the ECDSARECOVER function. Given a signature (v,r,u) and message m, we check if the recovered public key p_u^* yields the same address s^* as the expected address s:

$$p_u^* = \text{ECDSARECOVER}(m, v, r, u) \tag{2.10}$$
$$s^* = \beta_{96..255}(\text{KEC}(p_u^*)) \tag{2.11}$$
$$\text{Signature is valid} = \begin{cases} True, \text{if } s^* = s \\ False, \text{otherwise} \end{cases} \tag{2.12}$$

An additional feature of this scheme is the ability to recover the sending address from the signature. Given a signed transaction, we can use the signature (v,r,u) to recover the address of the sender. This allows us to omit to add the sender address in each transaction, decreasing the size of transactions.

Data Encoding Schemes

In some cases, an Ethereum account owner may want to sign an arbitrary piece of data with the key-pair related to their Ethereum account. In the Ethereum ecosystem, this common practice for identification, as the address of an account, is recoverable from a signature. This allows such signatures to be used for access control schemes, either within a smart contract or within a standalone application outside of the blockchain. However, this introduces risk for the signer as a program may trick a naive signer into signing a piece of data corresponding to a valid transaction. To solve this issue, EIP-712 [30] introduces the following encoding scheme for non-transactional data to be signed:

$$encode(b) = \text{"\x19Ethereum Signed Message:\n"}||\text{len}(b)||b \tag{2.13}$$

2.5.3 Smart Contracts

Smart contracts are a new paradigm implemented in the Ethereum blockchain. They are the main toolchain available to developers who want to create distributed applications that run directly on the blockchain. We refer to such applications as a dApp. Smart contracts are composed in the same manner as classes in object-oriented languages, as they have internal state, constructors, inheritance, and externally or internally accessible functions. Smart contracts are stored as Ethereum Virtual Machine (EVM) bytecode on the blockchain ledger, where they get their own address s, making them addressable as if they were any other account on the platform. The bytecode stored on the ledger defines the state variables, which is stored in the world state $\sigma[s]$. The bytecode also defines the possible transition functions for changing this state.

Smart contracts are typically developed writing code in a high-level language such as Solidity [31], which is compiled down to EVM bytecode. To deploy a smart contract, a user takes the compiled EVM bytecode and packs this into a contract creation transaction with the data T_d. They sign it using their ECDSA secret key, resulting in the signature component T_s, allowing the full transaction T to be composed. This transaction is sent to the Ethereum blockchain platform by sending it to a node in the Ethereum blockchain network.

Figure 2.9 shows how we compile a smart contract into bytecode and create a contract creation transaction for deploying the smart contract to the blockchain platform. Our

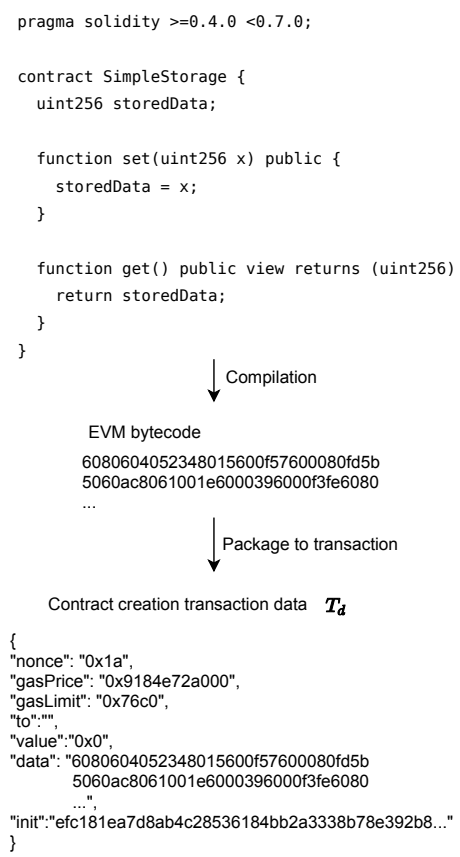


Figure 2.9: Compiling a smart contract and creating a contract creation transaction

example smart contract stores a single integer on the blockchain ledger. The value of this variable is stored on the blockchain in the context of the contract address s in the world state $\sigma[s]$. For interacting with the internal state, the contract defines two functions. The first is a procedure causing a state transition of the variable. We can interact with this function by creating a transaction for invoking it. The other is a view function, allowing other contracts or off-chain users to access the variable easily. This allows us to access the state by querying a node in the network, without creating a transaction.

Once a contract is added to the ledger via contract creation transactions, it gets assigned an address s. In contrast to addresses deduced from ECDSA key-pairs, this address is not associated with a given key-pair but is rather a calculated address under the control of the contract. This address is calculated by taking the last 160 bits ($\beta_{96..255}$) of the Keccak hash of the Recursive Length Prefix (RLP) encoded

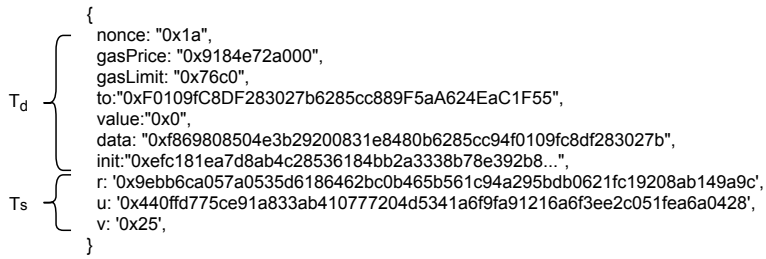


Figure 2.10: An example of the Ethereum transaction composition

sending address and nonce from the contract creation transaction [2]:

$$address = \beta_{96..255}(KEC(RLP((s, \sigma[s]_n - 1)))). \tag{2.14}$$

To interact with a contract, users have to create a new contract invocation transaction. This transaction contains EVM bytecode indicating which function to invoke in the smart contract, along with the function parameters. This bytecode is created by using the contracts Application Binary Interface (ABI). This interface can be created during the compilation of the contract and describes how to encode and decode contract calls into EVM bytecode. The destination of this transaction must be the address of the smart contract, as calculated in Equation 2.14.

2.5.4 Ethereum Transactions

Ethereum transactions, as illustrated in Figure 2.10, are a fixed data structure that can be interpreted by nodes in the Ethereum blockchain platform. Transactions are created by users, allowing them to publish new data on the Ethereum blockchain ledger. The first central component of a transaction is the data T_d, which describes a state transition in the Ethereum world state σ. In general, this data takes one of three forms:

1. A simple transfer of Ether (The cryptocurrency used by Ethereum) from one address to another.

2. Smart contract creation, where a user uploads the bytecode of a contract and gets a dedicated address for it.

3. A smart contract invocation, where the transaction contains data about which smart contract to use, which procedure to invoke, and parameters.

The other central part of a transaction is the signature T_s. This is a ECDSA signature generated with the private key of the account. The corresponding public address of the private key is the transaction sender, and the transaction will be executed in the context of this account.

Transaction Composition

In addition to the payload, the transactions contain many additional fields that are used for handling the transaction. They are important both for establishing the execution context, preventing attacks, and for building and disturbing blocks.

- **Nonce** is a monotonically increasing number associated with an address. Once the nonce of an address is present on the blockchain, all subsequent transactions from the same address must have an increased nonce value. The last observed nonce for an address on the blockchain is denoted $\sigma[s]_n$ as a function of the world state. Miners will not accept transactions with nonces lower or equal to the one stored in the world state. This prevents replay attacks where existing (and publicly available non-the-less) can be resubmitted to the platform.
- **gasPrice** is a value in *wei* set by the sender of the transaction. The total cost of a transaction is a function of the gas price and the gas cost of the transaction.
- **gasLimit** the upper acceptable cost bound for the transaction in terms of gas.
- **to** is the recipient of the transaction. In the case of an Ether transfer, this is the receiving account address. When calling a smart contract, this is the contract address. During contract creation, this value is empty.
- **value** is the amount of Ether to send to the target address. When the destination is a contract call or a contract creation, then the amount will be controlled by the receiving contract.
- **init** is a field that must be present during contract creation. It contains the data to be passed to the construction function of the contract.

2.5.5 Transaction Costs

Using the Ethereum blockchain has a cost associated with it. Each transaction submitted to the blockchain will require all the nodes in the network to execute the Ethereum state transition function, previously shown in Equation 2.1. The result of the transition function is found by executing the procedure associated with the transaction and will cause the node to execute a number of EVM instructions of different types. Each of these instructions has a cost associated with it in the form of *gas*, which is a variable unit of *Ether*.

- **Ether** is the main currency of the Ethereum blockchain. The main purpose of Ether is to be the fuel for transactions. Ether is attached to each transaction and is used to pay the transaction validators (miners) who include the transaction in blocks on the ledger. Ether can be gained either through mining or from receiving it from another account through a send transaction. Due to the intrinsic value of Ether, it is often used as a store of value and as a currency for payment.
- **Wei** is the smallest atomic unit of Ether. One Ether is defined as 10^{18} *wei*.
- **Gas** is a unit for transaction costs. When submitting a transaction, the sender will use the *gasPrice* field in the transaction to select the amount of *wei* to pay per unit of gas. Honest miners will prioritize the transactions with the highest gas prices when selecting the transactions to include in a block. In practice, this leads to an auctioning system, where the transaction will have a greater chance of being included in a block if a higher *gasPrice* is set.

The Gas Price of Transactions

The gas cost of transactions in Ethereum is a function of the executed instructions by the miner. These transactions can be calls to arbitrary contracts, which include procedures defined in a Turing-complete language. Thus, miners cannot predict the cost of a transaction before its execution. The general procedure for transaction payment goes like this:

1. The sender creates a transaction. This transaction includes a *gasPrice*, denoted T_p, and a gasLimit, denoted T_g. This transaction is sent to an Ethereum node, which propagates it in the network.
2. A miner picks the transaction for inclusion in a block and executes the transaction. As EVM instructions are executed, the miner keeps track of the cost through mapping the instruction to a gas cost, as sampled in Table 2.1. If the gas cost G rises above the gasLimit T_g, then the transaction is reverted (but still charged for.) If the total cost of the transaction $T_p * G$ exceeds the account balance $\sigma[s]_b$, then the transaction is also reverted and charged for.
3. Once the transaction terminates, the final cost G is charged from the senders account balance $\sigma[s]_b$.

Generally, the sender of a transaction will select a gasPrice for the transaction aligned with its urgency. The average gasPrice for the transactions in recently mined blocks is often used as a reference for determining this price.

Name	Gas cost	Description
G_{SSET}	20000	Paid for a SSTORE operation when the storage value is set to non-zero from zero. Stores a 256-bit value on the ledger
$G_{JUMPDEST}$	1	Paid for a JUMPDEST operation. Mark a memory address for jumps
G_{SLOAD}	200	Paid for a SLOAD operation. Loads a 256-bit value into memory

Table 2.1: Gas costs associated with a sample of EVM instructions [2]

Minimum Balances

We have described how the cost of a transaction is calculated during execution, and charged from the senders' account balance. In this setting, miners are never punished for selecting transactions which cannot complete and are reverted, as the gas for all instructions up until termination is still charged for. However, this setup opens up for some denial of service attacks as transactions may be created from accounts with no Ether balance at all. If a malicious actor produces a large number of such transactions, miners may use most of their time on reverting transactions. Therefore, Ethereum nodes require the sending account to have a minimal Ether balance for their transaction to be propagated in the network or processed. This minimum balance is equal to the maximum possible gas cost for the given transaction, as it is calculated as such:

$$B_{min} = T_p \cdot T_g \tag{2.15}$$

where T_p is the *gasPrice* and T_g is the *gasLimit*. Both are defined as parameters in each transaction, and a valid range for these are determined by miners. Ethereum nodes will check $B_{min} < \sigma[s]_b$ before processing or propagating a transaction.

2.5.6 Ethereum Ledger Construction

The construction of the Ethereum blockchain ledger is fundamentally similar to the Bitcoin construction shown in Figure 2.6. However, some key modifications are added to accommodate the more expressive world state σ, which must support arbitrary updates based on smart contracts written by users. We show the construction in Figure 2.11, where we can observe two new root hashes in the block header. The state root-hash is a cryptographic hash over the world-state after the transactions in the block are applied. The receipts root hash verifies the receipts from all transactions, showing the effect of each applied transaction.

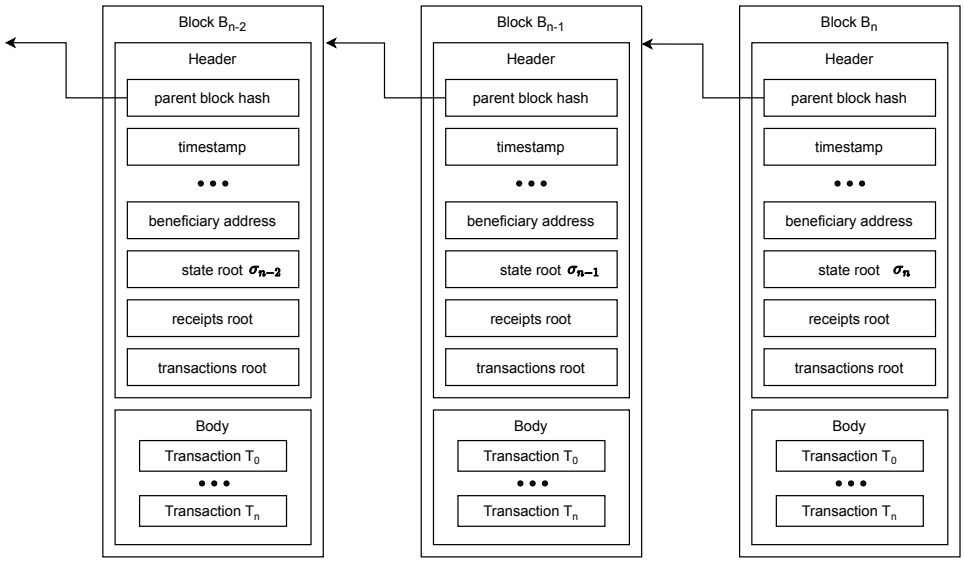


Figure 2.11: The structure of the Ethereum blockchain ledger.

Modified Merkle Patricia Trees

A Merkle Patricia Tree is a prefix-tree structure that is used to create an index over a set of data while incorporating tree-hash mechanisms to find a root-hash for the whole tree. It can be interpreted as Patricia tree [32], where the root hash can be calculated in the same manner as in a Merkle tree (See Section 2.2.3). In Ethereum, the transaction root, state root, and receipt root are constructed via Modified Merkle Patricia Trees over transactions, the world state, and transaction receipts.

Transaction Receipts

When miners execute transactions, they first and foremost use the current world-state along with the transaction to generate a new world state via the Ethereum state transition function, as shown in Equation 2.1. In addition to the world-state update, they also produce a *Transaction receipt* as an independent output from the transition function. This general pattern is shown in Figure 2.12. These receipts contain metadata about the transaction execution, and contain gas usage in the block so far, the logs created by the transaction execution, a bloom filter for the logs, and the transaction termination status code.

Transaction receipts are useful for multiple purposes. They are first and foremost used by miners to ensure consistency when executing transactions. Developers creating

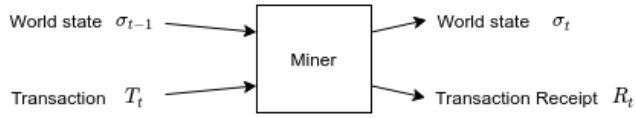


Figure 2.12: Executing a Ethereum transaction to produce a new world state and transaction receipt.

Address	Balance
570e7ff1	10 ETH
570e97be	3 ETH
0b3a9957	5 ETH
0b3af9be	1 ETH

Table 2.2: Initial account balances for accounts used in Figure 2.13

dApps can parse receipts from transactions sent to the Ethereum platform. The exit status code will show if the transaction succeeded or failed (e.g., when the balance of the sender was too low to execute the whole transaction). Furthermore, the EVM includes an event system, where events are stored in the produced logs. By parsing these events, developers may create subscription-based architectures.

The State Root Hash

The state-root hash included in each block serves as an integrity check over the world state σ_n. To check the validity of a world state, the verifier starts with a previously calculated world-state σ_{n-1}. They then run all the transactions in the block by using the Ethereum Block transition function (Equation 2.5). The block is valid if the root-hash of the resulting Modified Merkle Patricia tree is equal to the state root hash.

We show a practical example for how these-root hashes are constructed in Figure 2.13. We start with a world state σ_{n-2} where the initial balances are given in Table 2.2. Block B_{n-1} contains a transaction where the account with address *0b3a9957* sends 3ETH to the address *570e97be*. In the subsequent block B_n, the account with address *570e7ff1* sends 6ETH to the address *0b3a9957*. During each transition between world states σ_n and σ_{n+1}, only the differences in the world states as produced by the transactions are materialized.

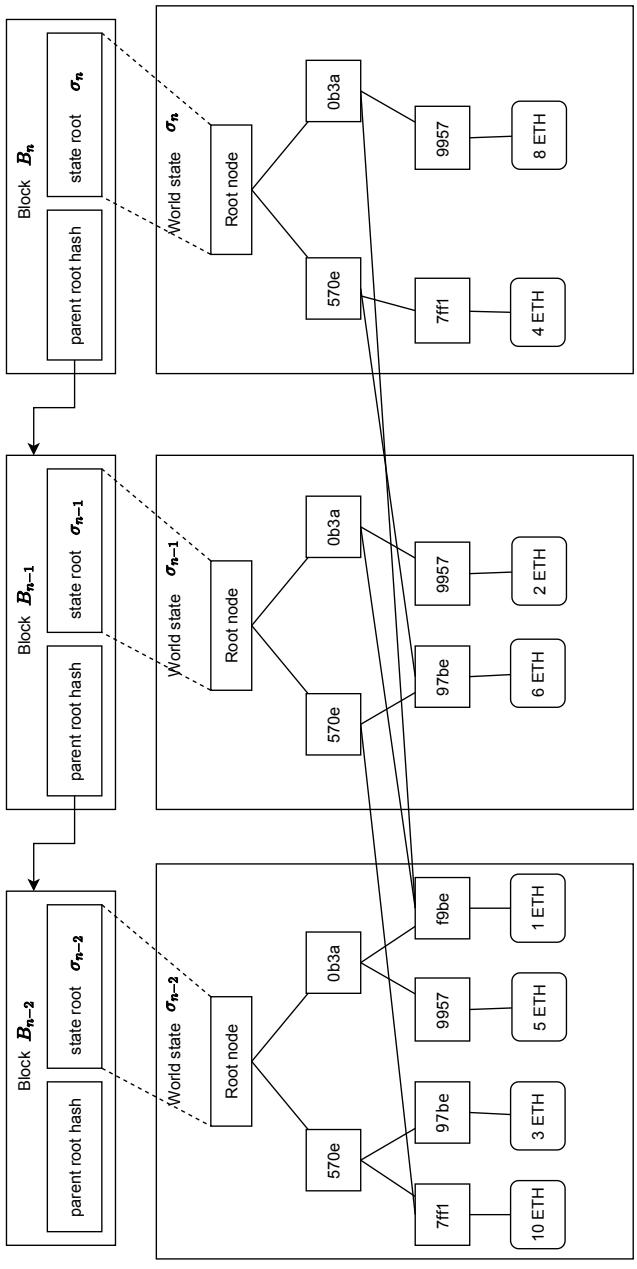


Figure 2.13: This figure shows how the Modified Merkle Patricia tree from which is root-hash is produced updates when adding new transactions.

2.6 Prior Art

To our knowledge, no prior proof-of-concept application for providing trust in healthcare workers' qualifications, skills, and competence through blockchain technology exists. However, blockchain-based proofs-of-concept exist for other topics within the healthcare domain. Furthermore, research on trust in healthcare predates blockchain technology and is extensive from both a technical and analytical perspective. Finally, we note that extensive analytical work exists on how we can apply blockchain in healthcare.

2.6.1 Blockchain in Healthcare

Extensive research on the use-cases of blockchain within the healthcare exists. Blockchain is often proposed as a solution to the data management problems within the healthcare domain. In particular, the potential for solving problems related to inter-organizational data-sharing is high. The majority of previous research on blockchain in healthcare has focused on managing Electronic Health Records (EHR) or Personal Health Records (PHR) [33]. Some applications related to establishing trust in healthcare through blockchain exist [34, 35, 36]. However, this research is focused on conceptual analysis and does not include any practical implementations, designs, or proofs-of-concept.

MedRec [37] is a proof-of-concept application that relies on the existing data-infrastructure within the healthcare domain. They seek to ease data sharing and access control by using the blockchain as a public registry for EHRs. They use this registry to store a simple mapping between a pseudonymous patient identifier, providers, and pointers to EHRs. Also, a permission system is implemented and appended with the public data pointers. Overall, this architecture allows patients and organizations to locate and access data from a range of providers given patient consent. The application initially used the public Ethereum network, but a transition process to a private ledger is in progress.

Ancile [38] is a system design for controlling access to EHRs, and tries to solve the same problem statement as MedRec. They improve on previous solutions by including a key management mechanism for symmetric keys over the blockchain to encrypt the data stored at providers. The system is designed for a permissioned Ethereum-based blockchain, but does not specify the underlying platform further. They use a distributed governance mechanism to manage permissions for interacting with the blockchain.

Blockcerts [39] is a standard developed for verifying certifications of competence by storing signatures on a blockchain. The standard relies on existing trust relationships between the issuer and the verifier of the certificate. Baldi et al. [40] show that

certificates within this system can be spoofed. They also propose to use decentralized identifiers for governing such certificates.

2.6.2 Evaluating Healthcare Applications

Some research on requirements for blockchain applications within the healthcare domain exists. Zhang et al. [41] define a set of metrics for evaluating blockchain applications within the healthcare domain. Here they also describe some fundamental principles which should be applied when creating dApps within the domain. Although mainly directed towards the American Health Insurance Portability and Accountability Act, the framework can be generalized to specific requirements that will work in the European context.

Chapter 3
Trust Establishment in a Virtualized Healthcare Environment

Providing trust in healthcare workers by using existing data sources is a challenging issue. Currently, getting access to data related to a healthcare worker is problematic due to a low capability to share data within the healthcare domain. This section will first describe the general issues of data-sharing within the healthcare domain. In the context of these issues, we will describe how we can establish trust in a virtualized healthcare environment. Finally, we describe a novel framework for defining the data required to provide such trust.

3.1 Data Sharing in the Healthcare Domain

A fundamental problem within the healthcare domain is a low capability to share data between healthcare institutions and services. The low data-sharing capability materializes as multiple different problems. Blockchain, as a technology, can address some of them. Previous research on the use cases of blockchain in healthcare has defined four healthcare industry requirements where the technology has great potential [33, 42]:

1. **Interoperability** The ability of systems to easily exchange information is called interoperability. To achieve this, we must make data accessible through standard means of communication and store it in a machine-readable, parsable, and commonly used format. Within the healthcare industry, the lack of interoperability results in medical data stored in isolated stores within healthcare institutions. Moving these datasets to other systems and organizations is challenging, as it may not use standard formats for storage or formatting. Patients and healthcare workers have a hard time accessing their data since they may be fragmented across multiple healthcare institutions and hard to integrate due to variable formatting and access methods.
2. **Security** The volume and variety of data generated within the healthcare domain is increasing. In addition to EHRs, we can now create produce data

via on-body sensors and virtualized healthcare platforms. These datasets span outside of the traditional EHR storage systems. As the surface area of data increases, so does the requirements to keep this data secure. Once data expands outside the traditional healthcare providers, existing requirements such as non-repudiation and universal access control mechanisms become more important. Patients should be able to trust that their data has not been modified, and has not been accessed by unauthorized parties.

3. **Data sharing** The lack of interoperability, along with strict security requirements, makes it hard for patients and healthcare workers to gain access to all their data in a unified view. While efforts towards centralized national stores for some datasets are undergoing in multiple countries [43], sharing between organizations and across borders remains a large problem. These initiatives try to solve the problem of data-sharing through the centralization and integration of data. However, better interoperability could instead result in more efficient data sharing and thus making federation over the data possible instead.

4. **Mobility** While also related to interoperability, the portability of data enables the mobility of healthcare workers and patients. A patient traveling between countries, changing services, or switching their healthcare domain should able to transfer their data from one healthcare service to another. Likewise, practitioners should be able to transfer data related to their experience, credentials, and performance between domains. Both these cases are, in general, very challenging with the current structure of the healthcare domain.

Blockchain can address these problems by offering infrastructure for inter-domain data sharing. We can apply the technology in multiple different architectures and configurations to achieve this. Examples include blockchains for discovering data through indexing, distributed storage of data, providing non-repudiation, or supporting access control mechanisms. Common for all of them is that blockchain can autonomously and instantly share information between trustless domains.

3.2 Data Sharing for Healthcare Workers

Most previous solutions [44, 37, 38] use Blockchain technology for addressing data-sharing problems in the context of patients and their EHRs. However, we note that these problems are just as severe for data contextual to healthcare workers. To show this, we frame the previously mentioned industry requirements in the context of healthcare workers:

1. **Interoperability** Data related to the healthcare worker are fragmented between data stores where formats and access methods vary from organization to organization. Additionally, this data is often stored in the context of patients. Building applications that can access and integrate all this data in the context of a healthcare worker may is therefore hard.

2. **Security** New security mechanisms within the healthcare domain try to enforce patient-controlled data. In such schemes, patients must give specific access to anyone who wants to access their data. However, the healthcare worker who may have produced the given data may not have this ability.

3. **Data sharing** As healthcare workers change employers, their data documenting their work-history does not follow. Thus, the evidence of their work-history becomes increasingly fragmented over time between different organizations.

4. **Mobility** The inability to share data contextual to healthcare workers' work-history may limit their ability to move across borders and jurisdictions. Gaining formal certifications and licenses can thus take a long time, reducing the overall efficiency of healthcare worker mobility.

3.3 Trust in a Virtualized Healthcare Environment

Trust in a physical healthcare environment is often taken for granted, as there is an inherent trust relationship between a patient and healthcare worker in the setting of a physical meeting within a healthcare institution [4]. This trust relationship can be rooted in factors such as the competence, reliability, compassion, and integrity of the healthcare worker [45].

Trust relationships from the physical setting are extended into the virtualized work when the patient is talking with a practitioner whom the patient already knows from a previous physical setting. In a fully virtualized healthcare domain where the patient does not know the healthcare worker before the virtual interaction, this principle cannot be used. Thus, there is a need for enabling such trust relationships to be built within a virtualized healthcare environment.

A Novel Framework for Trust in Healthcare Workers

We propose establishing a trust relationship between the patient and the healthcare worker in a virtualized setting through data sharing. The shared data includes evidence for trust. This evidence should convey some of the same information from which the physical trust relationships are rooted. In the context of the patient and healthcare worker relationship, we can define the following framework categorizing evidences for trust:

1. **Evidence of authority** The healthcare workers must be able to show that they have formal credentials allowing them to practice as a healthcare worker. They need a formal license, their background must be legitimate and approved, and they must be working in a trusted healthcare institution.

2. **Evidence of experience** The healthcare worker must convince the patient that they have the experience required to deal with the patient's specific health issues. As specialization increases, this evidence will increasingly be an essential ground for trust.

3. **Evidence of competence** Experience as a standalone metric does not convey information about the quality of care delivered. Therefore, the healthcare worker should also be able to show that they have previously delivered positive outcomes to patients. Thus, a metric for patients' satisfaction can be another crucial evidence.

By making these evidences available to patients, we can establish grounds for trust between the patient and the healthcare worker. However, designing such a solution is not trivial due to the significant problem-categories defined within the healthcare domain: Interoperability, Security, Data Sharing, and Mobility. These problems make it challenging to create an application that works on top of the existing organizational structure where data is fragmented over different organizations with a diverse set of formats. Although possible, we instead may require fundamentally different solutions with higher availability and transparency for patients, while ensuring the evidence they receive is not fraudulent.

Trust in AI healthcare workers

By defining a general model of evidence-based trust mechanisms, AI-healthcare workers can be compared to human healthcare workers, which can increase confidence from patients. While metrics such as *accuracy* and *true positive rates* may be excellent from a technical and research perspective, they are not very approachable for patients. However, metrics deduced from the *experiences* of other patients can inspire a patient to trust the AI-healthcare worker. By creating systems that support capturing such metrics, we can improve overall patient experiences with such new technology.

The advantage for healthcare workers

Making data about healthcare workers' authority, experience, and competence transparent, available, and immutable can be perceived as a privacy issue for healthcare workers. This structure also has some significant advantages for the healthcare worker. The availability of data allows us to simplify processes related to turnover, on-boarding, and mobility processes. Employers will gain an increased

ability to perform efficient background checks related to their profession. Healthcare workers will be able to share information about their background and competence efficiently. They can also have visibility into their reputation, providing a significant incentive to provide better care.

Chapter 4
Needs and Requirements

Healthcare is going through a trend of an increased virtualization of healthcare services and increased mobility of healthcare workers. During this transition, the need for structured trust in healthcare workers is becoming more critical than ever before. Patients and healthcare institutions will, to an increased degree, require evidence showing that a healthcare worker has formal authorization, is experienced, and is competent. However, gaining access to this evidence in the current healthcare system structure is challenging due to the fundamental problems of data-sharing in existing systems. To solve the problem of trust in healthcare, we propose an application that exposes our proposed sources of trust in healthcare workers: *Evidence of authority*, *Evidence of experience*, and *Evidence of competence*. We have previously stated that this evidence can serve as a ground for trust in a healthcare worker. However, these evidences offer no value to either patients or healthcare institutions if their source cannot be trusted.

We propose to deliver evidence of authority, experience, and competence by using a public blockchain. We will first explain why blockchain is suitable for this specific use-case, and explain our reasoning for using the Ethereum blockchain platform. Based on the decision to use a public blockchain, we define the most important functional requirements for our application. Additionally, the healthcare domain requires strict properties such as security and privacy, which may be hard to combine with the transparency of public blockchains. We capture such requirements in the form of quality attributes, where we mainly focus on the ones tightly related to blockchain and the scope of our application.

4.1 Using Blockchain for Trust in Healthcare

To ensure that the evidence for a healthcare worker's trust is credible, we can use a blockchain platform for storage. Blockchains offer a data-store that is immutable and highly distributed, making them easy to access. Blockchain technology has been

coined as a critical enabling technology for better data-sharing and interoperability within the healthcare industry [44], as it enables patients and healthcare institutions to share, index and control access to data in a fully distributed manner. Blockchain platforms are also highly available and easily accessible by all participants in the blockchain network.

Through the means of smart contracts, we can create a dApp running directly on a blockchain platform. We can use these smart contracts to store the evidence of a healthcare worker's authority, experience, and competence while incorporating access control mechanisms to ensure that the published data is credible. The data uploaded via these contracts are immutable, resulting in the evidence of trust to be non-reputable for healthcare workers. Additionally, since we source evidence directly from the blockchain, the healthcare worker cannot selectively hide some evidence. The non-repudiation property denies any change to the evidences once published, disabling healthcare workers to alter their evidences fraudulently. Data can be easily available to patients and healthcare institutions by using a public platform, allowing them to validate evidence on-demand and without authenticating.

The Ethereum blockchain is the most popular blockchain platform to incorporate the concept of smart contracts. As a consequence, it offers a rich suite of developer tooling, enabling rapid prototyping and testing. It, therefore, offers a compelling value proposition for creating proof-of-concept applications. By using the Ethereum platform, we can focus on developing the business logic of our blockchain application, while refraining from focusing on low-level concepts such as consensus mechanisms. Although a permissionless blockchain platform in nature, developers stand free to implement their own access control mechanisms within smart contracts to limit how data can be published. Smart contracts on the blockchain can interact with each other, enabling the creation of complex architectures with a rich set of features.

4.2 Scope

Before defining the requirements, it is crucial to clearly define the scope in which we are designing our application. We will focus on the core problem of providing trust in a healthcare worker by using blockchain technology. However, when designing such a system, some closely aligned categories of problems become apparent. We define these as out-of-scope:

Lower level blockchain technologies

This thesis aims to create a proof-of-concept application that uses an existing public blockchain. We only focus on developing applications on top of such a platform. We do not go deeply into the underlying technology used in this blockchain, unless

directly relevant to the implementation of the application. We discuss properties such as prices for usage, block compositions, and how to interact with the blockchain. However, we will not go deeply into topics such as consensus models, multi-layer scaling, block verification, and other related concepts.

Identity and access management

For our proof-of-concept application to be usable in the real world, the application should interact with existing public identification services to verify the identities of patients and healthcare workers. Although highly relevant to the security of the system, identity and access management mechanisms are not within our scope. We assume that we can verify such identities where relevant.

Key management

The blockchain's ability to provide trusted data relies on actors storing their key-pairs in a secure and portable way that can be readily loaded into our application or used externally for signatures. Securely handling such keys is a common problem for any solution which relies on cryptographic solutions. We will not provide any solutions for how these keys are stored and handled by stakeholders.

Data formats within the healthcare industry

Datasets such as PREMs, treatment descriptions, and other related healthcare data have specific formats that change over time. Therefore, we do not define any specific data formats for these datasets and define these formats as out of scope. Where we meet such formats, we handle them as high-level textual data and thus abstract away any underlying formats. If we require any specific data formats, such as a rating, we specifically define these. If we create assumptions about these datasets, they are only present for demonstration purposes, and not necessarily map to real-world data formats.

4.3 Requirements

We base our proof-of-concept on a set of functional and non-functional requirements. The functional requirements are based on use-cases for stakeholders and define *which* functionality to implement. Non-functional requirements surface the properties which the system must have, such as privacy and scalability. When architecting a system, non-functional requirements have a sizeable architectural impact, influencing *how* the system is made.

4.3.1 Functional Requirements

- A patient using a virtualized healthcare platform to talk with a healthcare worker should be able to get access to evidence of the healthcare worker's authority via the system. This evidence must show if the healthcare worker is allowed to practice. The same system should show that the healthcare worker holds no authority if this is the case.

- A patient should be able to evaluate a healthcare worker's experience by using data provided from the system. The system must allow for the generation of additional evidence of this kind by storing information about treatments.

- A patient should be able to evaluate a healthcare worker's performance by looking at data from the blockchain. The system must allow for the generation of additional evidence of this kind by storing information about the evaluations of treatments.

- All evidence served from the system should not require the patient to trust the given healthcare worker initially.

4.3.2 Quality Attributes

In addition to the functional requirements, we define some non-functional attributes of the system through quality attributes. These define the qualities the system must possess while conforming to the functional requirements. The number of quality attributes for a system is, in theory, unbounded. This section, therefore, presents the quality attributes which have the greatest architectural impact on the system.

Privacy Requirements

Priv1: Unlinkability to patients
The identity of patients must be treated as confidential. It must not be possible to link a transaction on the blockchain to a specific patient without any further knowledge from outside the blockchain.

Priv2: Anonymity of patients
The content of evaluations and treatments should not reveal the identity of patients. The encoding scheme in the dApp should not allow for such information to be publicized.

Security Requirements

Sec1: Fraudulent treatments
It should be impossible for a treatment to be published to the blockchain for

unauthorized parties. All treatments must be cryptographically approved by an entity with direct or implicit authority to publish treatments.

Sec2: Fraudulent evaluations
It should be impossible to publicize an evaluation without going through a valid treatment first. Once a treatment has a related evaluation published, it should not be possible to create another evaluation that relates to the same treatment.

Sec3: Fraudulent patients
Creating fake patients that can submit fraudulent treatments and evaluations undermines the credibility of the system. Therefore, it should be impossible for a healthcare worker to create a treatment that can be evaluated without the support of other trusted entities.

Availability requirements

A1: Addition of new governance entities
It should be possible to add new governance entities dynamically without any code changes to the original contracts on the blockchain.

A2: Denial of service of governance
Temporary downtime of governance entities should result in a low long-time disruption of the application behavior.

A3: Recoverability after authority loss
If a minority of governance entities becomes permanently unavailable or misbehaves, it should be possible to remove them. It should be possible to recover the dApp into a healthy state without interaction from the misbehaving governance entities.

Scalability Requirements

Scale1: The amount of data on the blockchain should be minimal
The blockchain is an expensive storage medium. Small data formats and encoding should be used to represent data on the blockchain.

Scale2: Decentralization of data
The system should work across a range of jurisdictions and institutions. Therefore, storing all data in a centralized manner may be impossible from both a technical and judicial perspective. The system should support a decentralized architecture where data can be fragmented across organizations.

4.3.3 Quality Attribute Scenarios

Some quality attributes have a higher architectural impact than others. In this section, we refine some of the listed quality attributes further. This is done through a scenario where we address the risk associated with it through a measure(s).

Priv1: Unlinkability to patients

A vital feature of the system is the publication of data linkable to healthcare workers on the blockchain. This data is used to provide trust in the healthcare worker from a patient's perspective. Some of the data published have a patient as a subject. These datasets can include treatments or evaluations. Due to privacy regulations and the long-lived and immutable nature of the blockchain, it should be impossible to link this data back to the given patient, as this may have real-world consequences for the patient.

Source: A patient who submits data to a blockchain

Stimulus: A transaction related to the patient is published on the blockchain. This transaction contains data related to the patient and key information such as an address.

Artifact: The smart contracts on the blockchain. Patients or treatment providers who upload transactions.

Environment: During runtime of the platform.

Response: The smart contracts should be designed in a manner where it is impossible for the body of data to reveal identifying information about a patient. The design of the system should allow for the usage of keys and addresses which cannot be linked back to a specific patient.

Response measure:

- Given a set of any public transactions, it should be computationally infeasible to link one or more of these transactions to a patient.
- Given the knowledge of a patient's identity or long-lived key, it should be computationally infeasible to find transactions on the blockchain linked to the patient.

Sec3: Fraudulent Patients

By using evaluations as a measure of trust, we create an incentive for healthcare workers to create *fake patients* who can submit evaluations which are not grounded in a real-world patient interaction with the healthcare system. This can allow healthcare workers to create a fraudulently positive view of themselves. Such a setting discredits the system as a source of trust and thus counteracts the system's intent.

Source: A healthcare worker

Stimulus: A healthcare worker tries to create a fraudulent evaluation to influence their trust positively. They do so by trying to create a *fake patient* who go through a non-existent patient interaction with the healthcare system.

Artifact: The systems authorized to publish treatments on the blockchain.

Environment: During run-time of the platform.

Response: There should be governance in place to limit the stakeholders who can publish evaluations on the blockchain. These stakeholders should apply mechanisms that ensure that these evaluations originate from real-world patients. This can be achieved by piggybacking on national identity providers, or via cryptographic proofs such as zero-knowledge-proofs. The authorized stakeholders should store cryptographic proofs off-chain, enabling them to show that treatments on the blockchain originate from real-life patients.

Response measure:

- It should be computationally infeasible for healthcare workers to fraudulently create evaluations within the system.
- Stakeholders with authorization to publish data on the blockchain-related to patient evaluations must be able to cryptographically prove to auditors that the data on the blockchain originates from a real-world patient.

A2: Recoverability after authority loss

One of the entities governing the system is breached, resulting in loss of control. As a result, it starts to act fraudulently. Actions may include issuing fraudulent treatments, and evaluations related to these.

Source: A governance entity with authority of the application.

Stimulus: The keys of the governance entity is leaked, resulting in a malicious third party getting hold of the keys to the related blockchain account.

Artifact: The smart contracts running on the blockchain, and the software running on the governance entity.

Environment: During run-time of the platform.

Response: Distibuted governance should be used to distribute responsibility between multiple entities. A voting system should be implemented, which allows a majority of governance entities to remove a malicious actor. The same voting system should be able to allow new entities to be added for governance. It should also be easy for governance entities to remove themselves from the system.

Response measure:

- Malicious or rouge governance entities should not result in long-term loss of authority for honest governance entities.
- It should be possible to maintain the robustness of the application through distributed management.
- It should always be possible to recover to a healthy state if over half of governance entities are honest.

Chapter 5
Artifact Design and Architecture

This chapter describes our design and architecture of the VerifyMed platform for providing trust in healthcare workers. Our proposed system architecture is designed to store evidence of Authority, evidence of experience, and evidence of competence on the public Ethereum blockchain platform. For these pieces of evidence to hold any legitimacy, the application incorporates a concept of governance, where a set of stakeholders cooperate to create a trusted environment on the blockchain via smart contracts. These governance entities allow us to relate the existing structure in the healthcare industry to a model on the blockchain. Patients use this trusted environment to gain evidence for trust in a healthcare worker, and publish their own experiences once the patient and healthcare worker interaction is completed. While the high-level view, as shown in Figure 5.1, is simple, the underlying system design is of high complexity.

The first contributor to increased complexity is the real-world trust relationships within the healthcare system. Our top-level model depicts a single *governance entity*.

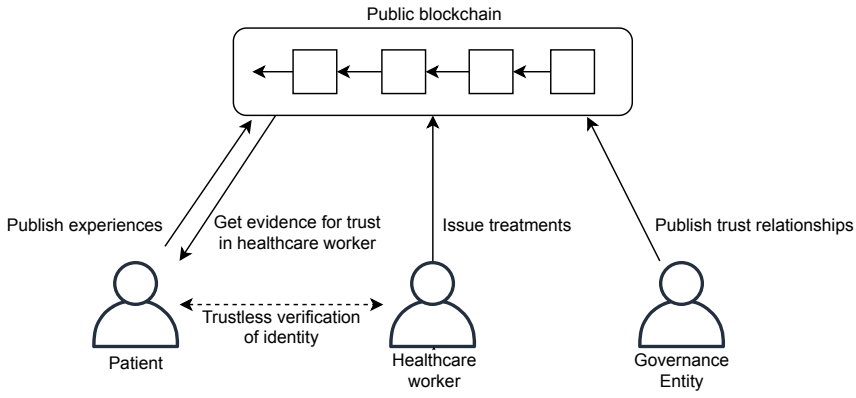


Figure 5.1: Interacting with the blockchain to gain trust in a healthcare worker

However, no such entity exists in the real world. The trust relationships within the healthcare industry include a large set of different organizational entities. These entities hold specific responsibilities, and they can only together create overall trust in the healthcare system. Our system architecture includes multiple organizational entities, and we capture the trust relationships between these on the blockchain. We will present these stakeholders, along with the model for trust between them.

The previously defined quality attributes also include many other challenges to be solved. These include patient privacy, prevention of fraudulent patients and healthcare workers, and scalability considerations. We can only address these quality attributes through architectural choices. After describing our overall system architecture and our choices, we will break this down into procedures and subsections to show how the architecture addresses these requirements.

5.1 Modeling Evidence for Trust

We have previously defined our novel framework for categorizing evidence for trust in a healthcare worker. This section describes how these evidence categories materialize in our design.

5.1.1 Evidence of Authority

The first evidence for trust in healthcare workers is evidence of authority. This evidence consists of the formal credentials which allow healthcare workers to practice. By providing this evidence on a blockchain, patients or any other interested stakeholder can access these freely. If the patient can confirm the link between a healthcare worker and the evidence of authority on the blockchain, they should trust that the healthcare worker has formal authorization. In practice, we choose to model the evidence of authority as two different statements which the healthcare worker wants to prove:

1. The healthcare worker is currently in possession of a valid License for Health Personnel and is thus formally qualified to practice in healthcare.
2. The healthcare worker is formally associated with an authorized healthcare facility.

The healthcare worker cannot fulfill these statements alone. They are instead statements of trust from other organizational entities that are deemed trusted themselves. This structure of entities and their trust relationships quickly serves as the foundation of trust in the system. We define the following stakeholders as present this hierarchy of trust:

- **Authorities** is top-level healthcare authorities responsible for the formal authorization of healthcare institutions, educational facilities, and other organizations that provide services related to providing healthcare. Organizations with such authorities are typically national health directorates.
- **License Issuers** are organizations that are responsible for the formal authorization of healthcare workers. They are responsible for the background check of applicant healthcare workers requiring a license for healthcare personnel. They verify that the healthcare worker has the competence and background required to practice. If that is the case, they choose to issue such a license and thus establish a trust relationship with the healthcare worker. Such organizations are often units within a national health directorate.
- **License Providers** are authorized healthcare facilities responsible for the practice of the healthcare worker on a day-to-day basis. These facilities are under continuous evaluation by authorities and have to ensure their associated healthcare workers' competence. Such organizations can include hospitals or clinics.
- **Treatment Providers** are healthcare service providers who are responsible for facilitating interactions between patients and healthcare workers. These stakeholders may be perceived as similar to license providers, as they will, in many cases, be the same organizations, such as clinics or hospitals. However, these stakeholders are, in some cases, different. For example, when a clinic healthcare worker interacts with a patient via a virtualized healthcare platform. In this case, the clinic serves as the license provider responsible for the healthcare worker's day-to-day practice. In contrast, the virtualized healthcare platform is the treatment provider which the healthcare worker and patient interact through.
- **Licenses** is the representation of healthcare workers within our trust model. A license can only be created by a license issuer, and is tied to credentials (a key-pair) in possession of the healthcare worker. Once issued, it may be transferred between License Providers, License Issuers, and associated with additional Treatment Providers if these stakeholders agree to these movements.

Together, these stakeholders can interact to form a complete trust hierarchy. We show this hierarchy of trust in Figure 5.2. This model is captured via smart contracts deployed to the blockchain ledger, storing data about stakeholders and their trust relationships. The top level is our authorities, where the smart contract creator start as an authority as a bootstrap. These authorities organize themselves via a distributed governance protocol. We propose to use simple majority voting for this purpose, but implementations stand free to choose any protocol of their liking.

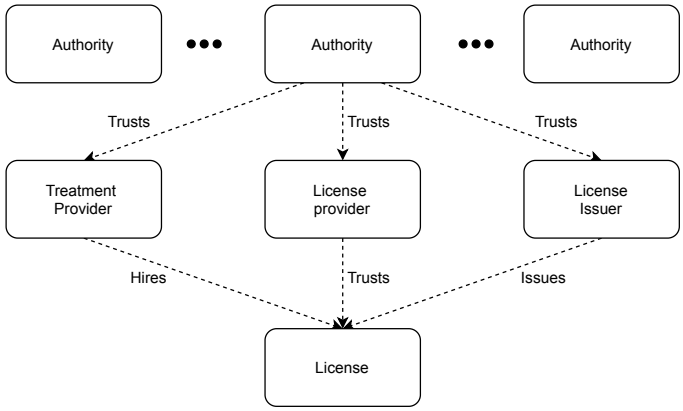


Figure 5.2: A trust model for stakeholders in the healthcare industry

Authorities may, in turn, place their trust in License Issuers, License Providers, and Treatment Providers. These trust relationships should be justified in real-world decisions and processes to deem them eligible to hold such a responsibility. If an authority deems their trusted stakeholders unfit of their trust, they can choose to revoke this trust relation.

Validating the two statements for proof of authority is done by asserting the current state in the trust hierarchy as represented on the blockchain. Checking if a healthcare worker has a valid License for Health Personnel is done by inspecting their license representation and validate that a trust relationship exists from a License Issuer. An authority must also trust the given license issuer. A similar process is performed to validate the second statement. In this case, we validate if the license is trusted by trusted License provider.

5.1.2 Evidence of Experience

The second evidence for trust in healthcare workers is their experience. Depending on the context in which the patient meets a healthcare worker, experience within a relevant field may be of high importance to ensure that the healthcare worker can deliver the required care. Metrics for experience come in either qualitative or quantitative forms. The healthcare worker can convey qualitative evidence through certifications. They can convey quantitative evidence via metrics such as the number of treatments performed by the healthcare worker. To model proof of experience, we choose to focus on quantitative metrics.

Our model's goal is to convey information about the number of treatments performed by the healthcare worker. We can create evidence of authority through a formal

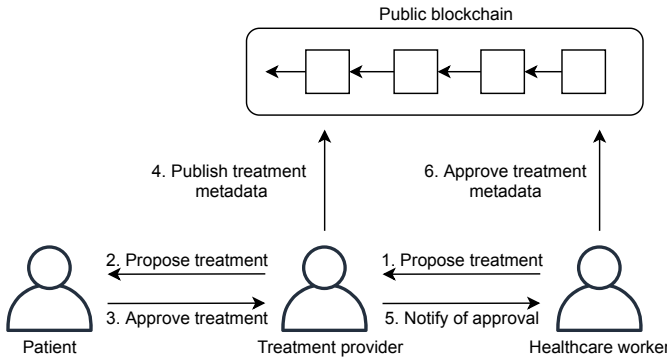


Figure 5.3: A model for generating evidence of the experience of healthcare workers

model where parties establish trust relationships between each other. In contrast, we generate evidence of experience through patient and healthcare worker interactions. Each new interaction results in a new treatment, thus forming evidence for future patients who want to interact with the healthcare worker.

Figure 5.3 shows our model for publishing treatment information on the blockchain. During the patient and healthcare worker interaction, the treatment provider is responsible for conveying information about treatments recommended by a healthcare worker to the patient. Once approved by the patient, we store the full content of the treatment at the treatment provider. The treatment provider will then publish metadata about the treatment to the blockchain. This metadata is, in turn, approved publicly by the healthcare worker, thus forming a public link from the healthcare worker to the treatment. Over time, this process will generate a public log capturing metadata about treatments performed by a healthcare worker, which serves as the proof of experience.

5.1.3 Evidence of Competence

While a quantitative metric like the number of treatments can be an evidence for experience, it does not represent the quality of these treatments. PREMs are a standardized way to measure patient experience from the treatment. By translating these experiences into quantitative metrics, and publishing them on the blockchain, we can measure the experienced quality of a treatment. We show this general process in Figure 5.4, where the patient interacts directly with the blockchain to publish their experience related to a treatment they have gone through. Since these treatments are linked to a healthcare worker, we can use them as a proxy for evaluating a healthcare worker's competence. The log of treatment metadata with corresponding experience measures on the blockchain serves as proof of competence.

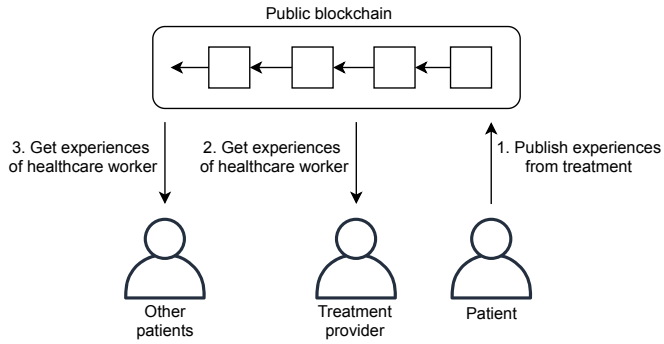


Figure 5.4: A model for generating evidence of the competence of healthcare workers

5.2 System Architecture

Our models present a high-level view of the concepts applied to generate trust in a healthcare worker. These models show the overall methodology for processes and convey information about the critical datasets required to provide trust. However, the models cannot be implemented as a software artifact directly and do not take all the required quality attributes into account. To handle these two cases, we create a complete software architecture that incorporates these models. The software architecture takes the underlying technology into account, along with the scope, functional requirements, and quality attributes presented previously.

We divide our software architecture into two main components. The first of these is a set of smart contracts. Together, these form a dApp running on the Ethereum blockchain platform. We will refer to this section as the *on-chain* part of our application. The other major part of the system is the artifacts outside of the blockchain platform that interacts with this dApp. Components in this part of the system are traditional software artifacts with a run-time presence on stakeholder owned server. We refer to this part as the *off-chain* component in our system.

Interaction between the off-chain and on-chain parts of the software architecture happens through transactions and state queries. Off-chain components create transactions. These are sent to a node in the Blockchain network. Transactions allow for the state in the contracts to be mutated. This process yields no return value from the network, so we fetch the updated state asynchronously. If the off-chain component wants to query the current state of the on-chain application, they will send a query to a node in the Ethereum blockchain network. This query will immediately return the current state of the contract $\sigma[s]$ without any additions to the Ethereum blockchain platform, and therefore has no related cost.

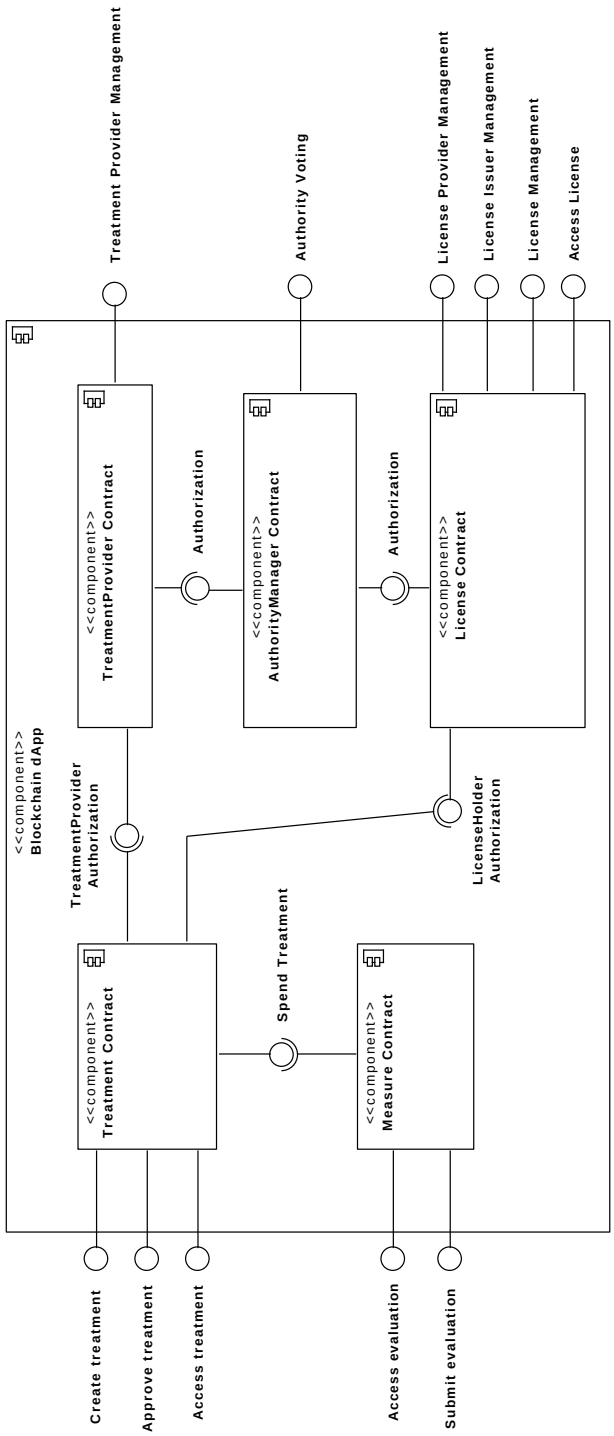


Figure 5.5: A component diagram representing the dApp running on the blockchain.

5.2.1 On-Chain Application Part

Our models include information about licenses, trust relationships between stakeholders, metadata about treatments, and their evaluations. We store this information on the Ethereum blockchain via smart contracts. The platform allows us to create a global state $\sigma[s]$ in the context of a smart contract, where this state can be mutated by adding transactions to the blockchain ledger. The smart contracts include logic which dictates how this state can be mutated, and on which condition these mutations can happen.

Our architecture includes a set of five different smart contracts that hold specific responsibilities. The contracts interact with each other during transaction executions, and the full consortium of smart contracts have all functionality required to support the processes in our models. We show the full consortium of these contracts in Figure 5.5, where the smart contracts have the following responsibilities:

AuthorityManager Contract is a smart contract responsible for storing information about authorities. It maintains a list of authorities and implements a distributed governance protocol allowing for the addition and removal of these.

TreatmentProvider Contract is a smart contract responsible for storing information about treatment providers. It maintains a list of treatment providers, along with their trust relationships. The contract exposes interfaces for checking if a treatment provider is trusted, registering an address as a treatment provider, and for authorities to manage their trust in treatment providers.

License Contract is a smart contract for storing information related to Licenses and their corresponding governance entities. Responsibilities include maintaining a list of Licenses, License Providers, License Issuers, and the trust relationships, both between each other and to authorities. The contract contains the logic to deduce if a license is trusted based on these trust relationships.

Treatment Contract is a smart contract responsible for storing treatment metadata. It maintains a list with this data along with links to relevant entities such as the approving license and treatment provider.

Measure Contract is a smart contract responsible for storing information about summarized patient experiences. It interacts with the Treatment Contract to establish a link between the treatment and the evaluation while ensuring consistency rules, such as only allowing a single evaluation of each treatment.

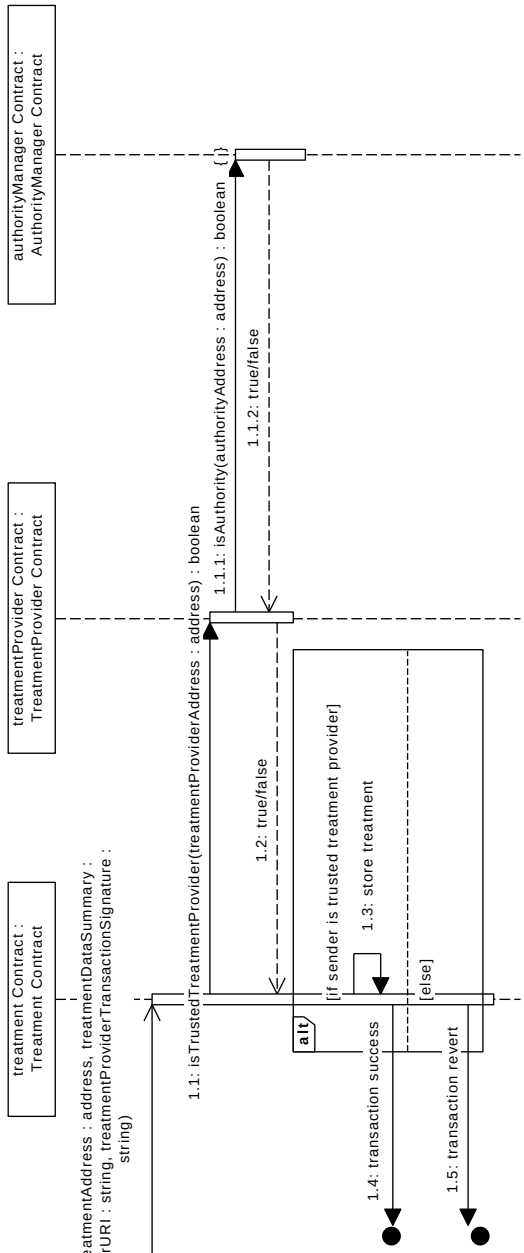


Figure 5.6: A sequence diagram showing authentication flow when submitting a create treatment transaction.

Access Control in Smart Contracts

We have designed our architecture to use a public and permissionless blockchain platform. Such platforms allow for any user to submit a transaction targeted for our smart contracts. Therefore, we must apply access control schemes to govern the state in our smart contracts. Adding these prevents trust relationships, treatments, and evaluations from being fraudulently published to our smart contracts. These schemes must only allow authorized senders to change the state within our blockchain application. We achieve this behavior by implementing access control logic within the smart contracts themselves.

The access control scheme implemented can be described as a RBAC scheme where accounts interacting with the blockchain must hold a particular role within our dApp to perform such an action. Examples of access control policies in the on-chain part of our architecture include:

- Only existing authorities may interact with the distributed governance protocol;
- Only existing authorities may add trust in a treatment provider, license provider, or license issuer;
- Only treatment providers trusted by an authority may add treatments;
- Evaluations can only be created by the patient, who is the subject in a treatment.

These policies are part of the internal contract logic and enforced through interaction between the contracts. We show an example of this in Figure 5.6, where the treatment contract checks if the sender of the transaction is indeed a trusted treatment provider. We notice how this check is propagated to the authority contract. Thus, if an authority is removed, any trusted descendants will lose their authority to create treatments immediately.

5.2.2 Off-Chain Application Part

The goal for the off-chain part of our software architecture is first to generate and use evidence for trust in a healthcare worker. While the on-chain smart contracts provide interfaces to store and evaluate this evidence, the off-chain components generate this evidence and create transactions to publish it. Figure 5.7 shows an overview of the complete software architecture with all components. Within this view, all components except for the *Blockchain dApp* are off-chain software artifacts. Here, the *Blockchain dApp* represents the Ethereum blockchain network, where the exposed interfaces represent sets of transactions for interacting with the smart contracts for querying their state.

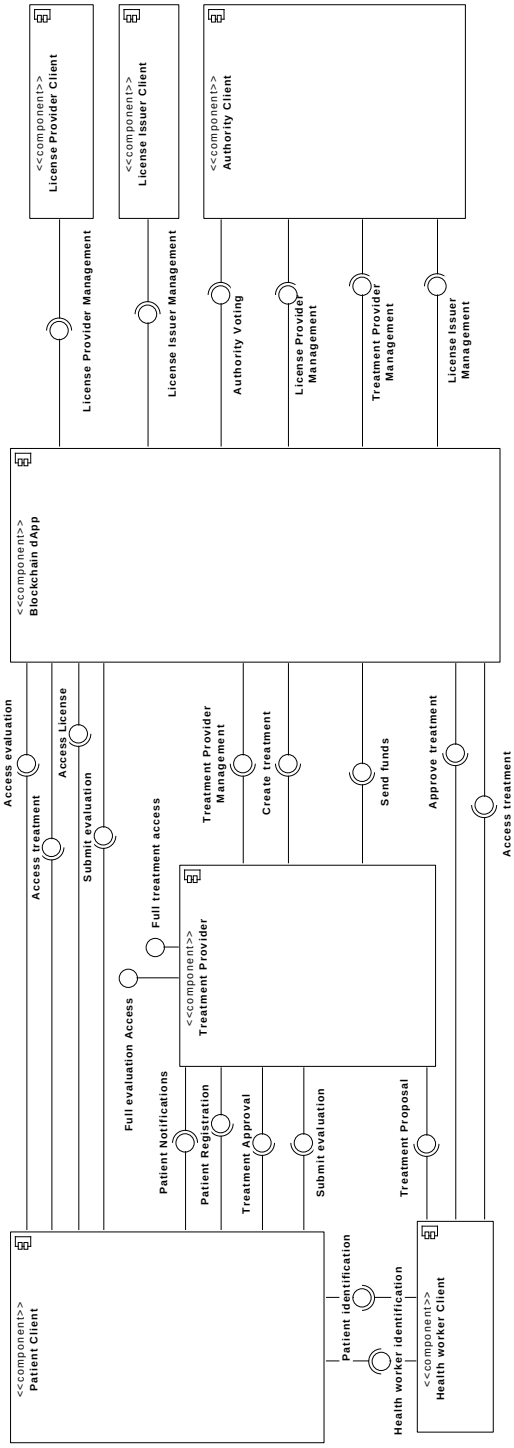


Figure 5.7: A component diagram capturing a top level view of the off-chain part of our system architecture

The flow of information generally follows the same pattern as described in our high-level models, with exceptions for some processes. Decisions for addressing our requirements cause changes in this behavior. This allows us to provide properties such as unlinkability to patients, limited on-chain storage, and a reduced number of transactions.

Off-Chain Components

The main responsibilities of the off-chain components within our software architecture are to publish data to smart contracts. However, the mapping from off-chain to on-chain components is not one-to-one. The off-chain components often communicate with multiple contracts and sometimes perform procedures isolated to the off-chain environment. The application also keeps a distributed architecture in mind. The number of instances for off-chain is arbitrary. The different components are also assumed to be under the control of different organizations with either good or bad intentions. Therefore, we present briefly each of these components, along with their responsibilities.

Authority Client is a component under the control of authorities, the top level in the trust hierarchy. It interacts with other authorities via the distributed governance protocol implemented within the Authority Contract. It can send transactions to the License Contract to add or remove trust in License Issuers and Providers. It can also send transactions to the Treatment Provider Contract to add and remove trust in treatment providers. This component never interacts with the other off-chain components directly from a technical viewpoint, but trust relationships should be justified in real-world interactions.

License Issuer Client is a component responsible for managing licenses on the blockchain. It can submit transactions to the License Contract for issuing licenses to addresses. It can also accept movements of licenses to themselves, as may be desirable when a healthcare worker takes a mobility decision to move jurisdictions. In the same manner as the Authority Client, the License Issuer does not interact directly with the other off-chain components but takes all actions via the contracts in the on-chain part.

License Provider Client administer trust relationships between the License Provider and licenses. It is responsible for creating transactions to publish such relationships on the blockchain. It can also accept proposed movements of licenses to them in the case of healthcare worker turnover and mobility. The License Provider should ensure that the published trust relationships are rooted in real-world processes such as performance reviews and evaluations of the healthcare worker.

Treatment Provider has responsibility for facilitating the interaction between the patient and the healthcare worker and for storing data from this interaction. It authenticates patients via some identity provider to ensure the legitimacy of the patient. The specifics of this process fall under identity and access management of patients and is, therefore, out of scope. Once the patient approves treatments, the treatment provider is responsible for publishing the summary of these to the blockchain by sending a transaction to the Treatment Contract.

Healthcare Worker Client is a device owned by the healthcare provider during operational procedures, for example, a computer in their office. The component is responsible for interacting with patients via treatment providers. It submits proposals for treatments to patients via the treatment provider. Once accepted by the patient, the healthcare worker client can verify the authenticity of the treatment metadata on the blockchain, and approve this if correct.

Patient Client is a device owned by patients who are responsible for the patient side of an interaction with a healthcare worker. It authenticates itself with the treatment provider responsible for the interaction and approves treatments suggested by the healthcare worker. Once completed, the Patient Client is used to submit an evaluation related to a treatment.

5.3 Addressing Quality Attributes

The main intent of the VerifyMed platform is to provide evidence for trust in a healthcare worker in a virtualized setting. Nevertheless, the platform must respect attributes such as patient privacy. We have previously defined a set of quality attributes, each with an impact on the overall system architecture. This section describes how the architecture addresses these quality attributes.

5.3.1 Privacy Requirements

Priv1: Unlinkability to patients Ethereum accounts and addresses can be considered pseudonymous. These identifiers cannot be linked directly to a real-world identity. However, if one can link this address to a patient, it is possible to gain information about all transactions submitted to the blockchain by that address. If these transactions contain treatment metadata, it is possible to create a view of a patient's medical history, which cannot be public. Therefore, the VerifyMed architecture does avoid this practice for patients.

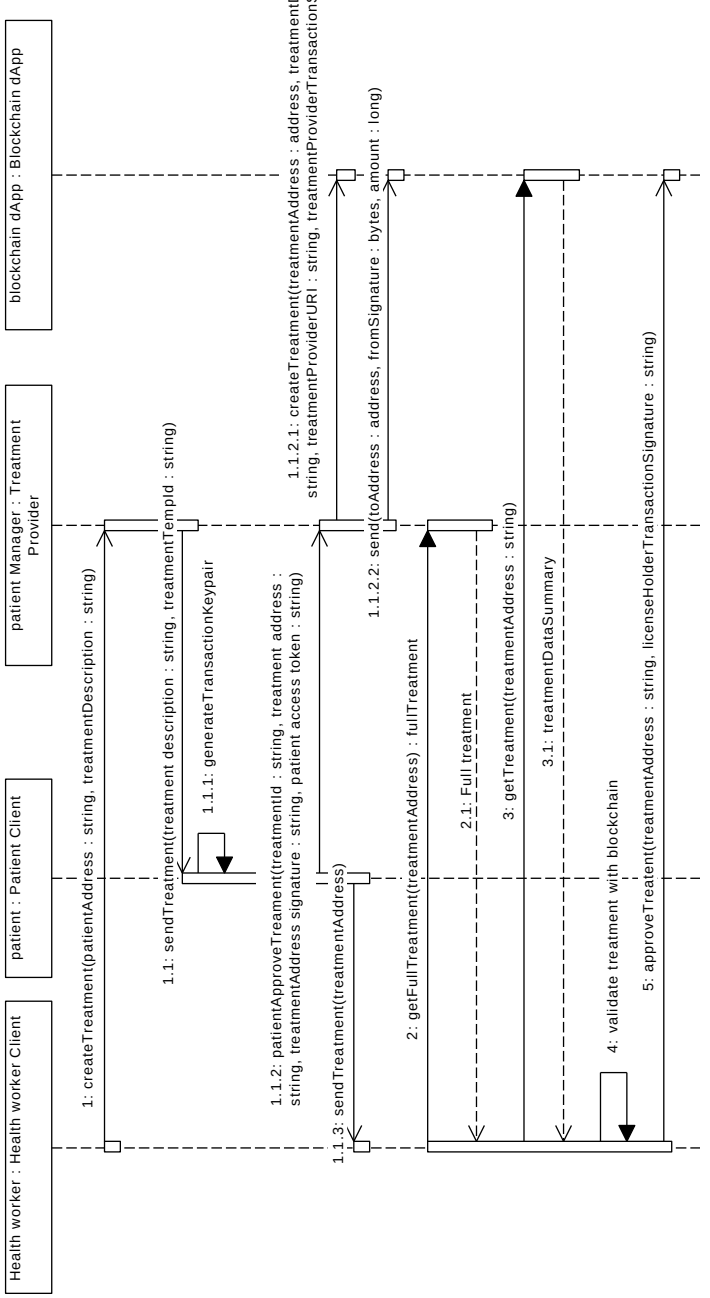


Figure 5.8: A sequence diagram showing how off-chain components interact during treatment creation

One solution for this problem is to use one-time ECDSA keys for each treatment, which are generated independently from the patients' long-time key-pair. By using a new and independent key for each transaction, we cannot link them to a patient. Furthermore, if the one-time key-pair never interacts with the long-time key-pair on the blockchain, they cannot be linked together via traffic analysis attacks [46]. However, such keys have a usability flaw in the Ethereum ecosystem, as we cannot use such key-pairs to create transactions before some Ether is sent to the account. This Ether cannot be sent directly from the patient, as this creates an association between keys.

The VerifyMed architecture uses both a trusted intermediary and one-time keys for treatments. During treatment approval, as shown in Figure 5.8, the patient generates a one-time ECDSA key-pair. The patient approves treatments by signing with both their long-lived identity key and the generated one-time key, forming a multi-signature on the treatment. Both these signatures are sent to the treatment provider. The treatment provider will only include the address of the one-time key in the treatment's metadata, thus preventing linkability from patient to treatments. If the patient ever wants to prove that they took part in treatment to any other entity, they can use a zero-knowledge proof to show that they own the one-time secret key related to the given treatment.

After publishing the treatment metadata to the blockchain, the treatment provider will send a small amount of Ether to the address of the one-time key generated by the patient. This process allows the patient to submit an evaluation independently while remaining unlinked from the patient. Thus, to submit an evaluation, he/she can simply create a transaction by signing with his/her one-time key-pair.

Priv2: Anonymity of patients To prevent a cryptographic link from evaluations to the patient, we apply one-time keys. However, the content within treatments and evaluations may nevertheless reveal such a link. To prevent such information from being leaked on the blockchain, the on-chain contracts will only accept summaries of both treatments and evaluations. The evaluation only includes quantifiable metrics such as scores and does not support any type of data that may reveal an identity. Treatments only include essential metadata about the complete treatments, for example, the general category of it. It includes a hash of the complete treatment, thus providing an integrity proof of the complete treatment data as stored elsewhere.

5.3.2 Security Requirements

Sec1: Fraudulent treatments As treatments are essential to provide both evidence of experience and evidence of competence, the system should only allow for legitimate treatments to be submitted. In order to enforce this behavior, treatments

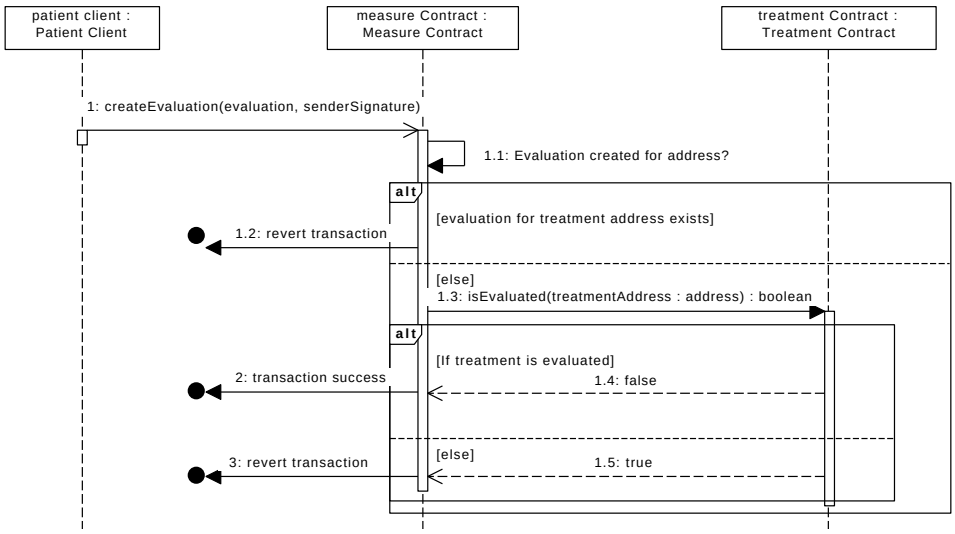


Figure 5.9: A sequence diagram showing the authentication flow when submitting a evaluate treatment transaction.

can only be published to the blockchain by treatment providers who are trusted by an authority. This access control mechanism is enforced as code within the treatment contract. Stakeholders with something to gain from fraudulent treatments (healthcare workers) cannot submit treatments themselves and must rely on another stakeholder, namely the treatment provider.

Sec2: Fraudulent evaluations We prevent the creation of fraudulent evaluations by enforcing a one-to-one mapping between treatments and evaluations. Treatment metadata includes the address of the one-time key generated by a patient. When evaluating a treatment, the patient uses this key to create the evaluation transaction, which is processed by the evaluation contract. The sending address is validated to check if no existing evaluation exists with the sending address and if this address has indeed been used previously for a treatment. We show this overall process in Figure 5.9, where we show the interaction between the patient client, measure contract, and treatment contract.

Sec3: Fraudulent Patients Treatment providers are responsible for the publication of treatments on behalf of the patient. This pattern is advantageous for providing patient anonymity on the blockchain. This component can also serve as a Policy Enforcement Point [47] to prevent fraudulent patients from submitting treatments. The treatment provider is responsible for validating that the patients

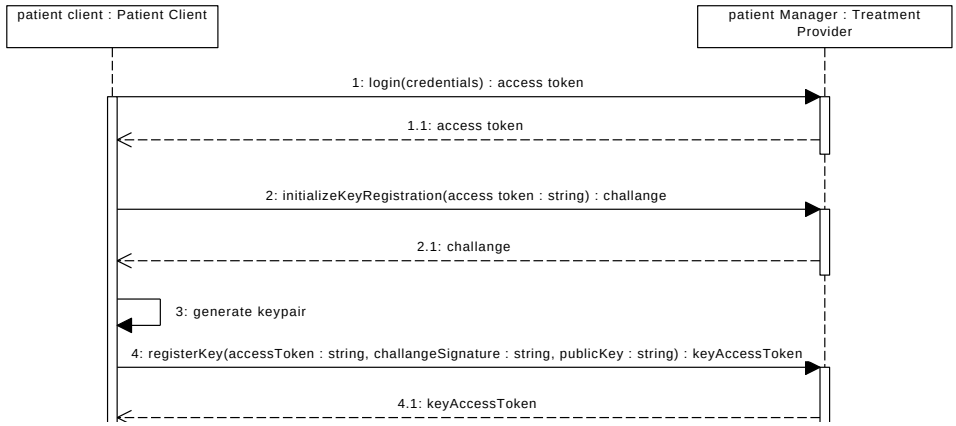


Figure 5.10: A sequence diagram showing how a patient authenticates themselves to gain access to services via the treatment provider.

interacting with it are indeed real-world patients. This validation can happen by consulting an (out-of-scope) national identity provider. We show an example of such a procedure in Figure 5.10, where the initial login procedures contain credentials for such a provider. The patient uses the created access token for subsequent interactions with the treatment provider. Here, the patient also uses a zero-knowledge-proof to show ownership of a public key, granting a token for showing ownership of this key in later procedures. The treatment provider stores all data related to the treatment to prove that it originated from the given patient.

5.3.3 Availability Requirements

The top-level in our hierarchy of governance entities is the authority. They interact with each other via a distributed governance protocol, allowing for the addition and removal of such entities. VerifyMed uses simple voting as its governance protocol. We implement this protocol as logic within the authority contract, which we may interact with by sending transactions. Authorities which are within the pool of authorities may do the following:

1. Propose a vote for adding a new authority to the pool of authorities.
2. Vote on a proposal for adding a new authority to the pool of authorities.
3. Enact a proposal for adding a new authority to the pool of authorities. We can only perform this action if over 50% of authorities in the pool have voted for the proposal.

4. Propose a vote for removing an existing authority from the pool of authorities.

5. Vote on a proposal for removing an existing authority from the pool of authorities.

6. Enact a proposal removing an existing authority from the pool of authorities. We can only perform this action if over 50% of authorities in the pool have voted for the proposal.

This protocol allows for the addition of new governance entities (Addressing **A1**), and removal of authorities who act fraudulently from the perspective of over 50% of authorities in the pool (addressing **A3**). Additionally, the protocol does not require the live-ness of authorities as it persists state even if an authority is going through downtime. Thus, an attacker seeking to gain a majority vote cannot use downtime as a tool (addressing **A2**).

5.3.4 Scalability Requirements

Scale1: The amount of data on the blockchain should be minimal Our smart contracts accept a set of transactions for storing state on the blockchain. Adding data to such a state is the main contributor towards high gas costs for transactions, and should thus be minimized. We can reduce the number of state updates by either minimizing the number of transactions sent to the blockchain or by reducing each transaction's size. We perform the latter by minimizing the amount of textual data on the blockchain. Instead, each contract only accepts hashes with a bit-length of 256, which serves as an integrity check for data stored off-chain.

Scale2: Decentralization of data The treatment providers are responsible for storing data created during a patient and healthcare worker interaction. While metadata about treatments are stored on the blockchain, the full treatment data is stored in the treatment provider. Thus, data about treatments is stored in a decentralized manner across treatment providers. Utilizing this pattern is problematic, as we have previously stated that data sharing is a problem within the healthcare industry.

To make treatment data discoverable and accessible, we can use the metadata on the blockchain. The treatment address is linked to a patient. Thus, the treatment provider can give access to any stakeholder who can prove that they own the corresponding private key. We can do this through a zero-knowledge proof. Healthcare workers can use a similar process for showing ownership of their license. Finally, we can make the dataset of full treatments discoverable by adding a resource locator to the treatment metadata, indicating where the full treatment is stored. If these

discovery and access control mechanisms are taken into account, we can state that the architecture supports a distributed architecture.

A healthcare worker who wants to access all data related to them may follow a simple procedure. First, they query the blockchain for all treatment metadata related to their licenses. Second, they use the locators in this metadata to find the treatment provider storing the full treatment. Finally, they prove ownership of their license to the respective treatment provider though a zero-knowledge proof, gaining access to the treatments after that. This same procedure can be used by patients to access their data.

Chapter 6
Application Implementation

This chapter describes the implemented proof-of-concept application representing the VerifyMed architecture described in Section 5.2. The implemented application does not have a run-time presence directly corresponding to the architecture. Instead, we create a simple client-server web application that simulates the processes and components as designed in the system architecture. The overall application structure is shown in Figure 6.1. This approach does not allow us to scale beyond local testing of the platform and is not suited for production use cases. Nevertheless, the reduced complexity of this approach drastically reduces development time while enabling us to assemble metrics, evaluate the architecture for design faults, and evaluate the system's properties.

During this chapter, we will present details for the application implementation and the applied technology. We focus on technological choices for each service and how they interact. Code is periodically presented for showing examples of behavior, or for describing the usage of some technology. In addition to the description in this chapter, we provide an admin guide for application setup in Appendix B, allowing the reader to set up the system locally. We also refer to Appendix A for a complete usage guide for application UI.

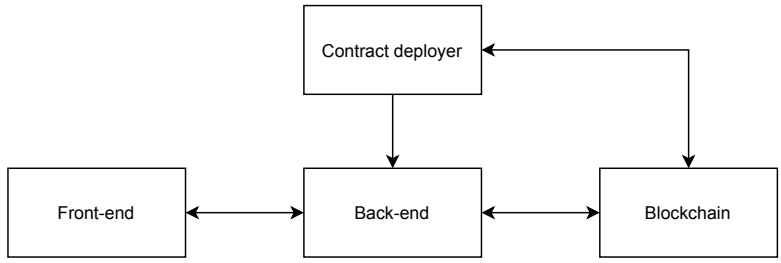


Figure 6.1: An overview of the run-time presence for our implemented services

6.1 Blockchain Service

A node from the live Ethereum network represents the blockchain component in our application. However, for development and testing purposes, we use the Ganache-cli[1] tool to set up such a node. The tool provides a Command Line Interface (CLI) for setting up a local single-node Ethereum network, bootstrapped with a set of accounts with a configurable amount of Ether. It implements the Ethereum specification[2], and uses a standard Ethereum node internally to replicate the Ethereum network. Therefore, it can be used for reliable testing, while remaining swappable with a node from the live Ethereum network if a production deployment is desired. By using such a setup, we can simulate the real-world behavior of the Ethereum network, without paying for the Ether required for using the live network.

6.2 Contracts Service

The Contracts Service contains the source code for the smart contracts composing our dApp on the Ethereum blockchain. Additionally, it is responsible for:

1. Compiling smart contracts into EVM bytecode.
2. Deploying smart contracts on the blockchain.
3. Unit testing the smart contracts.
4. Exporting the Application Binary Interface of each contract to the API server.
5. Exporting contract addresses to the API server.

Technology

The following list of technologies are used for creating our smart contracts and the supporting functionality:

- **Solidity** [31]: a programming language for writing Ethereum smart contracts;
- **Solc**[3]: the Solidity compiler for compilation to EVM bytecode;
- **Truffle**[4]: for compiling, deploying and testing smart contracts.

[1] https://github.com/trufflesuite/ganache-cli, version *6.9.0*
[2] https://github.com/ethereum/go-ethereum/wiki/Ethereum-Specification, commit *b28b61c*
[3] https://github.com/ethereum/solc-js, version *0.6.1*
[4] https://github.com/trufflesuite/truffle, version *5.1.8*

Solidity Smart Contracts

As modeled in Section 5.2.1, our application consists of five different smart contracts. We write each of these contracts in the Solidity language. Solidity is object-oriented, and uses patterns such as inheritance. The interfaces between contracts can, therefore, be defined individually as interfaces implemented by each contract. In brief, we implement the following smart contracts:

- **IAuthorization**
- **ILicenseProviderManager**
- **ITreatmentProviderManager**
- **IMeasure**
- **ITreatment**
- **AuthorityManager** is **IAuthorization**
- **LicenseProvide** is **ILicenseProviderManager**
- **TreatmentProvider** is **ITreatmentProviderManager**
- **Measure** is **IMeasure**
- **Treatment** is **ITreatment**

The simplest of these contracts are the **IMeasure** and **Measure** contracts. The interface IMeasure defines all the functions used to interact with the measure contract, along with data formats to be used within it:

```
1  pragma solidity ^0.6.1;
2
3  interface IMeasure {
4
5      // The definition of an evaluation
6      struct MeasureInstance {
7          uint8 rating; // Representation of the rating given by the
               patient
8          bytes32 fullMeasureHash; // SHA3 Hash of the extended
               evaluation
9          string fullMeasureURL; // Location of the extended evaluation
10     }
11
12     // Function for storing a new evaluation on the blockchain.
13     function createMeasure(
```

```
14          uint8 _rating,
15          bytes32 _fullMeasureHash,
16          string calldata _fullMeasureURL
17      ) external;
18
19      // Get the evaluation related to a specific treatment if it exists
20      function getMeasureForTreatment(address _treatment)
21          external
22          view
23          returns (uint8, bytes32, string memory);
24
25  }
```

Listing 6.1: The IMeasure.sol contract

The **Measure** contract must instance all the functions as defined in **IMeasure**. Additionally, the contract is responsible for storing data about evaluations and to enforce the access control mechanisms as described in our architectural model in Section 5.3.2 and further visualized in Figure 5.9. Although simple, it shows how we can create smart contracts that enforce access control mechanisms, store data, interact with other contracts, and provide functions for data access.

```
1
2   pragma solidity ^0.6.1;
3   import "./iface/ITreatment.sol";
4   import "./iface/IMeasure.sol";
5
6   contract Measure is IMeasure {
7       ITreatment treatmentContract; // Stores a pointer to the treatment
            contract
8
9       // A mapping from treatment addresses to evaluations
10      mapping(address => MeasureInstance) measures;
11
12      // Set the pointer to the treatment contract during contract
            creation. This argument is passed in the contract creation
            transaction
13      constructor(address _treatmentContractAddress) public {
14          treatmentContract = ITreatment(_treatmentContractAddress);
15      }
16
17
18      //  Function  for  storing a new  evaluation  on the  blockchain.
            The sender of the transaction should be a treatment address.
19      function createMeasure(
20          uint8 _rating,
21          bytes32 _fullMeasureHash,
22          string calldata _fullMeasureURL
23      ) external override {
```

```
24          // Check if the treatment has been evaluated previously, and
                revert  transaction if this is the case.
25          require(
26              !treatmentContract.isTreatmentSpent(msg.sender),
27              "Treatment is allready evaluated"
28          );
29          // Check if the sending address is indeed a treatment address,
                and revert if this is not the case.
30          require(
31              treatmentContract.isTreatmentInstanced(msg.sender),
32              "Treatment is not instanced"
33          );
34
35          // Store the information about the rating
36          measures[msg.sender].rating = _rating;
37          measures[msg.sender].fullMeasureHash = _fullMeasureHash;
38          measures[msg.sender].fullMeasureURL = _fullMeasureURL;
39
40          // Call the treatment contract to ensure that the treatment
                cannot be re-evaluated.
41          treatmentContract.spendTreatment(msg.sender);
42      }
43
44      // Get the evaluation related to a specific treatment if it exists.
45      function getMeasureForTreatment(address _treatment)
46          external
47          view
48          override
49          returns (uint8, bytes32, string memory)
50      {
51          // Return the value directly from our storage. If such a
                mapping does not exist, it will return default values for
                each data type. Clients must check if the returned data is
                default values.
52          return (
53              measures[_treatment].rating,
54              measures[_treatment].fullMeasureHash,
55              measures[_treatment].fullMeasureURL
56          );
57      }
58
59  }
```

Listing 6.2: The Measure.sol contract

We repeatedly use the patterns shown in the **Measure** contract during the implementation of the remaining contracts. However, we reemphasize that the **IMeasure** and **Measure** contracts are the simplest and most understandable contracts, and therefore do not introduce all the important concepts of Solidity. Additional concepts of importance includes *Events* and *Modifiers*. *Events* are objects that can be emitted as a result of a contract call. They are stored in the transaction

receipts within blocks. *Modifiers* allow us to create reusable code, which can be injected into functions. We typically use these for access control mechanisms. To illustrate both concepts, we present a snippet from the **AuthorityManager** contract.

```
contract AuthorityManager is IAuthorization {
    struct Proposal {
        uint256 proposalType;
        address target;
        bool isActive;
        address[] voters;
        mapping(address => bool) hasVoted;
    }

    ...

    mapping(address => bool) public authorities;

    modifier authorized() {
        require(authorities[msg.sender], "Unauthorized");
        _; // The injection point for the function using the modifier
    }

    event ProposalEvent(
        address _proposer,
        address _proposalSubject,
        uint256 _proposalType,
        uint256 _proposalID
    );

    ...
    // propose a vote for the addition or removal of an authority
    function propose(uint256 _proposalType, address _targetAddress)
        public
        authorized // Refers to the authorized modifier
    {
        ...
        // Emit an event to be stored in the transaction receipt
        emit ProposalEvent(
            msg.sender,
            _targetAddress,
            _proposalType,
            proposalID
        );
    }
    ...
}
```

Listing 6.3: A snippet from the AuthorityManager contract

The remaining smart contracts use the same overall structure and patterns as presented. However, we refrain from presenting them all in detail, as they surpass 1200 lines of Solidity code. We refer the reader to the open-source GitHub repository for the full implementation of the remaining smart contracts.

Compiling, Testing and Deploying Smart Contracts

Truffle is a general-purpose toolchain for the development, testing, and deployment of smart contracts. It simplifies contract development by allowing developers to write deployment and testing scripts in JavaScript running on the NodeJs run-time environment. Developers use the toolchain via a CLI with commands such as `truffle compile`, `truffle test` and `truffle migrate`

Solidity compilation The first feature provided by Truffle is smart contract compilation. It inspects all the Solidity source code to be compiled, and uses the correct **Solc** Solidity compiler based on the version preamble in the contract. This process creates EVM bytecode, which we can deploy to the blockchain, and a ABI describing how to encode transactions indented for the smart contract. We use this feature by running the `truffle compile` command.

Solidity testing To test solidity source code, we can write tests with the truffle framework, and run these with the `truffle test` command. Truffle will manage a lifecycle where it first deploys a contract and then tests it by submitting transactions to the blockchain. We validate the output from out transactions by either querying the node for contract state or by parsing the generated transaction receipts. We illustrate this process with some of our code for testing the AuthorityManager contract:

```
1  const AuthorityManagement =artifacts.require("./AuthorityManager.sol");
2  const assert = require("chai").assert;
3  const truffleAssert = require("truffle-assertions");
4
5  contract("AuthorityManager", accounts => {
6    let authorityManagementInstance;
7
8    before(async () => {
9      authorityManagementInstance = await AuthorityManagement.deployed();
10   });
11
12   it("Account0 should be only authority", async () => {
13     const authorities = await authorityManagementInstance.
           getAuthorities();
14     assert.ok(authorities.includes(accounts[0]));
15   });
```

```
16
17   it("Account2 should be unable to propose a vote", async () => {
18     await truffleAssert.fails(
19       authorityManagementInstance.propose(1, accounts[2], {
20         from: accounts[2]
21       }),
22       "Unauthorized"
23     );
24   });
25
26   it("Account0 should be able to propose Account1 as authority", async
         () => {
27     const result = await authorityManagementInstance.propose(1,
           accounts[1], {
28       from: accounts[0]
29     });
30     truffleAssert.eventEmitted(result, "ProposalEvent");
31   });
32
33   ...
34 }
```

Listing 6.4: A test for the authority manager contract

These tests illustrate some useful concepts. We start the batch of tests by deploying the **AuthorityManager** contract to the blockchain. The first test shows how we can query the state of the contract. Such a call does not generate a transaction, so we do not need to set a sending account, as this process is free in terms of Ether. We can also test if a transaction is invalid, and thus reverts, as shown in the second test. Finally, we can perform a function call and check if the block with the transaction contains a transaction receipt with a given event.

Deployment and export The final primary feature of Truffle is scripted deployments. These scripts allow us to define how to deploy a contract. During the deployment procedure, Truffle will deploy the contract to the blockchain through a contract creation transaction, and will optionally run any setup and cleanup procedures in our script. This functionality is especially useful when we have inter-contract dependencies, as shown in the measure contract (Listing 6.2) where we depend on the Treatment contract. The following deployment script shows how we deploy the **Measure** contract:

```
1 var Treatment = artifacts.require("./Treatment.sol");
2 var Measure = artifacts.require("./Measure.sol");
3 var fs = require("fs");
4
5 module.exports = async (deployer, network, accounts) => {
```

```
6
7   //Get the instance representing the deployed treatment contract
8   const treatmentInstance = await Treatment.deployed();
9
10  // Pass the address of the treatment contract when constructing the
        measure contract
11  await deployer.deploy(Measure, Treatment.address, {
12    from: accounts[0],
13  });
14
15  // Register the measure contract in the treatment contract for future
        mutual authentication.
16  await treatmentInstance.registerMeasureContract(Measure.address, {
17    from: accounts[0],
18  });
19
20  // Store the address of the measure contract in a file to be exported
21  var previousAddresses = JSON.parse(fs.readFileSync("shared/addresses.
        json"));
22  var json = JSON.stringify(
23    {
24      ...previousAddresses,
25      measure: Measure.address,
26    }, null, 2
27  );
28  fs.writeFileSync("shared/addresses.json", json);
29 };
```

Listing 6.5: The truffle deplotment script for the Measure contract

In this case, we use the deployment script not only to deploy the measure contract but also to link the measure and treatment contracts together. We also see how we can store the address of the deployed contract in a file to be exported.

6.3 Back-End Server

The back-end of the implemented application is responsible for simulating most workflows in the VerifyMed architecture. In the context of the off-chain part presented in Section 5.2.2, the back-end is a monolithic application with internal services representing the Treatment Provider, License Provider Client, License Issuer Client, and Authority Client. For supporting these services, the application includes functionality for simple key management, interaction with the Ethereum blockchain, RESTful data access, and a WebSocket-based publish/subscribe Application Programming Interface (API) for reacting to events on the blockchain.

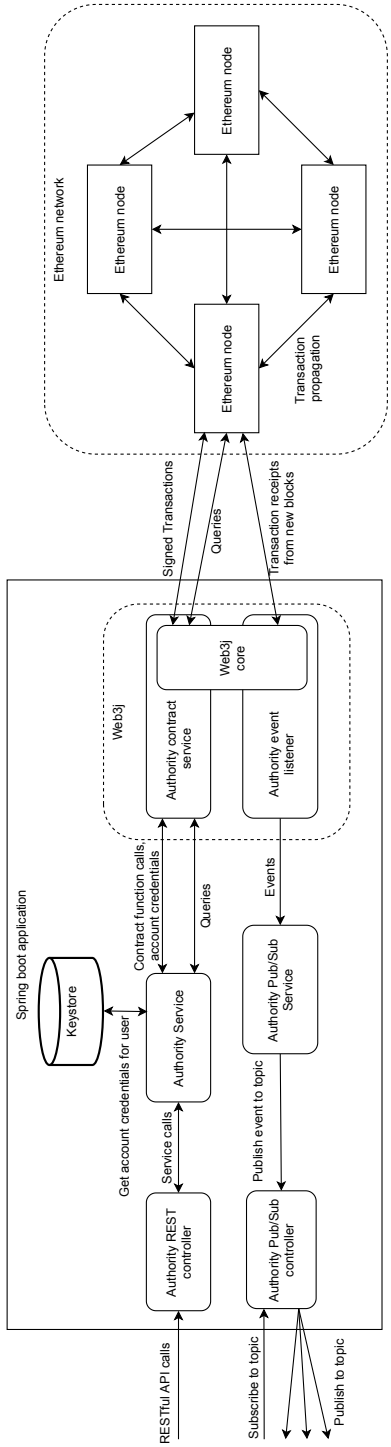


Figure 6.2: An overview of the back-end internals

Figure 6.2 shows the overall structure of the back-end application. Our example shows the Authority part of the application's internal structure. While not shown here, the application uses a similar structure for the remaining processes and components. The remainder of this section will describe the used technology and its use for interaction with the other services. Due to the maturity and broad usage of Java-based web-applications, we will only briefly describe this part. Our interface towards smart contracts will, on the other hand, be described in extensive detail.

Technology

To implement the application, we use the following core technologies:

- **Java**[5], the programming language used for implementing the system;
- **Spring boot**[6] a framework for creating RESTful web services in Java;
- **Web3j**[7] a library for interacting with the Ethereum blockchain.

Controllers

The controller part of the implementation is responsible for the management of the different endpoints for the RESTful web service part of the application, and the pub/sub-topics to which the clients can subscribe. In practice, this layer serves as a bridge between API logic used to interface with the application and the business logic from the model, which is placed in the service layer.

Service Layer

The services in our implementation instance the business logic, as previously presented in the VerifyMed architecture of Section 5.2. Examples of such business logic include:

- Translating API calls to smart contract function calls.
- Querying for data stored within smart contracts.
- Parsing events stored in transaction receipts on the blockchain and propagating these to users.
- Storing data about transactions internally in a database
- Performing other procedures such as patient registration and authentication.

[5]https://www.oracle.com/java/, version *11*
[6]https://spring.io/projects/spring-boot, version *2.2.4*
[7]https://github.com/web3j/web3j, version *4.5.5*

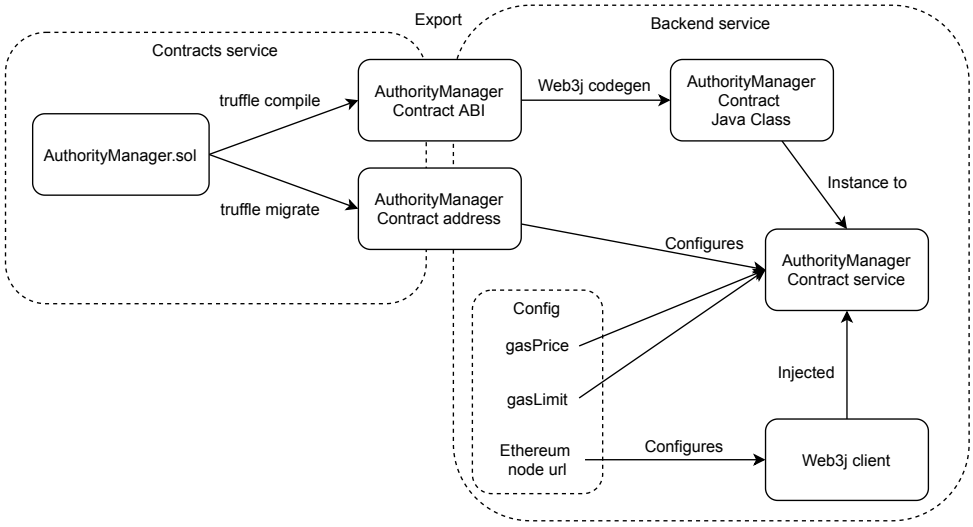


Figure 6.3: Visualization of the process for generating Java wrapper objects for contracts

Smart Contract Interface Layer

To interact with smart contracts, we use the Web3j library. This library contains the most needed functionality for interacting with the Ethereum blockchain and smart contracts deployed on it. The core functionality of the Web3j library is an API for creating and submitting transactions. This API will submit the transactions to the Ethereum node, and get the corresponding transaction receipt once available. The library allows us to submit transactions in a synchronous manner where a transaction is submitted, and a response with the receipt is returned.

In addition to the core functionality for submitting transactions, the Web3j library also includes code generation features for creating Java wrapper objects for interacting with smart contracts. The code generation functionality allows the user to interact with the smart contracts on the blockchain via standard Java objects. These Java objects are generated by using the ABI produced by the contracts service, along with required configuration parameters such as the *gasPrice*, *gasLimit*, and *contract address*. The overall process for generating these Java objects is shown in Figure 6.3.

Once we have instanced such a Java object, we have a reusable and straightforward service that we can use for interaction with the smart contract. To show this, we present a Java snippet using the **propose** function of the AuthorityManager Contract, as shown in Listing 6.3. This snippet creates a Java wrapper object loaded with the

private key of the authority sending the proposal:

```
1  @Service
2  public class AuthorityService {
3
4    @Autowired
5    private CAuthorityManagerFactory cAuthorityManagerFactory; // A
         factory preconfigured with gasPrice, gasLimit, the Web3j client
         and contract address.
6
7    ...
8
9    public void proposeAuthority(final Proposal proposal,
10                                final String privateKey) {
11     try {
12       cAuthorityManagerFactory
13         .fromPrivateKey(privateKey) // The private key of the proposer
14         .propose(proposal.proposalType(), proposal.subject())
15         .send(); // Send the transaction to the blockchain and wait
               for a receipt
16     } catch (final Exception e) {
17       // Triggered if the transaction is either reverted or rejected
             by the node due to insufficient funds in the sending account
18       throw new TransactionFailedException(e.getMessage());
19     }
20   }
21 }
```

Listing 6.6: A Java snippet submitting a transaction to the AuthorityManager Contract

To make this pattern manageable, we use the dependency injection features of the spring framework. This can be observed in line 4 of listing 6.6 This allows us to create a global instance of the contract wrapper once, and then inject it into all the services which require an interface to the contract.

6.4 Web Application

To enable easy testing of the architecture, we developed a web application for using the functionality exposed by the back-end. The web application allows us to simulate different workflows for the overall VerifyMed architecture. This process allows us to capture metrics such as cost and find flaws in the architecture of the system. In addition to exposing functionality from the back-end, the web application also serves as the patient and healthcare worker clients, as presented in the VerifyMed architecture. To handle different roles, we include some basic key management

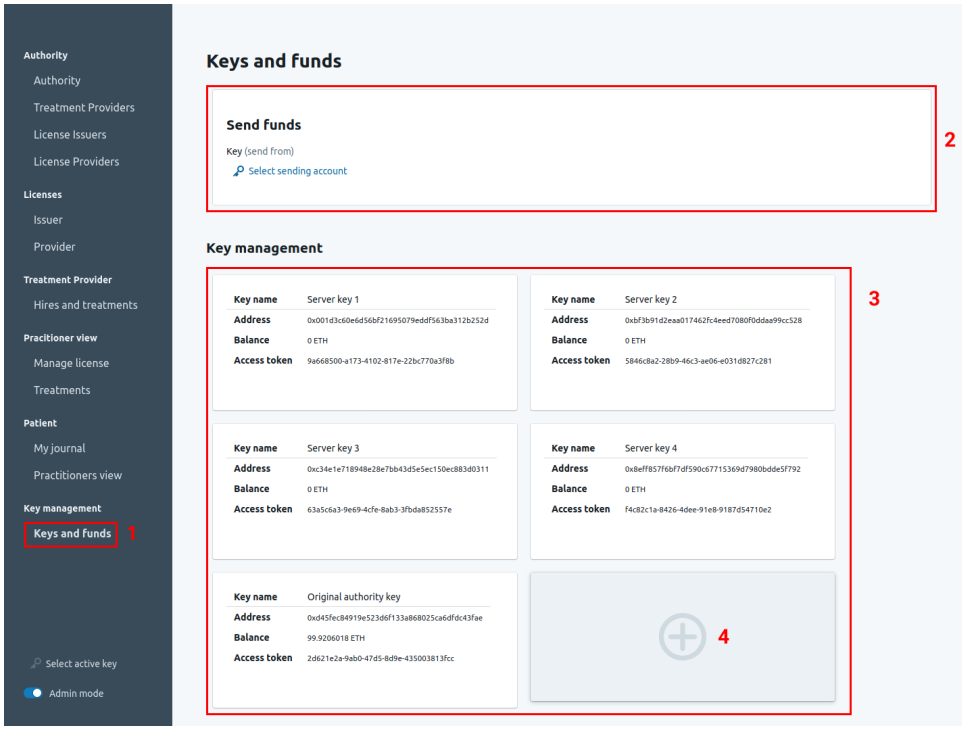


Figure 6.4: Overview of the key management panel in the web application

functionality for generating and storing credentials in the browser. We refer the reader to Appendix A for a complete guide for the whole web application.

We implement the web-application as a client-side rendered application by using the ReactJs framework[8], along with BlueprintJs[9] as a component library, and Undraw[10] for animations. Additionally, we use the Web3js library[11], the JavaScript counterpart to the Web3j library, as presented in the back-end section. This library contains cryptographic libraries for cryptographic hashing, key generation, and digital signatures by using the standards in Ethereum.

An overview of the key management panel is shown in Figure 6.4. The sidebar contains a navigation section where the user can visit pages intended for different stakeholders. Clicking area 1 routes the user to the key management panel. The user

[8]https://reactjs.org/, version *16.12.0*
[9]https://blueprintjs.com/, version *3.24.0*
[10]https://undraw.co
[11]https://github.com/ethereum/web3.js/, version *1.2.6*

can access additional panels in the same manner. Actions within these panels are related to one of the keys within the key management panel. The users can select which key to use with the menu at the bottom of the sidebar. An overview of all keys is shown in area 3, and additional keys can be generated by clicking the panel in area 4. Ether can be sent between accounts by using area 2.

Chapter 7
Test Results

We use the proof-of-concept application presented in Chapter 6 to perform a series of tests for measuring the performance and usability of the VerifyMed architecture presented in Chapter 5. To test the correctness of smart contracts, we applied unit and integration testing. System testing was performed to evaluate the architecture in the context of the requirements defined in Chapter 4. Cost metrics were collected by performing simulated workflows in the proof-of-concept application. These costs were also used the find the theoretical maximum throughput of the architecture.

7.1 Unit and Integration Testing

We tested the behaviors of the smart contracts deployed on the Ethereum blockchain through unit and integration tests. Examples of behaviors include proper enforcement of access control schemes, accurate storage of data, ability to serve the relevant data, and correct events emitted. Unit tests were used to test if the standalone contract performed as expected, while integration tests check if the contract performs as expected within the consortium of the other contracts.

A total of 109 unit and integration tests were written for the smart contracts. These tests were run through the use of the Truffle CLI, and tested against the local Ethereum node created by Ganache. The Ganache instance was configured to instantly produce a new block once a transaction was received. Figure 7.1 shows a subset of the output from the testing procedure with Truffle, and Figure 7.2 shows some of the metadata generated by Ganache for the submitted transactions during this procedure.

7.2 Requirements Validation

We used the implemented proof-of-concept to validate that we meet the functional requirements, as presented in Section 4.3.1. We validate this by using all functionality

```
Contract: Treatment
  ✓ Should be impossible to register Account8 as measureContract for Account1
  ✓ Should be impossible to reregister measureContract for Account0
  ✓ Account5 should be a trusted license
  ✓ Account2 should be a trusted treatment provider
  ✓ Account2 (TreatmentProvider) should be able to create a treatment (42ms)
  ✓ Account5 (LicenseHolder) should be able to approve the first treatment (39ms)
  ✓ Account2 (TreatmentProvider) should be able to create another treatment (45ms)
  ✓ Account5 (LicenseHolder) should be able to approve the second treatment (58ms)
  ✓ The number of treatments registered to account5 (LicenseHolder) should be 2.
  ✓ It should be possible to fetch the URL and datahash of the first treatment though only knowing the licenseAddress
  ✓ It should be possible to fetch the URL and datahash of the second treatment though only knowing the licenseAddress
  ✓ Should be impossible to spend the first treatment (Account6) for Account9 (None)
  ✓ Treatment1 (Account6) should not be spent.
  ✓ It should be possible to get all treatment addresses
  ✓ It should be possible to get all treatment data

Contract: Measure
  ✓ Treatment1 (Account7) should not be spent.
  ✓ Should be possible to create new measure for Account7 (44ms)
  ✓ Treatment1 (Account7) should be spent.
  ✓ Should not be possible to create new measure for Account7
  ✓ Should not be possible to create new measure for Account9 (40ms)
  ✓ Should be possible to get info for measure


109 passing (5s)
```

Figure 7.1: Sample of outputs from running unit and integration tests against smart contracts

```
eth_sendTransaction

  Transaction: 0xfe2ec328d0e57f51facafbbb0e53baf902c89506402398ca9fd058e4e1a89814
  Gas usage: 143741
  Block Number: 29
  Block Time: Tue May 26 2020 09:30:30 GMT+0200 (Central European Summer Time)

eth_getTransactionReceipt
eth_blockNumber
eth_getBlockByNumber
eth_call
eth_blockNumber
eth_getBlockByNumber
eth_sendTransaction

  Transaction: 0x3b3b3d7541f708b4847bc9ea7622cf4319c6e4ac232e3b05d6a17ea918a985f9
  Gas usage: 27022
  Block Number: 30
  Block Time: Tue May 26 2020 09:30:30 GMT+0200 (Central European Summer Time)
  Runtime Error: revert
  Revert reason: Treatment is allready evaluated
```

Figure 7.2: Sample of transactions execution logs from Ganache. These outputs were generated by running unit and integration tests against smart contracts

Figure 7.3: Details for the authority, treatments and evaluations related to a healthcare worker

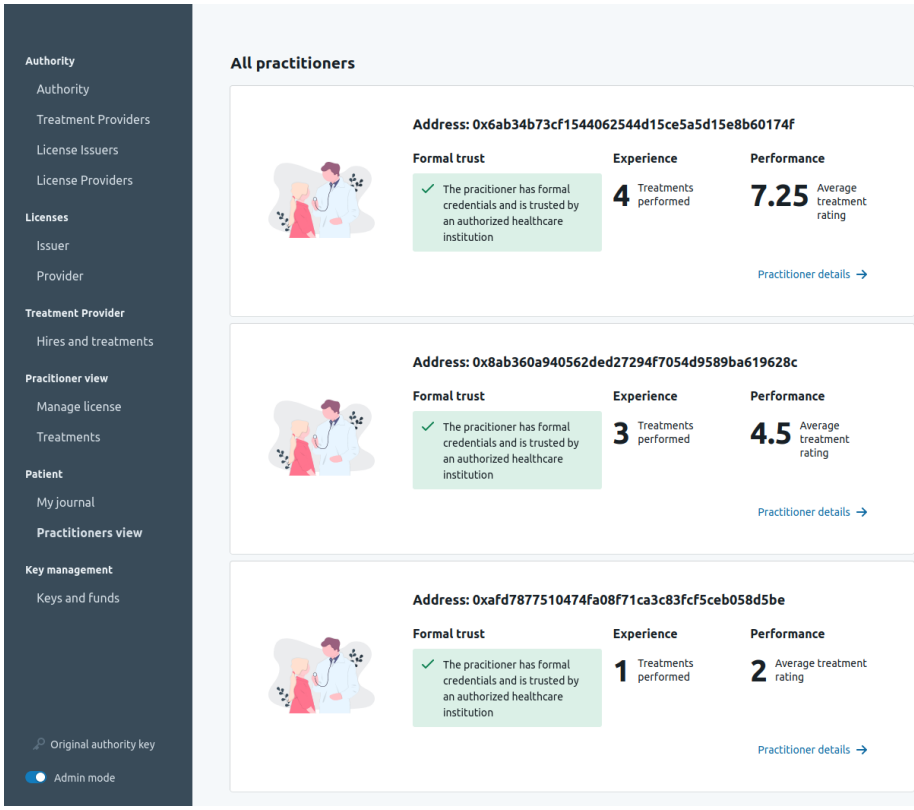


Figure 7.4: An overview of the healthcare workers together with their authority, experience and competence

in the application to generate evidence of authority, evidence of experience, and evidence of competence. These evidences should then be publicly accessible. We refer the reader to the complete runbook used for the tests in Appendix C. This runbook is summarized with the following overall actions:

1. We create accounts representing a set of governance stakeholders, a set of healthcare workers, and a set of patients.

2. We set up trust relationships between the governance entities, and from the governance entities to the healthcare workers.

3. We issue a set of treatments from the healthcare workers to the patients, approve them with the patient account, and finally approve them with the healthcare worker account.

Procedure	Gas cost
Proposing the addition of a new authority, voting on the said proposal, and enacting it. Assumes seven authorities to exist before the new authority is added.	464368
Submitting a treatment and getting it approved by a healthcare worker.	302839
Submitting an evaluation from a patient	143669

Table 7.1: Gas costs for a set of procedures for different stakeholders

By completing the runbook, we are left with an internal contract state on the blockchain, which is publicly available and accessible by anyone. This information was used to construct the overview panel of healthcare workers, as shown in Figure 7.4. Given the address of a healthcare worker, the patient can use this panel to gain information about them. This information includes their formal authority to practice, the number of treatments performed, and the average experience measure from these treatments. Further details about authority, treatments, and evaluations are also available, as shown in Figure 7.3. Overall, this shows that the system fulfills the functional requirements.

7.3 Cost of Usage

By using the web application, we can capture the gas prices for different procedures within the platform. Gas prices serve as a metric for measuring the operational cost of using the blockchain platform. Gas prices can be translated to cost in Ether, which in turn hold monetary value. We measure the gas prices by evaluating the output from our Ethereum node as we submit transactions to it. An example of such an output was previously shown in Figure 7.2. We generate these transactions by first following the runbook in Appendix C, followed by additional tests for edge cases and unpermitted actions.

Table 7.2 shows the gas costs for different transaction types within our platform. These individual transactions can be composed together into procedures. Examples of different procedures and their total gas cost is found in Table 7.1. Procedures with higher complexity will execute more instructions on the EVM, thus yielding higher costs. Storing additional state within the contracts is especially expensive.

To quantify these gas costs into real-world monetary costs, we simulate the system's use over a four-year duration. The exchange rate between gas and Ether is given though the *gasPrice* field within individual transactions. As no fixed exchange rate

between gas and Ether exists, we have to simulate this over a fixed period. This is done by using historical values for the *gasPrice* in transactions on the Ethereum platform. If we were to use the VerifyMed platform in a production use-case, we could expect to pay similar prices for gas as the average within historic blocks.

The simulated price of usage for some workflows is shown in Figure 7.3. These graphs were generated by combining gas costs for our procedures with two public datasets from EtherScan [48]. The first dataset [49] contains the average *gasPrice* used for transactions within blocks on the Ethereum blockchain, giving us a price in *wei* per gas. The second dataset [50] shows the average daily Ethereum price in USD, from which we can deduce USD per *wei*.

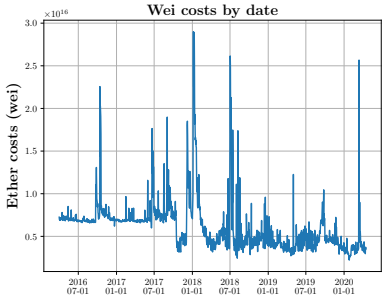
7.4 Throughput

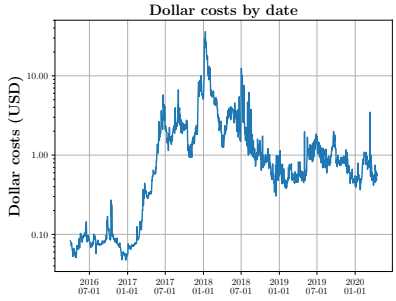
The off-chain part of the VerifyMed architecture presented in Chapter 5 is horizontally scalable. This means we can always add additional capacity to the system by registering additional nodes as authorities, treatment providers, license issuers, and license providers. As most interaction between these stakeholders happens via the blockchain, this process is, in general, easy. However, the on-chain part of our application does have a theoretical throughput limitation, as only a finite number of transactions can fit within a block on the Ethereum ledger.

We find the maximum theoretical throughput of our system by calculating the maximum number of treatments that can be processed by the Ethereum blockchain per year. We consider the full lifecycle of a treatment, including publication, approval, and evaluation. This process costs 446.508 gas in total. The current *gasLimit* on the Ethereum blockchain, limiting the max amount of gas to be used in a block, is 9.982.766 [51]. This limit will allow us to fit 22 treatments within a block. Blocks are generated once every 13.3 seconds [52]. By consuming 1% of the Ethereum blockchain's capacity, we can process 520.000 treatments per year. In contrast, the number of worldwide surgeries was estimated to be between 266,2 and 359,5 million in 2012 [53]. Therefore, we have a maximum theoretical throughput where we can process between 0.14% and 0.19% of worldwide surgeries.

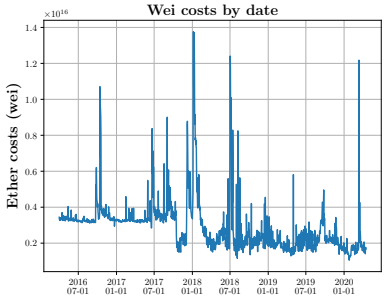
Gas cost	Stakeholder	Transaction type
21000	All	Sending Ether
179013	Authority	Propose to add another address as a authority
149040	Authority	Propose to remove an authority
73686	Authority	Vote on a proposal to add or remove a authority
64297	Authority	Enact a proposal to add or remove a authority
28045	Authority	Trying to enact a proposal without a majority vote in place
45332	Authority	Enact proposal to remove an authority
93707	Authority	Add trust in a registered treatment provider
22909	Authority	Remove trust in a registered treatment provider
48863	Authority	Add trust in a registered license issuer
18829	Authority	Remove trust in a registered license issuer
48906	Authority	Add trust in a registered license provider
15965	Authority	Remove trust in a registered license provider
85959	Treatment Provider	Register address as a treatment provider
200118	Treatment Provider	Create a new treatment
71059	License Issuer	Register as license issuer
88538	License Issuer	Issue a new license to address
23040	License Issuer	Approve movement of license to a new license issuer
86036	License Provider	Register as license provider
38019	License Provider	Approve movement of license to a new license provider
46059	License holder	Propose movement of license to a new license provider
46092	License holder	Propose license provider movement
102721	License holder	Approve published treatment for a given patient
143669	Patient	Submitting an evaluation

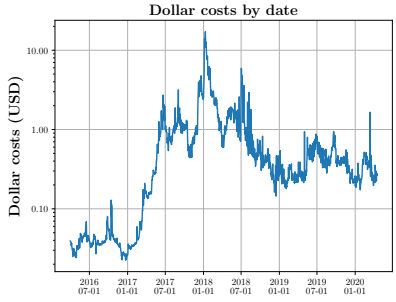
Table 7.2: Gas costs for calls to the the implemented smart contracts



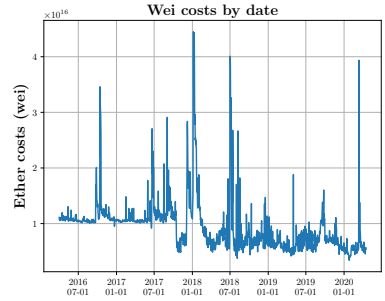


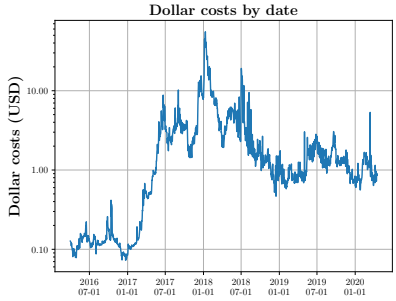
(a) Cost in wei and USD for creating a treatment and getting it approved





(b) Cost in wei and USD for evaluating a treatment





(c) Cost in wei and USD for the process of adding a new authority in a setting with seven existing authorities

Figure 7.5: Simulated wei and USD prices for different procedures over a timespan of four years

Chapter 8
Evaluation and Discussion

Our results show that our proof-of-concept implementation can be used to verify a healthcare worker's authority, experience, and competence. The verifier does not have to place any trust in the healthcare worker themselves. This process can be performed by anyone with access to the Ethereum blockchain network, making the evaluation process fully transparent.

This chapter evaluates and discusses the VerifyMed platform. Our results show that operational costs associated with the platform are moderate. However, the throughput does not allow for a truly global scale to be reached. Therefore, we discuss critical architectural choices, including the choices for data storage. We also discuss how the application can be improved to lower operational costs and increase throughput.

8.1 Ability to Provide Trust in Healthcare Workers

We have demonstrated that the VerifyMed platform can provide a patient with evidence for authority, experience, and competence. This evidence is rooted in a model of trust between governance entities, as presented in Chapter 5. These trust relationships are captured on the blockchain, allowing the patient to use evidence without any established trust-relationships off-chain. However, a patient can only trust this evidence if they inherently trust these governance entities.

Our model for trust is justified in the real-world governance of healthcare. As an environment with heavy regulatory oversight, capturing preexisting governance relationships on a public blockchain serves as a natural first step for providing trust in the virtualized setting. Furthermore, we strengthen our model by adding revocation abilities, where the trust of a governance entity can be revoked if it acts in bad faith. The result is a trust model that is justified in the inherit trust relationship between patients and the currently established healthcare system.

We note that our model for trust is extensible. A patient may trace all trust relationships from any evidence back to a top-level authority. The patient stands free to blindly trust the blockchain or use a third-party service to verify each of the upstream governance entities independently.

We finally state that our platform's main intent is to govern and publish evidence for authority, experience, and competence on a *public* blockchain. Once published, third-party applications stand free to use this information as they see fit. By publishing the ABI and contract addresses, third-party developers may construct contract APIs via the Web3js or Web3j libraries. Thus, using the on-chain part is simple for third-party developers. This enables usage within any healthcare platform wishing to provide secure verification of their healthcare workers to patients.

8.2 Social Impact

As virtualized healthcare platforms become more relevant for each passing day, we believe that blockchain-based solutions such as the VerifyMed platform will have a large social impact. The platform contributes to the security of patients and their trust in their healthcare system. We achieve this social impact by empowering the patient to become well-informed about the healthcare worker, thus decreasing the risk of malpractice and enabling the patient to gain an expectation for the quality of care.

We also emphasize that the information published on the blockchain has use-cases outside of the scope of the relationship between patients and healthcare workers. Other identified use-cases include:

1. The data may give external regulators an impression of the quality of care provided by a healthcare institution. The available metrics for experience and competence may be used for audits or to justify a healthcare institution's further inspection.

2. If deployed in an international setting, the platform may be used to evaluate healthcare systems on a broad scale. The platform allows for the comparative evaluation of national healthcare systems in real-time.

3. Information about the healthcare worker can be used within mobility processes for healthcare workers. Background checks can be simplified as the healthcare worker can refer to concrete evidence for their experience and competence. This offers a compelling alternative to current processes, which is often based on extensive manual background checks based on credentials received from the healthcare worker themselves.

4. The ability of healthcare workers to show experience and competence can enable new services to be created. Examples include premium second-opinion services where patients can indeed verify that the healthcare worker is very experienced.

Overall, we believe that introducing our model for improved trust in healthcare workers will be a crucial contribution to virtualized healthcare services. Platforms based on blockchain will enable better patient care, better patient outcomes, and a healthcare system that can scale to future needs.

8.3 Using a public blockchain

We use the public Ethereum blockchain for hosting our smart contracts. This choice is incorporated in our architecture, as we have to take the public nature of the blockchain into account. Using a public blockchain requires us to limit the published data to protect patient privacy, and we must implement access control schemes within our smart contracts. Additionally, we must incorporate a mechanism to transfer Ether between accounts, allowing them to submit transactions.

8.3.1 The Advantages of a Public Blockchain

Using a public blockchain gives us some key advantages. The most noteworthy of these is transparency. Once the state in a contract is updated, anyone with access to an Ethereum node can access it. This can allow patients and third-party services to interact with our smart contracts, without any further interaction from the off-chain part of our application.

The public nature can increase the adoption of the platform from healthcare institutions. No inter-organizational agreements or procedures are required to access data on the blockchain. Any healthcare service or institution can immediately gain access to data. Furthermore, these organizations can implement separate systems for publishing data to the blockchain, as long as their addresses are trusted in some manner. Organizations can publish and get data with a standard format via the blockchain. This increases the data sharing capabilities between these organizations.

The final argument for using a public blockchain is the possibility for interaction with the underlying cryptocurrency on the blockchain. This allows us to build incentive models on top of the existing smart contract, pushing the participants into correct behaviour. This allows us to solve problems which are tightly related to our initial problem statement of trust in healthcare.

One concise problem which we can solve with incentive models is evaluation response rates. We have assumed that patients are willing to submit experiences related to their treatments. However, many may choose not to submit a digital PREMs [54]. By extending our smart contracts to send cryptocurrency to the evaluating patient, we can incentivize this behavior, and thus increase response rates.

8.3.2 The Disadvantages of a Public Blockchain

One of the prices paid for using a public blockchain is monetary. Thus, the advantage gained in terms of features and properties must yield a return of greater value than these costs. Our results show that the operational costs associated with our platform are moderate. Recent months show prices of 1 USD for submitting a treatment and getting it approved. Evaluations costs around 50 cents. However, when simulating prices over a timespan of four years, large fluctuations were observed. This may lead to unpredictable operational costs over time, generating unnecessary financial risk for the healthcare industry.

The main scalability problem with our platform originates from the throughput. The Ethereum blockchain offers low throughput compared to the pace of the healthcare system. Additionally, throughput in terms of transactions per second decreases as transaction size increases. We also emphasize that our maximum throughput is based on an assumption of using 1% of the transaction capacity on the Ethereum blockchain, which can be orders of magnitude above feasible usage. Additionally, costs will scale linearly with the usage, yielding low returns from the economics of scale. On the contrary, large scale usage can trigger an increase in the price of Ether, further increasing operational costs.

Using a public blockchain does also give us some disadvantages in terms of architectural impact. As costs scale linearly with the amount of data within a transaction, we must limit the amount of data stored on the blockchain. This results in mostly hashes and small summaries being stored on the blockchain. Adding additional data will increase costs accordingly.

The immutable nature of public blockchains results in an inefficient storage architecture over time. As time progresses, data stored on the blockchain lose importance. If, for example, a healthcare worker passes away, data related to them will still remain on the blockchain indefinitely. In contrast to financial use-cases where a complete history for some cryptocurrency is required, this is not always the case within the healthcare industry. We are thus paying an unnecessary premium for a service that is not required long-term.

Finally, we must state that our platform is governed by a set of authorities, license issuers, license providers, and treatment providers. This allows us to publish evidence

for trust rooted in real-world trust relationships on the blockchain. This model contrasts with the fully trustless principles which usually are applied within the domain of public blockchains. We argue that the complexity of the healthcare system cannot be captured in a fully trust-less model. Nevertheless, this conflict introduces additional cost and complexity.

8.4 The Case for Private Blockchains

Using a public blockchain results in a cost premium, low possible throughput, and undesired architectural impact. However, we still believe that blockchain is a crucial technology to support the inter-organizational data sharing required for providing trust in virtualized healthcare. An alternative approach for creating such a solution is through private blockchain platforms.

Private blockchains allow a consortium of organizations to share data by taking part in a blockchain network that is shared between them. In the context of VerifyMed, we can envision a shared private blockchain maintained by our governance entities. We can extend our model for trust in governance entities to a permission model for the blockchain, where trusted governance entities can take part in the private blockchain. The access control schemes in our smart contracts can be translated to permissions in a private blockchain, allowing patients to gain permission to submit evaluations to it.

The biggest advantages of private blockchains are reduced cost and increased throughput. For common platforms such as Hyperledger Fabric [55] or Corda [56], the cost is linear with the operational cost of the underlying infrastructure. This offers a compelling value proposition compared to costs linear to the number of transactions. This transition offers dramatically reduced costs, along with lower variance over time, making the operational costs predictable. Additionally, throughput can be improved by using consensus algorithms with different trust assumptions than public blockchains.

8.5 Limitations of VerifyMed

We have validated that the VerifyMed platform fulfills our requirements through unit, integration, and system testing. The platform is capable of providing trust in healthcare workers in a transparent manner. Nevertheless, some key disadvantages can be identified. These were identified both during application modeling, implementation, and system testing.

8.5.1 Authentication of Patients

We have presented quality attributes stating that treatments and evaluations should be rooted in real-world interactions. This implies that *fraudulent* healthcare workers and patients should be impossible to create. Our architecture incorporates a precise model for preventing this situation on the healthcare worker side through on-chain access control schemes. However, we cannot deploy equivalent mechanisms for patients due to privacy concerns. We have therefore assumed that treatment providers are able to authenticate patients through some identity provider, and have defined the details of this procedure as out-of-scope. The resulting scheme trusts treatment providers with this task, which may be naive.

To give stronger guarantees for the prevention of fraudulent patients, the system should be extended with stronger cryptographic mechanisms to ensure that treatments and evaluations indeed originate from real-world patients. One possibility is applying zero-knowledge proofs for publicly showing that a treatment and evaluation originates from a patient while keeping the specific patient secret.

Another alternative for improving patient privacy by using schemes such as blind signatures, ring-signatures [57], and stealth addresses [58] to protect the sending address, and thus yield patient privacy. Such schemes can either be implemented within smart contracts or be delivered as a service from the underlying blockchain platform.

8.5.2 Key Management

The VerifyMed architecture has a heavy reliance on keys that are controlled by different stakeholders. These stakeholders can loose or leak their keys. In these cases, we rely on the governance entities to revoke their trust in these entities. This process becomes harder as the number of participants in the system grows. Further key management mechanisms such as key rotation and revocation should, therefore, be implemented.

8.5.3 Cost as an Architectural Limitation

In addition to the monetary cost related to the cost of transactions, some architectural limitations were identified. The most prominent of these is the extensive setup phase. Users of the platform cannot participate before their accounts have Ether associated with their account. This gives a considerable reliance on the Ether cryptocurrency from the healthcare sector. Thus, all stakeholders in the system must ensure that they have an Ethereum account with Ether, to be able to interact with our platform. Requiring ownership of such accounts is a burden and should not be a requirement for receiving healthcare. Additionally, adding Ether to an account creates an immutable

link between the sender and receiver. This limits how we can use concepts such as one-time addresses, limiting our possibilities for providing privacy.

8.5.4 Narrowly Scoped Security Model

Our requirements capture two main adversaries: fraudulent patients and fraudulent healthcare workers. The treatment provider is expected to authenticate patients to protect against fraudulent patients. During the patient and healthcare worker interaction, we assume a security model where the treatment provider cannot be adversarial. This model is justified with the possibility for authorities to revoke their trust in the treatment provider. However, this model cannot reasonably ensure that no trusted adversarial treatment providers do not exist at any moment in time.

8.5.5 Large Governance Complexity

Our model for creating a trusted environment on the blockchain tries to model and capture real-world trust relationships. This results in large smart contracts of high complexity, leaving room for errors. As Solidity is a limited language with many common security pitfalls [59], the risk of security flaws increases with complexity. As the blockchain is immutable, contracts cannot be patched once deployed. This setting results in a preference for simpler models, with less complexity on-chain. Therefore, an alternative approach for governance dictated by off-chain certificate authorities or similar should be considered.

8.6 Lessons Learned from VerifyMed

Creating a complex distributed application on the Ethereum blockchain is easier than ever before. The developer tooling allows anyone to develop, test, and deploy smart contracts easily. These smart contracts can be interacted with through multiple open-source tools such as Web3j and Web3js.

However, public blockchains are not necessarily the right choice for all blockchain-based applications. We have described how our platform delivers social impact via an open, fully distributed, and transparent platform. However, this comes at a considerable price. Using the system will couple the healthcare system with the Ethereum blockchain. This will require the usage of the Ether cryptocurrency, and smart contracts deployed on the blockchain, resulting in moderate costs and low scalability. Once in usage, moving on to a new platform will be hard.

Proofs of concepts can, in nature, reveal multiple types of findings. They can show that a concept works perfectly and is ready for real-world use-cases. They can also reveal the critical research areas which must be addressed before a solution is suited for a real-world use-case. We do see a future in the VerifyMed platform, but extensive

testing into our assumptions must be performed. Private blockchains should be tested to address our cost and scalability issues. Finally, our model for trust should be extended to cover the full extent of the healthcare industry.

8.7 Future Work

Systematic Testing of Gained Trust

Our presented application allows patients to gain evidence for the authority, experience, and competence of a healthcare worker. We have assumed that this information can either increase or decrease the patients' trust in the healthcare worker. While a rational assumption in nature, it should nevertheless be tested. Therefore, we propose to perform further research within the patient and healthcare worker interaction to see if such information indeed increases trust in the healthcare worker.

Scoped Research into Trust Models within the Healthcare Industry

Our research has only scratched the surface of the complexity within the healthcare industry. Areas such as pharmaceuticals, insurance, and education are vital players who should be included in a future variant of VerifyMed. Further research into the trust relationships within the healthcare system should, therefore, be performed.

Scaling the Platform with Private Blockchains

The VerifyMed architecture should be implemented on a private blockchain platform. This will allow us to assess if the platform can deliver the same social impact while remaining less transparent to the general public.

Scaling the Platform with Distributed File Systems

An alternative to using a private blockchain is to scale the VerifyMed platform by adding a storage layer between the on-chain and off-chain parts of our applications. Distributed file systems such as Swarm [60] and IPFS [61] allow files to be stored in a public, immutable and transparent manner. At the same time, the integrity can be protected by storing a Merkle root hash on the blockchain.

Chapter 9
Conclusion

In this thesis, we have designed, implemented, and evaluated VerifyMed - A blockchain platform for providing transparent trust in healthcare workers. The platform allows patients to verify the authorization, experience, and competence of a healthcare worker in a transparent, trust-less, and non-repudiable manner. Additionally, the platform may be used by third-party applications, regulators, and auditors for additional social impact. To our knowledge, this is the first blockchain solution addressing this specific problem.

The platform is the product of an extensive software development process. We have performed background research into the healthcare domain and preexisting proposals for using blockchain technology within it. We have identified a need for increased trust in virtualized healthcare, and have defined functional and non-functional requirements for blockchain applications trying to solve this problem. The VerifyMed architecture was created based on this, along with models for creating trust within a virtualized healthcare environment. The architecture was instanced to a proof-of-concept application with intuitive user interfaces. The proof-of-concept was used in extensive simulated testing.

The VerifyMed platform uses a novel model for capturing evidence for trust in a healthcare worker: evidence of authority, evidence of experience, and evidence of competence. This evidence is stored in a public and non-repudiable manner on the Ethereum blockchain. Evidence for authority is generated by a consortium of governance entities from the healthcare domain, who publish their preexisting trust relationships. Evidence of experience is created by creating a history of treatments delivered to patients by healthcare workers. Evidence of competence is created by storing experience measures from these treatments.

Our results show that using our platform has moderate costs associated with it. This cost originates from the gas costs related to publishing transactions to the Ethereum blockchain platform. Structured system testing of our platform shows that the average

cost for publishing a summary of a treatment and getting it approved is around 1 USD, and the cost for evaluating a treatment is around 50 cents. Additionally, extensive cost is associated with governance. As gas prices vary over time, we have simulated historic costs over a time span of 4 years. During this time frame, large fluctuations were observed, yielding high financial risk related to using the platform. Additionally, we have uncovered that the VerifyMed platform cannot scale globally by using the Ethereum blockchain network.

The VerifyMed platform shows how blockchain technology can have a place in the healthcare industry. The platform empowers patients to verify that that healthcare workers indeed hold the required authorizations, experience, and competence. This can enable further advances in virtualized healthcare. Still, further work is required to create a blockchain platform ready for production use-cases.

References

[1] M. Raikwar, D. Gligoroski, and K. Kralevska. Sok of used cryptography in blockchain. *IEEE Access*, 7:148550–148575, 2019.

[2] Gavin Wood. Ethereum yellow paper. *Internet: https://github. com/ethereum/yellowpaper, [version 7e819ec - 2019-10-20]*, 2014.

[3] Christoph Thuemmler and Chunxue Bai. *Health 4.0: Application of Industry 4.0 Design Principles in Future Asthma Management*, pages 23–37. Springer International Publishing, Cham, 2017.

[4] Graham Scambler and Nicky Britten. System, lifeworld and doctor–patient interaction: Issues of trust in a changing world. In *Habermas, critical theory and health*, pages 53–75. Routledge, 2013.

[5] Michael Nofer, Peter Gomber, Oliver Hinz, and Dirk Schiereck. Blockchain. *Business & Information Systems Engineering*, 59(3):183–187, 2017.

[6] Melanie Swan. *Blockchain: Blueprint for a new economy.* " O'Reilly Media, Inc.", 2015.

[7] Alan R Hevner, Salvatore T March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS quarterly*, pages 75–105, 2004.

[8] Klaus Pohl. *Requirements engineering: fundamentals, principles, and techniques.* Springer Publishing Company, Incorporated, 2010.

[9] Jens-Andreas Hanssen Rensaa, Danilo Gligoroski, Katina Kralevska, Anton Hasselgren, and Arild Faxvaag. Verifymed – a blockchain platform for transparent trust in virtualized healthcare: Proof-of-concept, 2020.

[10] Hasselgren, Rensaa, Gligoroski, Kralevska, and Faxvaag. *Blockchain for increased trust in virtual healthcare; Ushered through the VerifyMed platform.* A work in progress paper on blockchain for trust in healthcare, to be submitted to the journal Information Processing & Management, for their special issue Blockchain for Information Systems Management and Security.

[11] World Bank. Age dependency ratio (% of working-age population). *Internet: https://data.worldbank.org/indicator/SP.POP.DPND, Accessed 02.04.2020*, 2020.

[12] World Health Organization et al. Future of digital health systems: report on the who symposium on the future of digital health systems in the european region. *Report on the WHO Symposium on the Future of Digital Health Systems in the European Region*, 2019.

[13] Theresa Weldring and Sheree MS Smith. Article commentary: Patient-reported outcomes (pros) and patient-reported outcome measures (proms). *Health services insights*, 6:HSI–S11093, 2013.

[14] Charlotte Kingsley and Sanjiv Patel. Patient-reported outcome measures and patient-reported experience measures. *Bja Education*, 17(4):137–144, 2017.

[15] Saif Al-Kuwari, James H. Davenport, and Russell J. Bradford. Cryptographic hash functions: Recent design trends and security notions. Cryptology ePrint Archive, Report 2011/565, 2011. https://eprint.iacr.org/2011/565.

[16] Morris J Dworkin. Sha-3 standard: Permutation-based hash and extendable-output functions, 2015.

[17] Guido Bertoni, Joan Daemen, Michaël Peeters, and GV Assche. The keccak reference, version 3.0. *NIST SHA3 Submission Document (January 2011)*, 2011.

[18] Bertoni Guido, Daemen Joan, P Michaël, and VA Gilles. Cryptographic sponge functions, 2011.

[19] Ralph C Merkle. Method of providing digital signatures, January 5 1982. US Patent 4,309,569.

[20] Don Johnson, Alfred Menezes, and Scott Vanstone. The elliptic curve digital signature algorithm (ecdsa). *International journal of information security*, 1(1):36–63, 2001.

[21] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472, 1985.

[22] Neal Koblitz. *A course in number theory and cryptography*, volume 114. Springer Science & Business Media, 1994.

[23] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Annual International Cryptology Conference*, pages 174–187. Springer, 1994.

[24] Ian Miers, Christina Garman, Matthew Green, and Aviel D Rubin. Zerocoin: Anonymous distributed e-cash from bitcoin. In *2013 IEEE Symposium on Security and Privacy*, pages 397–411. IEEE, 2013.

[25] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE, 2014.

[26] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive arguments for a von neumann architecture. *IACR Cryptology ePrint Archive*, 2013:879, 2013.

[27] Xavier Boyen, Christopher Carr, and Thomas Haines. Graphchain: A blockchain-free scalable decentralised ledger. In *Proceedings of the 2nd ACM Workshop on Blockchains, Cryptocurrencies, and Contracts*, BCC '18, page 21–33, New York, NY, USA, 2018. Association for Computing Machinery.

[28] Pierangela Samarati and Sabrina Capitani de Vimercati. Access control: Policies, models, and mechanisms. In *International School on Foundations of Security Analysis and Design*, pages 137–196. Springer, 2000.

[29] Nicolas T. Courtois, Marek Grajek, and Rahul Naik. Optimizing sha256 in bitcoin mining. In Zbigniew Kotulski, Bogdan Księżopolski, and Katarzyna Mazur, editors, *Cryptography and Security Systems*, pages 131–144, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[30] Bloemen R, Logvinov L, and Evans J. Eip-712 ethereum typed structured data hashing and signing. *Online at https://github.com/ethereum/EIPs/blob/master/EIPS/eip-712.md*, 2020.

[31] Chris Dannen. *Introducing Ethereum and Solidity*, volume 1. Springer, 2017.

[32] Arne Andersson and Stefan Nilsson. Efficient implementation of suffix trees. *Software: Practice and Experience*, 25(2):129–141, 1995.

[33] Anton Hasselgren, Katina Kralevska, Danilo Gligoroski, Sindre A. Pedersen, and Arild Faxvaag. Blockchain in healthcare and health sciences—a scoping review. *International Journal of Medical Informatics*, 134:104040, 2020.

[34] Peter B Nichol and Jeff Brandt. Co-creation of trust for healthcare: The cryptocitizen framework for interoperability with blockchain. *Research Proposal. ResearchGate*, 2016.

[35] Eric Funk, Jeff Riddell, Felix Ankel, and Daniel Cabrera. Blockchain technology: A data framework to improve validity, trust, and accountability of information exchange in health professions education. *Academic Medicine*, 93(12):1791–1794, 2018.

[36] Angelo Capossele, Andrea Gaglione, Michele Nati, Mauro Conti, Riccardo Lazzeretti, and Paolo Missier. Leveraging blockchain to enable smart-health applications. In *2018 IEEE 4th International Forum on Research and Technology for Society and Industry (RTSI)*, pages 1–6. IEEE, 2018.

[37] A. Azaria, A. Ekblaw, T. Vieira, and A. Lippman. Medrec: Using blockchain for medical data access and permission management. In *2016 2nd International Conference on Open and Big Data (OBD)*, pages 25–30, Aug 2016.

[38] Gaby G. Dagher, Jordan Mohler, Matea Milojkovic, and Praneeth Babu Marella. Ancile: Privacy-preserving framework for access control and interoperability of electronic health records using blockchain technology. *Sustainable Cities and Society*, 39:283 – 297, 2018.

[39] Philipp Schmidt. Blockcerts—an open infrastructure for academic credentials on the blockchain. *MLLearning (24/10/2016)*, 2016.

[40] Marco Baldi, Franco Chiaraluce, Migelan Kodra, and Luca Spalazzi. Security analysis of a blockchain-based protocol for the certification of academic credentials. *arXiv preprint arXiv:1910.04622*, 2019.

[41] P. Zhang, M. A. Walker, J. White, D. C. Schmidt, and G. Lenz. Metrics for assessing blockchain-based healthcare decentralized apps. In *2017 IEEE 19th International Conference on e-Health Networking, Applications and Services (Healthcom)*, pages 1–4, Oct 2017.

[42] Thomas McGhin, Kim-Kwang Raymond Choo, Charles Zhechao Liu, and Debiao He. Blockchain in healthcare applications: Research challenges and opportunities. *Journal of Network and Computer Applications*, 135:62 – 75, 2019.

[43] Morten Hertzum and Gunnar Ellingsen. The implementation of an electronic health record: Comparing preparations for epic in norway with experiences from the uk and denmark. *International Journal of Medical Informatics*, 129:312 – 317, 2019.

[44] William J Gordon and Christian Catalini. Blockchain technology for healthcare: facilitating the transition to patient-driven interoperability. *Computational and structural biotechnology journal*, 16:224–230, 2018.

[45] Steven D Pearson and Lisa H Raeke. Patients' trust in physicians: many theories, few measures, and little data. *Journal of general internal medicine*, 15(7):509–513, 2000.

[46] T. Chang and D. Svetinovic. Improving bitcoin ownership identification using transaction patterns analysis. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 50(1):9–20, 2020.

[47] R Yavatkar, D Pendarakis, and R Guerin. Rfc 2753–a framework for policy-based admission control, ieft, internet engineering task force std., january 2000.

[48] Etherscan. Etherscan block explorer. *Online at https://etherscan.io/*, 2020.

[49] Etherscan. Etherscan block explorer - ethereum average gas price chart, csv export. *Online at https://etherscan.io/chart/gasprice*, 2020.

[50] Etherscan. Etherscan block explorer - ether daily price (usd) chart, csv export. *Online at https://etherscan.io/chart/etherprice*, 2020.

[51] Etherscan. Etherscan block explorer - ethereum average gas limit chart. *Online at https://etherscan.io/chart/gaslimit*, 2020.

[52] Etherscan. Etherscan block explorer - ethereum average block time chart. *Online at https://etherscan.io/chart/blocktime*, 2020.

[53] Thomas G Weiser, Alex B Haynes, George Molina, Stuart R Lipsitz, Micaela M Esquivel, Tarsicio Uribe-Leitz, Rui Fu, Tej Azad, Tiffany E Chao, William R Berry, et al. Size and distribution of the global volume of surgery in 2012. *Bull World Health Organ*, 94(3):201–209F, 2016.

[54] Nicole JE Horevoorts, Pauline AJ Vissers, Floortje Mols, Melissa SY Thong, and Lonneke V van de Poll-Franse. Response rates for patient-reported outcomes using web-based versus paper questionnaires: Comparison of two invitational methods in older colorectal cancer patients. *J Med Internet Res*, 17(5):e111, May 2015.

[55] The Linux Foundation. Hyperledger fabric. *Online at https://www.hyperledger.org/projects/fabric*, 2020.

[56] Corda. Corda - open-source blockchain platform for business. *Online at https://www.corda.net/*, 2020.

[57] Sherman S. M. Chow, Siu-Ming Yiu, and Lucas C. K. Hui. Efficient identity based ring signature. In John Ioannidis, Angelos Keromytis, and Moti Yung, editors, *Applied Cryptography and Network Security*, pages 499–512, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[58] Bassam El Khoury Seguias. Monero's building blocks part 10 of 10–stealth addresses. 2018.

[59] Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. A survey of attacks on ethereum smart contracts sok. In *Proceedings of the 6th International Conference on Principles of Security and Trust - Volume 10204*, pages 164–186, New York, NY, USA, 2017. Springer-Verlag New York, Inc.

[60] Ethersphere. Swarm storage and communication infrastructure for a sovereign digital society. *Online at https://ethersphere.github.io/swarm-home/*, 2020.

[61] Juan Benet. Ipfs-content addressed, versioned, p2p file system. *arXiv preprint arXiv:1407.3561*, 2014.

Appendix A

Application Guide for Users

This guide shows how to use the implemented UI for our proof-of-concept. We first showcase the standard functionality used across all pages, such as the overall user interface and key management. Afterward, we walk through workflows from the perspective of each of the individual stakeholders. We refer the reader to Appendix B for setting up the application for local testing.

A.1 The User Interface

Figure A.1 shows the overall user interface, which is common for all stakeholder workflows. The left side shows a navigation sidebar, which has the following sections as numbered in the figure:

1. Section for navigating to the panels relevant to the authority stakeholder
2. Section for navigating to the panels relevant to the License Provider and license issuer stakeholder
3. Section for navigating to the panels relevant to the Treatment Provider stakeholder
4. Section for navigating to the panels relevant to the healthcare worker stakeholder
5. Section for navigating to the panels relevant to the patient stakeholder
6. Section for navigating to the panels relevant to key management, relevant to all stakeholders
7. Selection button for selecting the current active Ethereum keypair (ECDSA keypair) to be used for actions in the UI. Relevant for all stakeholders
8. Toggle for *admin mode*. This toggle gives access to an account, which is the first default authority, giving a baseline allowing the user to expand the hierarchy

Figure A.1: An overview of sidebar in the proof-of-concept UI

from there. This account has an initial balance of 100ETH, which can be sent to other accounts, so they are able to create transactions.

A.2 Key Management

As this proof-of-concept application is technical, we expose functionality for managing the ECDSA keys used for accounts and blockchain transactions during the application run-time. Figure A.2 shows the panel intended for this purpose. This panel allows users to create and view key-pairs. These keys are either stored on the server or locally, dependant on the intent. One can also use the panel to send Ether from one account to another.

Key Management Functionality Overview

Figure A.2 shows the key management panel with the following sections:

1. The button to click to access the key management panel

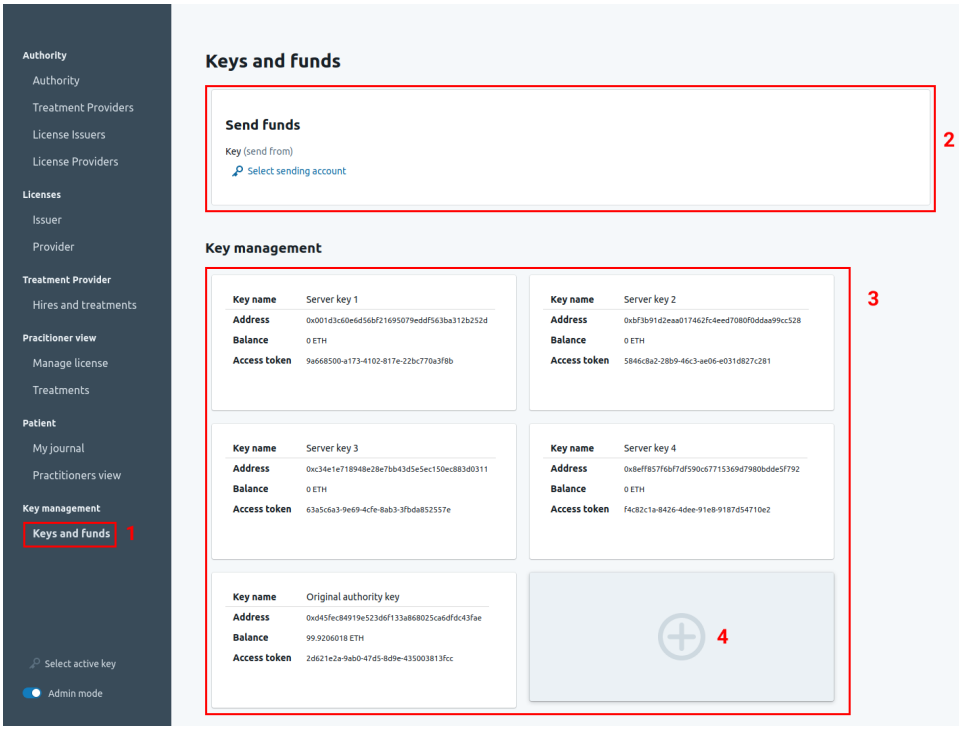


Figure A.2: Overview of the key management panel of the UI

2. The panel to send Ether from one account to another
3. The section to view current keys of all formats. This shows fields such as address and balance. If a local key is shown, the private key will be used. If present, an access token to use the key on the back-end is shown.
4. A card which can be clicked to create new keys with a selection of types.

Sending Ether

Section 2 in Figure A.2 allows all stakeholders to send funds between the keys to which they have access. Figure A.3 shows how the interaction with the panel, where the following steps are taken to send funds:

1. Select the key to send funds from
2. Select the amount of Ether to send

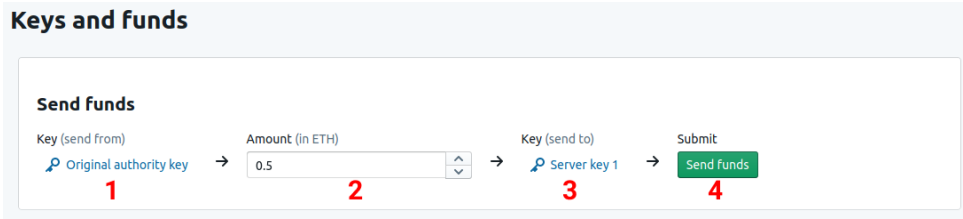


Figure A.3: The send funds panel in use within the UI

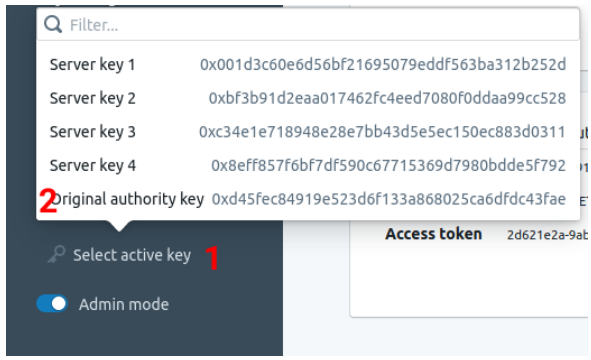


Figure A.4: The process for selecting a key in the UI

3. Select the key to send funds to
4. Click to send the send funds transaction to the blockchain.

This will create a new send transaction on the blockchain platform. Once the transaction is added to the ledger, the user may refresh the page to see the new balances.

Selecting a Key

In any panel, it is possible to select the key which is used. This allows the user to get data which is related to the key, and for transactions to be signed by the selected key. Figure A.4 shows how a user can select among all stored keys.

1. Click the *Select active key* button at the bottom of the side panel.
2. Select a key type from the list which appeared.

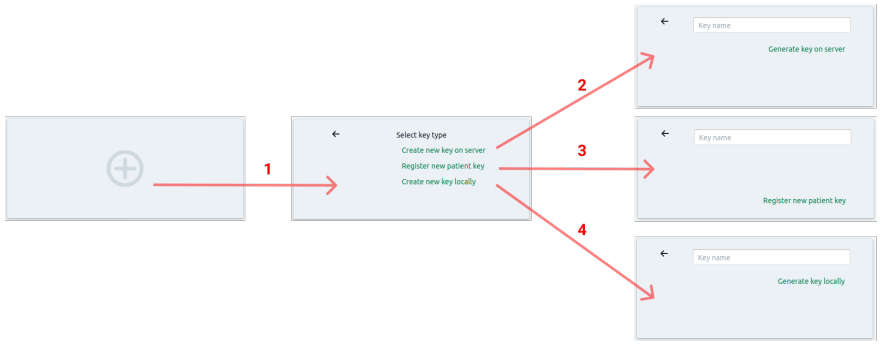


Figure A.5: The process for creating a key in the UI

Create a Key

There are three different keys which can be created in the application:

- **Server side keys** are generated on the back-end. These keys are associated with an access token, which allows the user to use them. These keys are, for simplicity, used in contexts where the key management is of low importance to the quality attributes in the system.
- **Local keys** these keys are locally generated, where both the secret and public key is isolated to the local client.
- **Patient keys** these keys are locally generated, and only stored on the client. However, during creation, the client and server go through a registration procedure where the client proves ownership of the key to the back-end. The client gets an access token, which can be used for access control to prove the patient's identity in some scenarios.

Figure A.5 shows the process for creating a key. This process is initialized by pressing the card shown in section 4 of Figure A.2. The user first selects the type of key they want to generate, then provide a name for the key, and finally click the respective generate button.

A.3 The Authority Stakeholder

The authority is the top level in our trust hierarchy. As an authority, two main workflows are present. The first is participating in the distributed governance protocol. When doing so, authorities may propose to add new authorities or to remove existing ones. All authorities vote on these proposals, and if a proposal gets a majority vote,

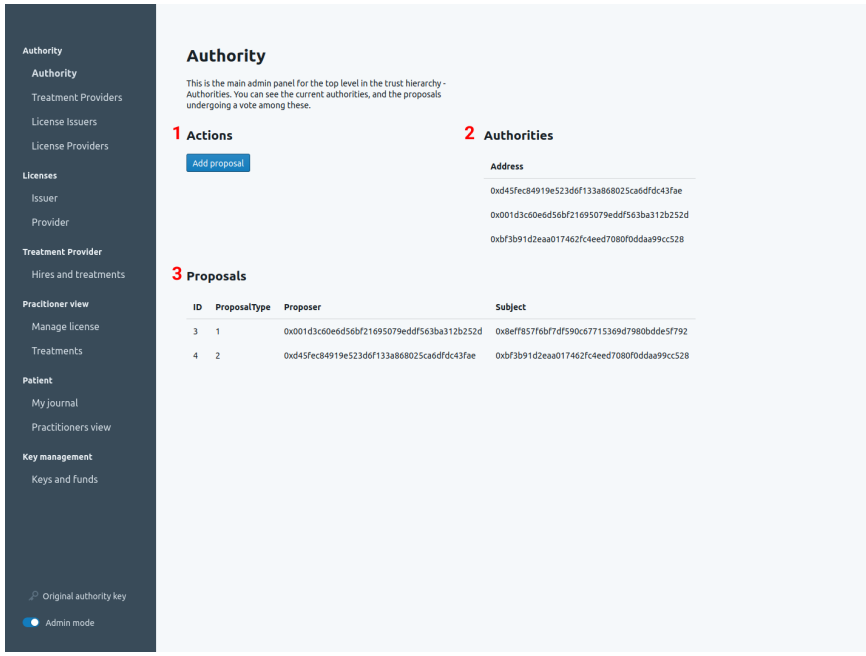


Figure A.6: The main panel for managing authorities in the UI

they can be enacted. The second workflow is adding trust in the stakeholders beneath authorities in the trust hierarchy - treatment providers, license issuers, and license providers.

Managing Authorities

Figure A.6 show the main panel for viewing and managing authorities. All the data in this panel originates directly from the blockchain ledger. The panel includes the following section:

1. A button allowing the authority to add proposals for adding or removing new authorities. Figure A.7 shows the popup which appears once clicked. The user must input the account address of the subject in the proposal and select the proposal type.
2. A section listing all authorities
3. A section listing all proposals which are currently active on the blockchain ledger. Each row contains a summary of the proposal. The rows are clickable, which will forward the user to a page with further details.

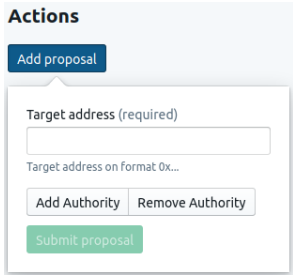


Figure A.7: The popup for adding new proposals as authority in the UI

Voting on Proposals

By clicking on the rows in section 3 of Figure A.6, the user gets forwarded to a page showing further details for the proposal. This panel is shown in Figure A.8 and contains the following sections:

1. A section with two buttons, allowing the user to vote on the proposal if they already have not. If the proposal has a majority vote, the user can click the second button to enact the proposal, triggering the action to be taken in the smart contract on the blockchain.
2. A section showing all the available details for the proposal.
3. A section showing the authorities which have voted on the proposal.

Managing Trust in Treatment Providers

Once we have access to a trusted authority, we can build a trust hierarchy in treatment providers. The page *treatment providers* under the *Authority* section of the sidebar contains this functionality. The page is shown in Figure A.9, and it contains the following features:

1. This section allows us to see all the treatment providers in which we have placed trust.
2. This section allows us to see all the treatment providers which we have not trusted.
3. Each card contains address information and if they are trusted. If a treatment provider is listed as trusted, it means that another trusted authority has trust in them.

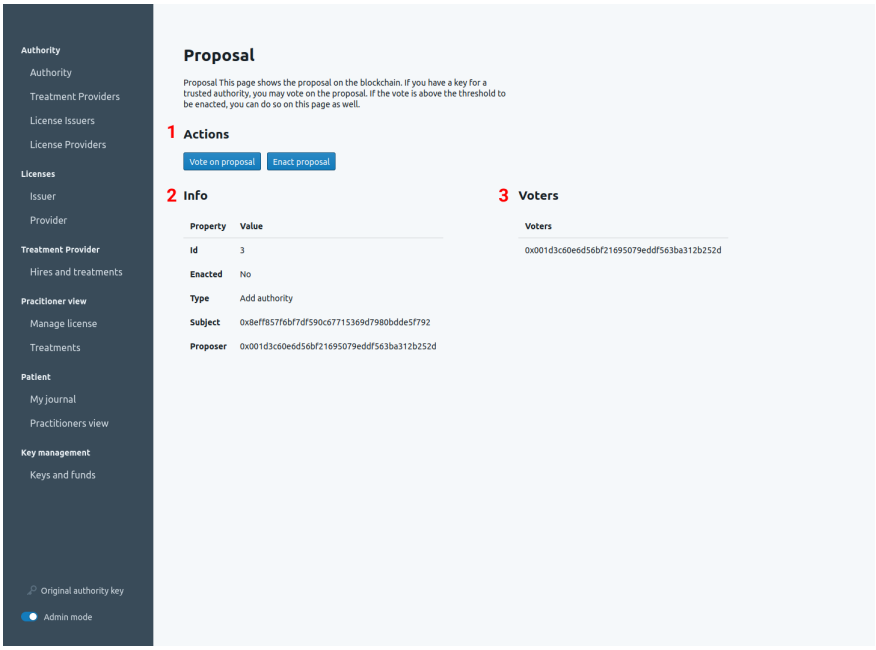


Figure A.8: A page in the UI for managing proposals related to distributed governance protocol for authorities.

4. Untrusted treatment providers are accounts who have registered themselves as a treatment provider but are yet to be trusted by an authority.

5. This button allows authorities to add trust in the provider. This trust should be rooted in a real-world process, such as an application process and an audit.

6. If an authority ever wants to remove a trust relationship in a treatment provider, they may remove it with this button. This should also be rooted in a real-world event, such as a bankruptcy or uncovered malpractice.

Managing Trust in License Issuers

Figure A.10 shows the page for managing trust in license issuers as an authority. The page follows the same flow and usage as presented for treatment providers in section A.3

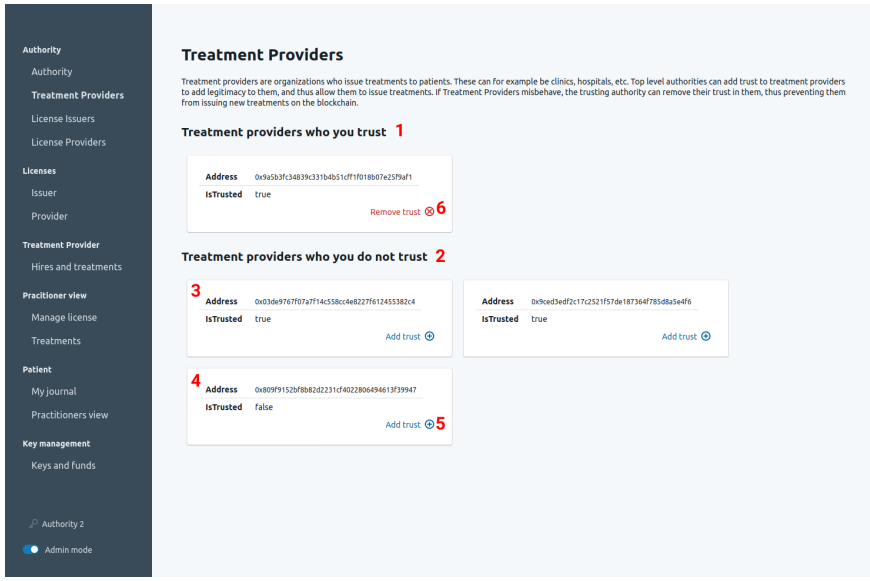


Figure A.9: Page in the UI for authorities to manage and view treatment providers and manage their trust in them.

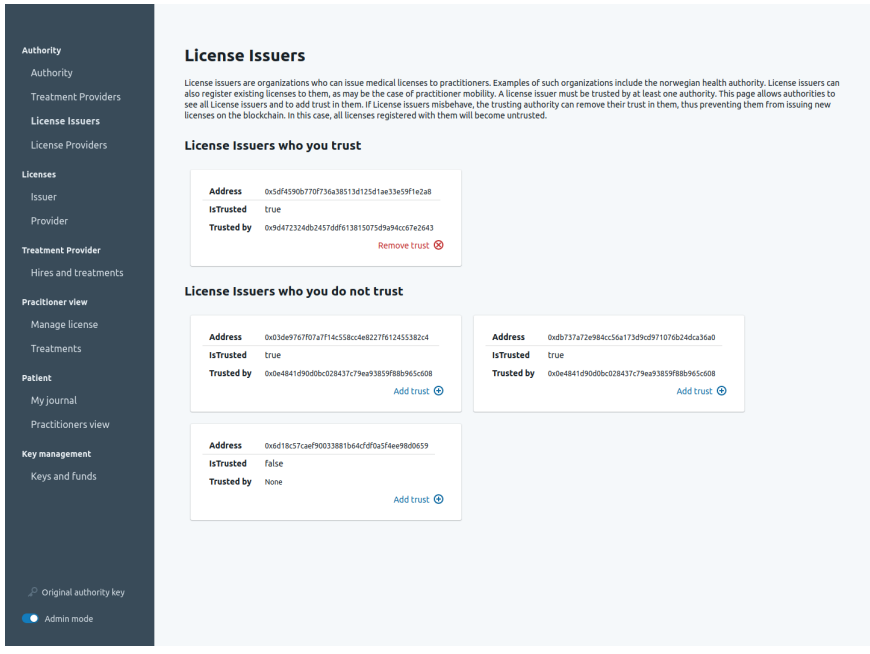


Figure A.10: Page in the UI for authorities to manage and view license issuers, and manage their trust in them.

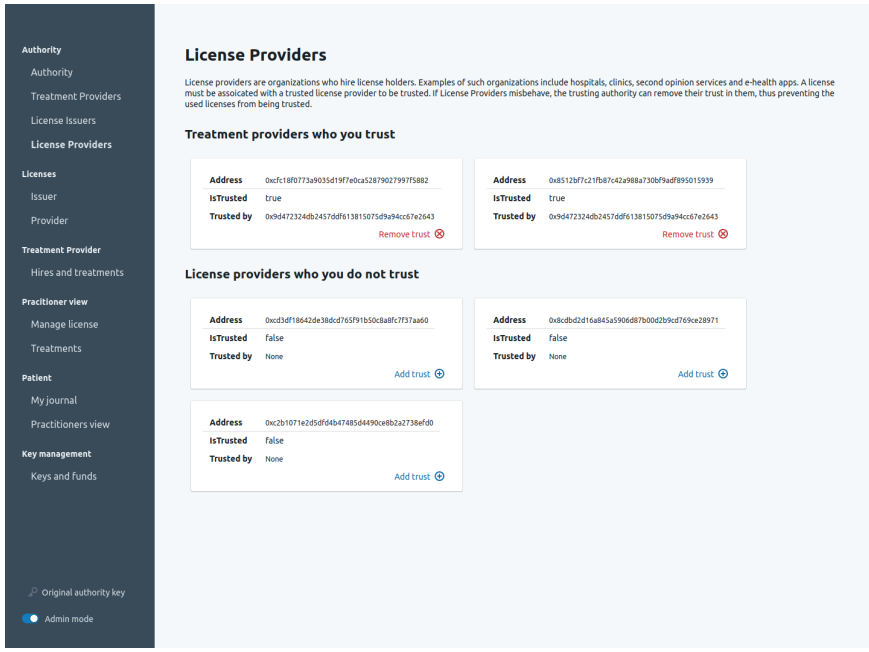


Figure A.11: Page in the UI for authorities to manage and view license providers and manage their trust in them.

Managing Trust in License Providers

Figure A.11 shows the page for managing trust in license providers as an authority. The page follows the same flow and usage as presented for treatment providers in section A.3

A.4 The License Issuer Stakeholder

License issuers are organizations that can issue medical licenses to practitioners. Examples of such organizations include units within national healthcare authorities. License issuers can also register existing licenses to them, as may be the case of practitioner mobility. A license issuer must be trusted by at least one authority. If License issuers misbehave, the trusting authority can remove their trust in them, thus preventing them from issuing new licenses on the blockchain. In this case, all licenses registered with them will become untrusted.

License Issuer Registration

The first action required by license issuers is to perform a registration step. When performed, the selected account (key-pair) will be registered as a license issuer,

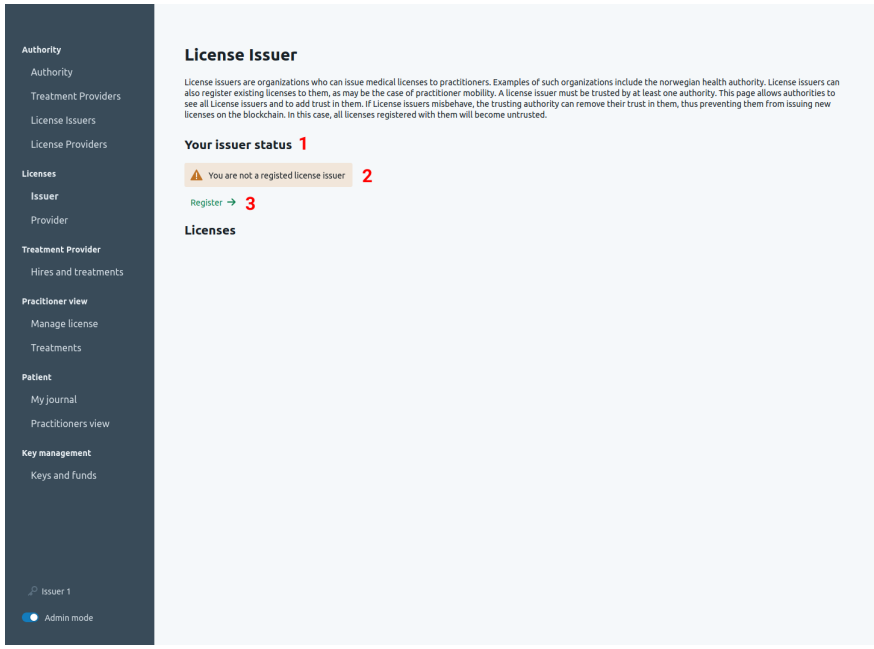


Figure A.12: Page in the UI for accounts to register themselves as a license issuer on the blockchain.

allowing authorities to see it and possibly place trust in it. Figure A.12 shows the treatment provider page when the user has yet to register. The page includes the following content:

1. A panel the current status of the account in the context of being a license issuer, here presented when the issuer is yet to register.
2. A notification showing the current status of the license issuer
3. A button allowing the selected account to register as a license issuer.

License Issuer Page

Once registered, the license issuer gets access to the full overview page of the license issuer. This page can be seen in Figure A.13. This page allows license issuers to see their current status as trusted/untrusted, issue licenses, see all licenses they trust, and all they do not trust. This functionality is present on the page in the following manner:

1. A panel showing the status of the license issuer. The notifications will show if the license issuer is registered and if an authority trusts the issuer.
2. License holders can propose to get their license moved to a new issuer. This can be the case during practitioner mobility.
3. Pressing this button allows the license issuer to approve a move of a license to themselves. This places the license under their trust domain.
4. This panel allows the license issuer to issue new licenses to accounts.
5. Pressing this button will create a new license for the given account address if they do not hold one from before. This creates a transaction on the blockchain, which will be public.
6. An overview of all licenses which are issued and trusted by the license issuer. Each card shows all available information about each license holder, including their trust status.
7. An overview of all licenses who are not issued license issuer. Each card shows all available information about each license holder, including their trust status.

Keep in mind that all actions are available to any registered license issuer, including issuing licenses. However, the issued licenses will not be trusted unless the license issuer themselves are trusted.

A.5 The License Provider Stakeholder

License providers are organizations who are the main health-service associated with license holders and are responsible for confirming the occupation of a healthcare worker. Examples of such organizations include hospitals and clinics. A license must be associated with a trusted license provider to be trusted. If License Providers misbehave, the trusting authority can remove trust in them, thus preventing their licenses from being trusted.

License Provider Registration

As is the case with license issuers, license providers must also register themselves first on the blockchain. This is done in a similar manner as shown in section A.4. The registration page is shown in Figure A.14. The page contains the following functionality:

1. This section shows the current status of the license provider. In this context, it only shows that the account is not registered as a license provider.

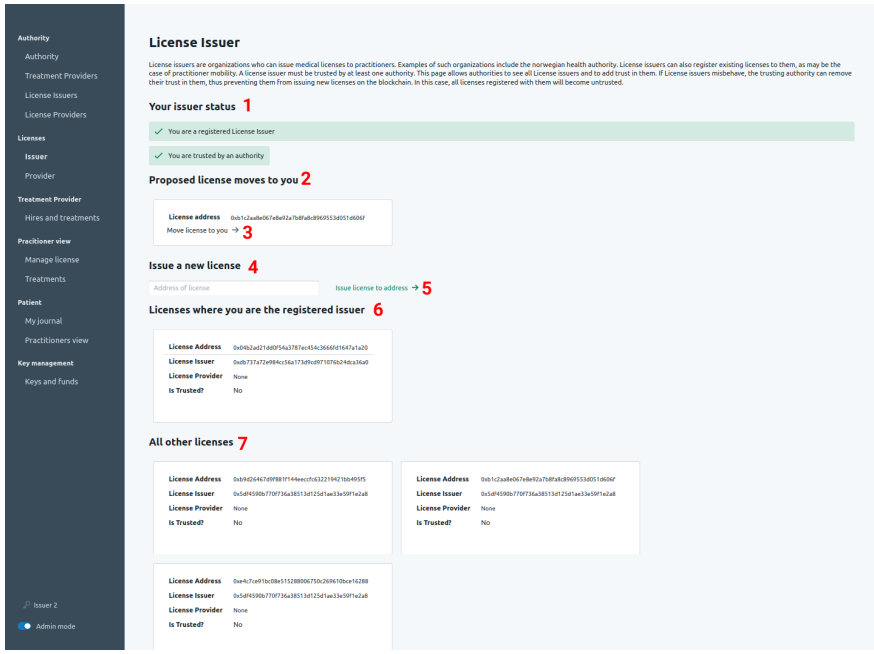


Figure A.13: Overview page in the UI for license issuers.

2. Clicking this button creates a transaction registering the selected account as a license provider.

License Provider Page

Once registered, the license provider gets access to the full overview page of the license provider. This page can be seen in Figure A.15. This page allows license provider to see their current status as trusted/untrusted, see all licenses they trust, and all they do not trust. This functionality is present on the page in the following manner:

1. A panel showing the status of the license provider. The notifications will show if the license provider is registered and if an authority trusts the license provider.

2. License holders can propose to get their license moved to a provider. This can be the case when practitioners change their main employer.

3. Pressing this button allows the license provider to approve a move of a license to themselves. This places the license under their trust domain.

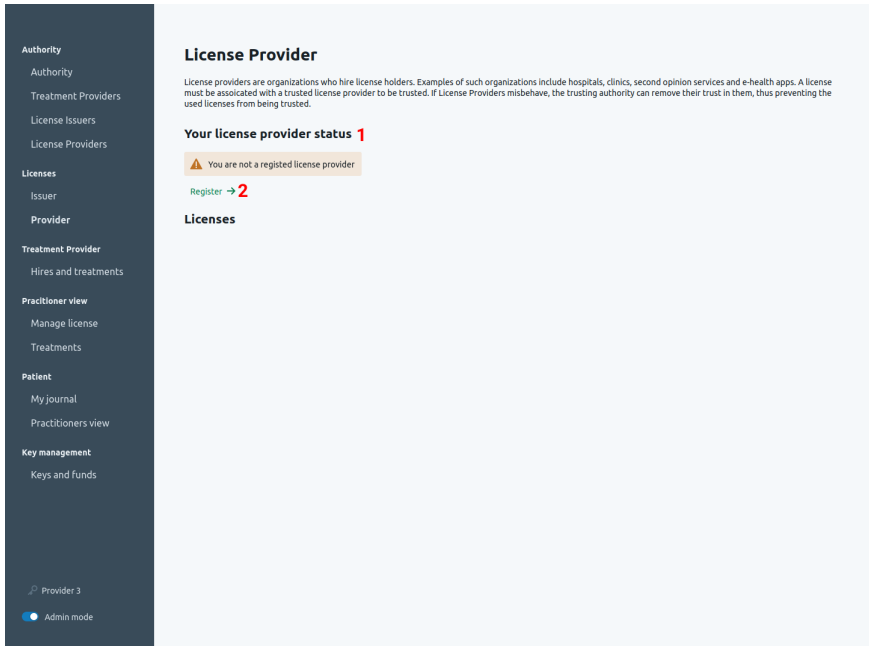


Figure A.14: Page in the UI for accounts to register themselves as a license provider on the blockchain.

4. An overview of all licenses which are trusted by the license provider. Each card shows all available information about each license, including their trust status.

5. An overview of all licenses that are not directly trusted by the license provider. Each card shows all available information about each license holder, including their trust status.

A.6 The Treatment Provider Stakeholder

Treatment providers are responsible for facilitating the interaction between the patient and the healthcare worker. It stores information about treatments and publishes summaries from these to the blockchain. To achieve this, the treatment provider hires a set of healthcare workers who can interact through them. They also notify patients about treatments pending their approval.

Treatment Provider Registration

Accounts must first and foremost register themselves as treatment providers on the blockchain. This is done via the panel shown in Figure A.16, containing the following

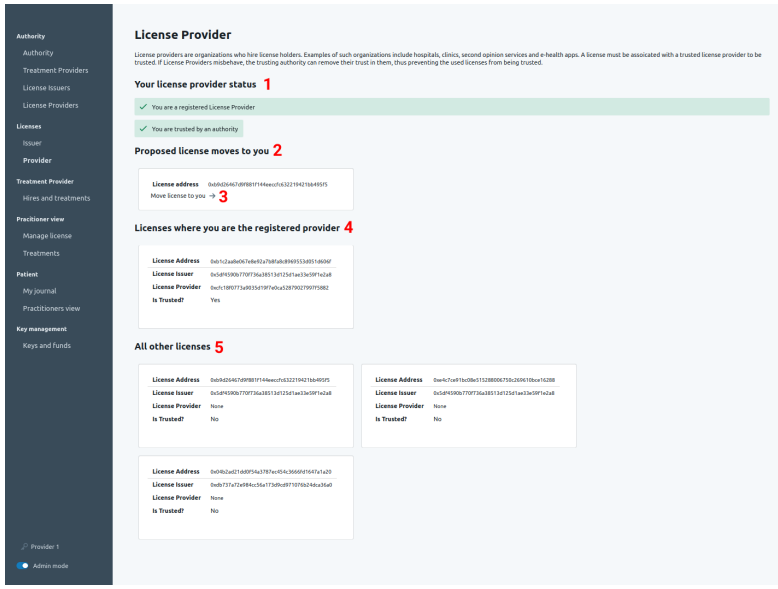


Figure A.15: Overview page in the UI for license providers

parts:

1. The button to click in the sidebar to access the panel.
2. A section showing information about the treatment provider
3. A button allowing the selected account to register as a treatment provider.

Treatment Provider Hires and Treatments

Once registered, the treatment provider becomes discoverable by authorities, which in turn can add their trust in them. This procedure has been shown previously in section A.3. Once trusted, the treatment provider can get access to the full page. An example of this is shown in in Figure A.16. This page contains the following additional parts:

1. A section for managing healthcare workers associated with the treatment provider.
2. A section showing the treatment providers associated with the treatment provider.

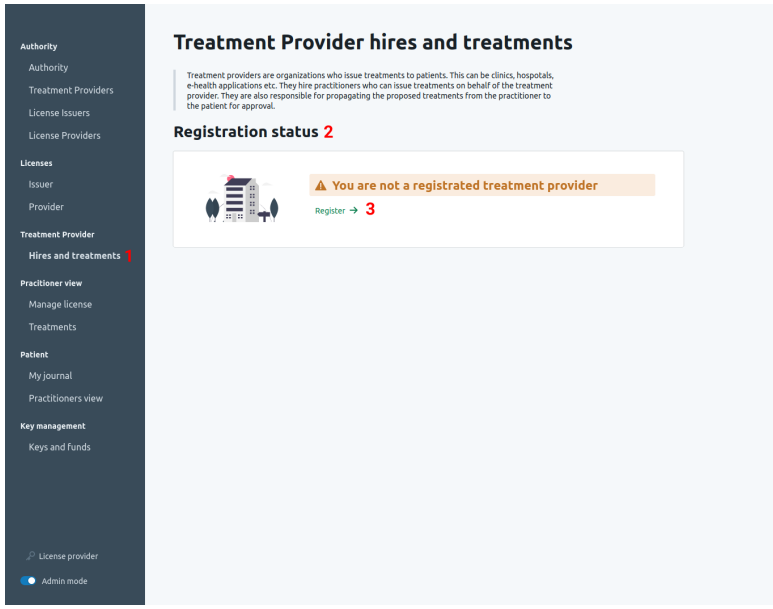


Figure A.16: Registration page in the UI for treatment providers

3. A section for associating new healthcare workers with the treatment provider.
4. The button to press for adding a new healthcare worker association.
5. A section for showing information about treatments delivered via the treatment provider.

A.7 The Healthcare Worker Stakeholder

Healthcare workers are the main subject of our platform. They are responsible for delivering care to patients. To be able to practice, healthcare workers must have a valid license issued to them, be trusted by a license provider, and be hired by a treatment provider. As time progresses, healthcare workers get associated with their issued treatments, which, in turn, is evaluated by the patient.

Getting a License

Figure A.18 shows the license management panel for healthcare workers. As healthcare workers cannot register themselves, a limited version is shown if no license is held. The panel has the following sections:

1. The button to click in the sidebar to access the panel.

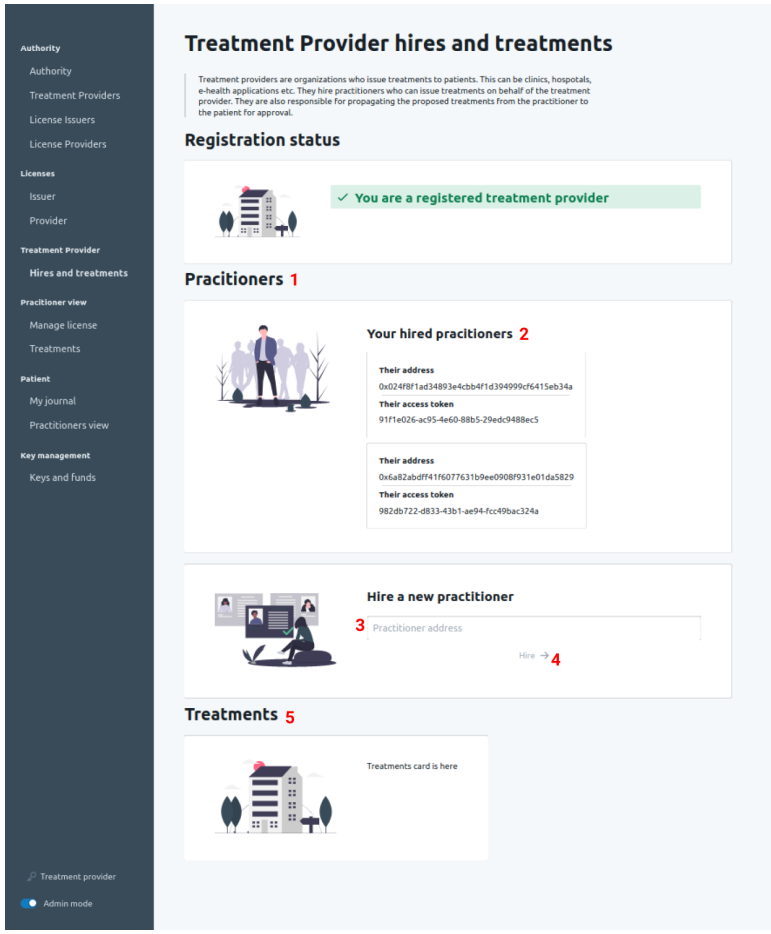


Figure A.17: Overview page in the UI for treatment providers

2. A section showing if the account has a trusted license.
3. A section showing information about the current association to license providers and issuers.

License Overview

Healthcare workers cannot register themselves. Instead, they have to share their account address with a License Issuer, who can issue a license to the healthcare worker. This process is show in previouslyA.4. Once registered, the healthcare worker can access the full license management page. This is shown in Figure A.19. The following additional sections are present:

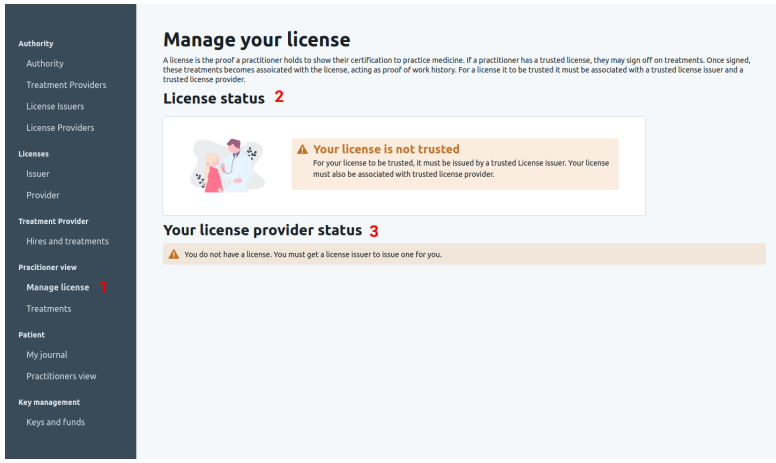


Figure A.18: Page in the UI for healthcare workers without an issued license

1. A section showing information about the current license issuer associated with the license.

2. A label showing if an authority trusts the license issuer.

3. An input field for proposing a change of license issuer. The address of the issuer must be provided.

4. A button for submitting the proposal to change license issuer.

5. A section showing information about the current license provider associated with the license.

6. An input field for proposing a change of license provider. The address of the provider must be given.

7. A button for submitting the proposal to change the license provider.

To become trusted, the license provider must first submit a proposal to be associated with a license provider. Once proposed, the given license provider must approve it. This procedure has been shown previously in section A.5. Once associated with a trusted provider, the page will transition into the view, as shown in Figure A.20, where we can see that 1) The license is trusted and 2) is associated with a trusted license provider.

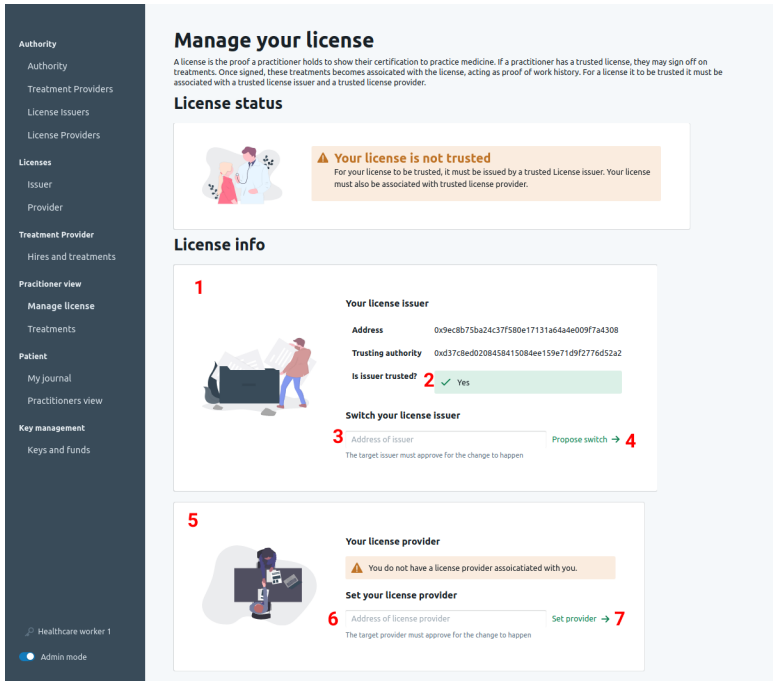


Figure A.19: Page in the UI for healthcare workers who have received a license

Managing Treatments

The other main panel for healthcare workers is shown in Figure A.21. This panel allows the healthcare worker to manage treatments for patients. The panel includes the following parts:

1. The button to click in the sidebar to access the panel.
2. A section showing the status of the selected accounts license.
3. A section for selecting the treatment provider to issue the treatment though.
4. A selection menu where the healthcare worker can choose between all their associated treatment providers.
5. A section for issuing treatments to patients.
6. A input box for the address of the patient.
7. A input box for the description of a treatment.

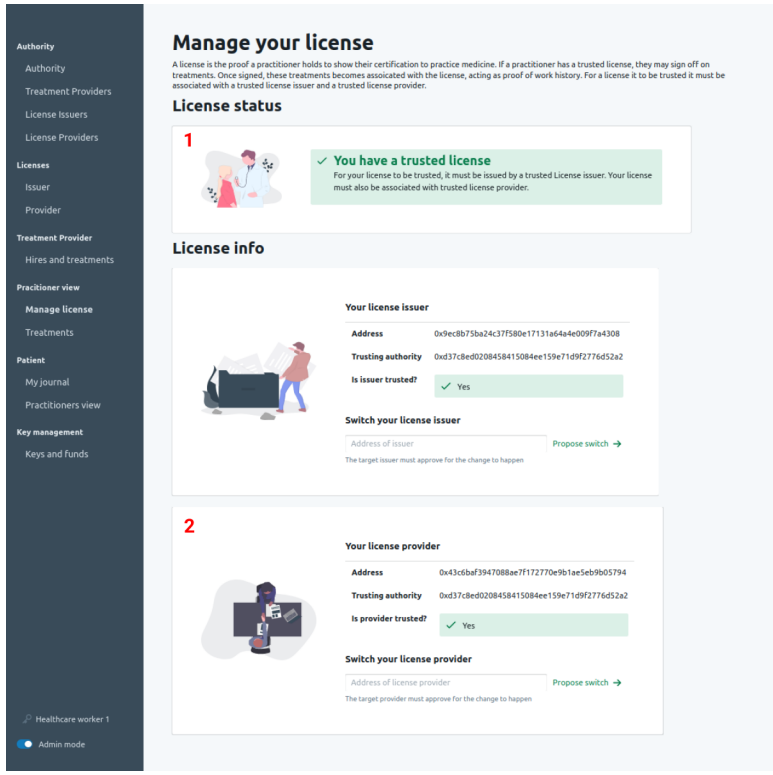


Figure A.20: Page in the UI for healthcare workers who have received a license and is trusted by a license provider

8. A button that submits the treatment to the treatment provider, which in turn will send it to the patient for approval.

9. A section showing the treatments which are pending the healthcare worker's approval.

Approving Treatments

Once treatments are approved by the patient and submitted to the blockchain, the healthcare worker may give final approval. Figure A.22 shows the approve treatments section if such treatments are present. The healthcare worker client will validate that the complete treatment as received from the treatment provider corresponds to the summary of the treatment as published on the blockchain. This is done by comparing the hash of the full treatment.

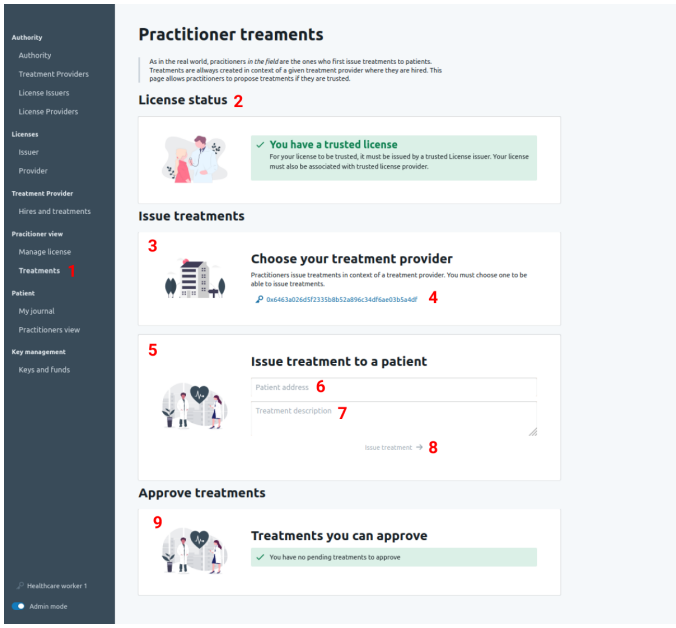


Figure A.21: Page in the UI for managing treatments by healthcare workers

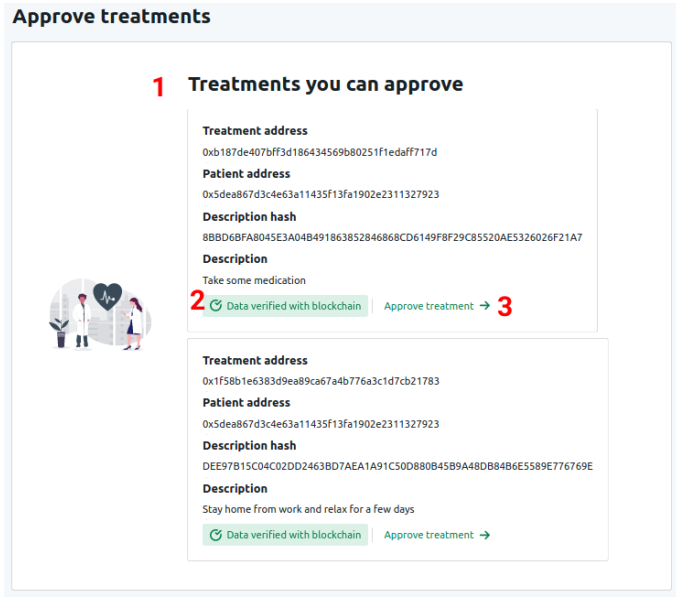


Figure A.22: Panel in the UI for approving treatments for healthcare workers

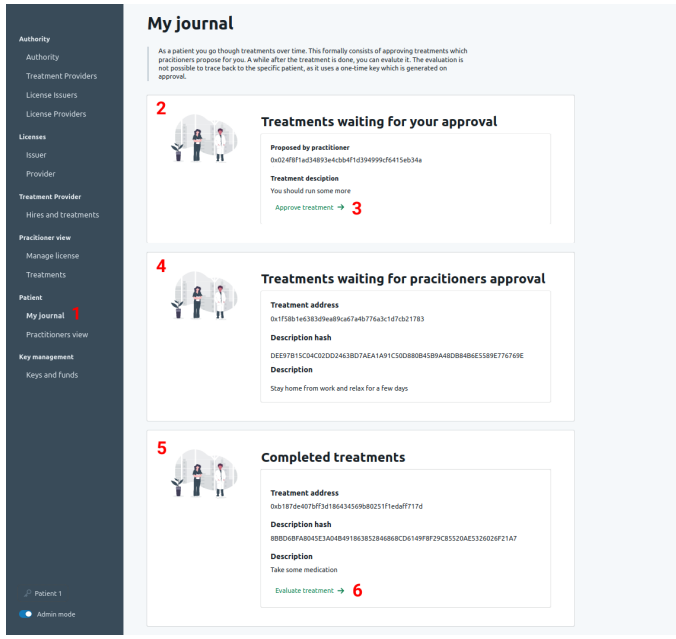


Figure A.23: Overview of journal page for patients in the UI

A.8 The Patient Stakeholder

The final stakeholder is the patient. They are the subject in treatments and can evaluate them after their completion.

Treatment Management for Patients

To get an overview of all proposed, pending, and completed treatments, the patient will use the panel, as shown in Figure A.23. This contains the following parts:

1. The button to click in the sidebar to access the panel.
2. Section showing all treatments pending the approval of the patient.
3. Button for approving a given treatment.
4. Section showing all treatments pending approval from a healthcare worker.
5. Section for showing all previous treatments which the patient has gone through.
6. Button for evaluating the given treatment.

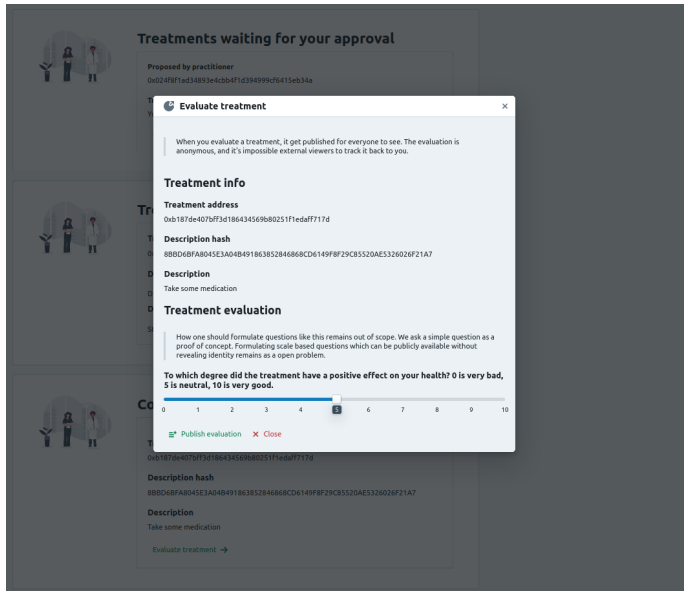


Figure A.24: Panel in the UI, allowing patients to evaluate treatments

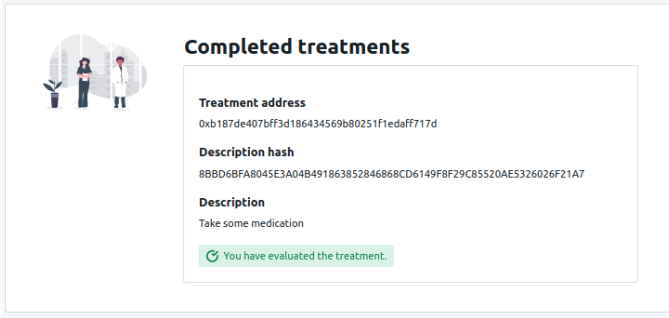


Figure A.25: Panel in the UI for showing the treatments evaluated by the patient

Evaluating a Treatment

Once a treatment is approved on the blockchain by a healthcare worker, the patient can evaluate it. This is done by pressing button 6) in Figure A.23, which will show the panel in Figure A.24. Here, the patient can give a simple metric for evaluating the treatment. Once evaluated, the treatment is marked as such in the overview panel, as shown in Figure A.25

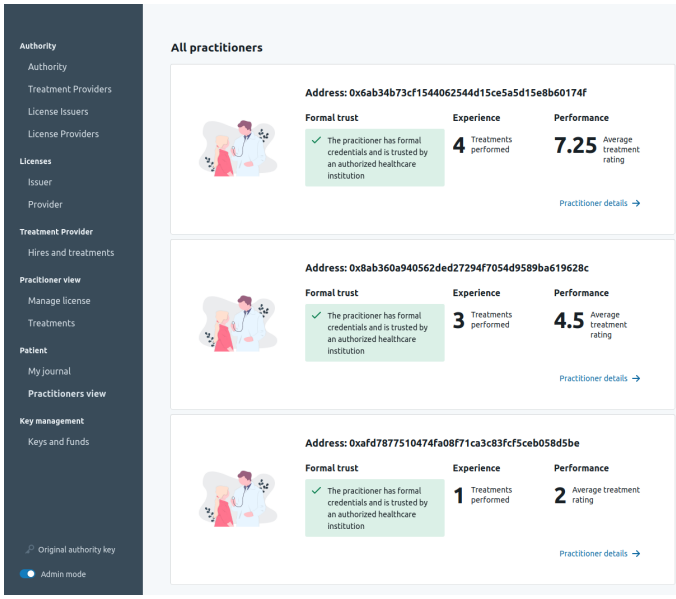


Figure A.26: Page in the UI for showing an overview of healthcare workers with a summary of metrics

A.9 Healthcare Worker Overview

The final panel is accessible for any user is the healthcare worker overview. The panel does not require an account for access. The panel, as shown in Figure A.1 shows an overview of all healthcare workers, along with a summary of their formal authority, experience, and competence. Authority trust is based on their current license status, showing if they are trusted or not. Experience is captured as the number of treatments approved. Finally, competence is given by the average rating from patients on the given treatments.

If the user wants further details about the underlying data for each healthcare worker, they can press the respective *practitioner details* button. This will show the panel in Figure A.27.

Figure A.27: Page in the UI for showing details about all data related to a healthcare worker

Appendix B
Application Guide for Administrators

We have developed a fully working proof-of-concept application for evaluating the architecture. During the application development process, we tried to keep usability for administrators in mind, where we tried to make the process of administering and running the application to be easy. This will allow for further development of the application, and allows third parties to test and set up the application easily. This appendix explains how to get hold of the code, how the codebase is structured, and how to get up running. We first cover a simple setup guide using docker-compose for testing. Afterward, we show how to set up the services manually, allowing for an easier development process.

B.1 The Codebase

All the code for the software application is found in the GitHub repository[1]. The codebase is a monorepo divided into folders representing different services. `ganache` is a single-node Ethereum network intended for testing purposes. `contracts` is a service containing the smart contracts and logic for compiling, testing, and deploying these to the blockchain. `providerService` is a spring-boot application providing a REST API. `webapp` serves a client-side rendered web application. The structure for how these services interact is shown in figure B.1. These servies should be started in the following order: `ganache`, `contracts`, `providerService` and `webapp`. In addition to the code, the repository contains a License and some documentation.

B.2 Setup with Docker

A `docker-compose.yml` file is provided at the root of the project. This file is designed to build and start all services in the correct order automatically. Running `docker-compose up` in the project root should be sufficient.

[1] https://github.com/jarensaa/transparent-healthcare, commit *34c09d9*

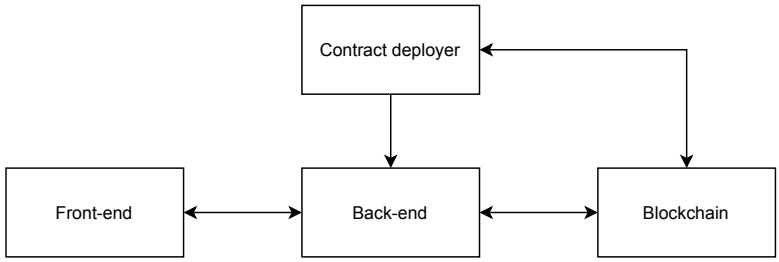


Figure B.1: An overview of the run-time presence for our implemented services

B.3 Starting Services Manually

When developing on the application, you probably want to run the services manually. To do this, follow this procedure:

1. Open a terminal window and navigate to the `ganache` folder.
2. Run `yarn start`
3. Open another terminal window and navigate to the `contracts` folder.
4. Run `yarn start`
5. Open another terminal window and navigate to the `providerService` folder.
6. Run `./gradlew bootRun`
7. Open another terminal window and navigate to the `webapp` folder.
8. Run `yarn start`

B.4 Accessing the Application

By default the web application is accessible at `http://localhost:3000` and the API is accessible at `http://localhost:8080`

Appendix C
System Testing Runbook

To validate that the system meets the functional requirements specified, we perform a system test. This test is performed by manually performing actions in the web application. This appendix presents a runbook with a step-by-step guide for reproducing this test. We refer the reader to the user guide in Appendix A for specifics on how each action is performed.

Application Setup

1. Clone the open-source repository[1]
2. Build the application by running `docker-compose build`
3. Start the application by running `docker-compose up`
4. Visit `http://localhost:3000` to load the web application

Setup Phase

We setup all the accounts and keys required for the test:

1. Ensure that the `admin mode` toggle found at the bottom of the sidebar is enabled.
2. Visit the panel `Key Management > Keys and funds`
3. Generate two keys stored on the server named `Authority` and `Secondary Authority`
4. Generate a key stored on the server named `License issuer`
5. Generate a key stored on the server named `License provider`

[1] https://github.com/jarensaa/transparent-healthcare, commit *34c09d9*

6. Generate a key stored on the server named `Treatment provider`
7. Generate three keys stored on the server named `Healthcare worker <1,2,3>`
8. Generate four patient (local) keys named `Patient <1,2,3,4>`
9. Send `0.1` in Ether from the `Original Authority Key` to the `Authority` key, `License issuer` key, `License provider` key, `Treatment provider` key and **healthcare worker** keys.

Registration Phase

The generated keys must register for their given roles:

1. Select the `License issuer` in the selection menu at the bottom of the sidebar.
2. Visit the `Licenses > Issuer` panel.
3. Click the `Register` button.
4. Select the `License provider` in the selection menu at the bottom of the sidebar.
5. Visit the `Licenses > Provider` panel.
6. Click the `Register` button.
7. Select the `Treatment provider` in the selection menu at the bottom of the sidebar.
8. Visit the `Treatment Provider > Hires and treatments` panel.
9. Click the `Register` button.

Adding an Authority

We first want to add our new authorities to the consortium of trusted authorities:

1. Note down the addresses of `Authority` and `Secondary Authority`
2. Select the `Original Authority Key` in the selection menu at the bottom of the sidebar.
3. Visit the `Authority > Authority` panel.
4. Click the `Add proposal` button.

5. Paste in the address of `Authority`, select `Add authority` and click `Submit proposal`
6. Wait until the proposal appears (Should take a few seconds) and click it.
7. Click the `Enact proposal` button.
8. Go back to the `Authority > Authority` panel.
9. Click the `Add proposal` button.
10. Paste in the address of `Secondary Authority`, select `Add authority` and click `Submit proposal`
11. Select the *Authority* in the selection menu at the bottom of the sidebar.
12. Wait until the proposal appears (Should take a few seconds) and click it.
13. Click the `Vote on proposal` button.
14. Click the `Enact proposal` button.
15. Go back to the `Authority > Authority` panel.
16. The list of authorities should now show three addresses.

Authority Setup Phase

We set up the trust relationships from the Authority:

1. Select the `Authority` in the selection menu at the bottom of the sidebar.
2. Visit the `Authority > Treatment providers` panel.
3. Add trust in shown treatment provider
4. Visit the `Authority > License issuers` panel.
5. Add trust in shown license issuer
6. Visit the `Authority > License providers` panel.
7. Add trust in shown license provider

Issuing Licenses

1. Select the `License issuer` in the selection menu at the bottom of the sidebar.
2. Visit the `Key management > Keys and funds` panel and note down the addresses of the healthcare workers.
3. Visit the `Licenses > issuer` panel.
4. For each healthcare worker, write the address into the input box and press `Issue license to address`.

Associating licenses with license providers

For each healthcare worker, do the following procedure:

1. Select the `healthcare worker` in the selection menu at the bottom of the sidebar.
2. Visit the `Key management > Keys and funds` panel.
3. Visit the `Practitioner view > Manage License` panel.
4. In the `Your license provider` section, enter the address of the treatment provider and click `Set provider`.

Then, access the transfer to the license provider:

1. Select the `license provider` in the selection menu at the bottom of the sidebar.
2. Visit the `Licenses > Provider` panel.
3. Accept all the proposed license moves.

Treatment Provider Setup

1. Select the `Treatment provider` in the selection menu at the bottom of the sidebar.
2. Visit the `Treatment provider > Hires and treatments` panel.
3. Go to the section `Hire a new practitioner`.
4. For each healthcare worker, enter their address and press the `hire` button.

Issuing Treatments to Patients

Repeat the following procedure as many times as you want:

1. Select the key of a `Healthcare worker` in the selection menu at the bottom of the sidebar.
2. Visit the `Practitioner view > Treatments` panel.
3. Go to the `Choose your treatment provider` section, and select the treatment provider from the selection menu.
4. Go to the `Issue treatment to a patient` section, type in the patient address, a description (of at least 10 characters in length) of the treatment and press `issue treatment`.
5. Select the key corresponding to the patient which the treatment was issued to in the selection menu at the bottom of the sidebar.
6. Visit the `Patient > My journal` panel.
7. Approve the proposed treatment in the `Treatments waiting for your approval` section.
8. Select the key of the same healthcare worker as in step 1).
9. Visit the `Practitioner view > Treatments` panel.
10. Go to the `Treatments you can approve` section and click `approve treatment`
11. Select the key of the same patient as in step 5)
12. Visit the `Patient > My journal` panel.
13. Go to the `Completed treatments panel` and click the `Evaluate treatment` button.
14. Drag the slider to a random rating, and press the `Publish evaluation` button.

Appendix D
VerifyMed Conference Paper

The work in this thesis resulted in a conference paper describing the VerifyMed platform. The paper was submitted to the *2nd Blockchain and Internet of Things Conference (BIOTC 2020)*, to be held in Singapore from 8th to 10th of July 2020. After being reviewed by international experts in related fields, the paper was accepted for presentation. The remainder of this appendix contains the final version of the paper.

VerifyMed - A blockchain platform for transparent trust in virtualized healthcare: Proof-of-concept

Jens-Andreas Hanssen Rensaa
Dep. of Inf. Sec. and Comm. Techn.
Norwegian University of Science and Technology, (NTNU)
jens.rensaa@gmail.com

Danilo Gligoroski
Dep. of Inf. Sec. and Comm. Techn.
Norwegian University of Science and Technology, (NTNU)
danilog@ntnu.no

Katina Kralevska
Dep. of Inf. Sec. and Comm. Techn.
Norwegian University of Science and Technology, (NTNU)
katinak@ntnu.no

Anton Hasselgren
Dep. of Neuromedicine and Movement Science
Norwegian University of Science and Technology, (NTNU)
anton.hasselgren@ntnu.no

Arild Faxvaag
Dep. of Neuromedicine and Movement Science
Norwegian University of Science and Technology, (NTNU)
arild.faxvaag@ntnu.no

ABSTRACT

Patients living in a digitized world can now interact with medical professionals through online services such as chat applications, video conferencing or indirectly through consulting services. These applications need to tackle several fundamental trust issues: 1. Checking and confirming that the person they are interacting with is a real person; 2. Validating that the healthcare professional has competence within the field in question; and 3. Confirming that the healthcare professional has a valid license to practice. In this paper, we present VerifyMed - the first proof-of-concept platform, built on Ethereum, for transparently validating the authorization and competence of medical professionals using blockchain technology. Our platform models trust relationships within the healthcare industry to validate professional clinical authorization. Furthermore, it enables a healthcare professional to build a portfolio of real-life work experience and further validates the competence by storing outcome metrics reported by the patients. The extensive realistic simulations show that with our platform, an average cost for creating a smart contract for a treatment and getting it approved is around 1 USD, and the cost for evaluating a treatment is around 50 cents.

CCS CONCEPTS

• **Applied computing** → **Health informatics**; • **Security and privacy** → *Social aspects of security and privacy.*

KEYWORDS

Blockchain, Healthcare, Trust, Ethereum

BIOTC 2020, July 08–10, 2020, Singapore

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

ACM Reference Format:
Jens-Andreas Hanssen Rensaa, Danilo Gligoroski, Katina Kralevska, Anton Hasselgren, and Arild Faxvaag. 2020. VerifyMed - A blockchain platform for transparent trust in virtualized healthcare: Proof-of-concept. In *BIOTC 2020: 2nd Blockchain and Internet of Things Conference, July 08–10, 2020, Singapore.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

The healthcare industry is currently ongoing through a digital transformation, and innovations within information, and communication technologies have enabled the healthcare industry to improve the delivery of health services. Similar to Industry 4.0, Healthcare 4.0 [20] aims to use modern digital technologies to enable a virtualized healthcare environment by providing distributed and patient-centered care delivery. This virtualization is expected to accelerate with the emergence of next-generation mobile network strategies (5G) and artificial intelligence (AI), enabling virtualized care services to be executed in real-time and performed based on real-time data collection from anywhere at any time.

The transition to virtualized health services poses some challenges; one of them is providing trust. In the current healthcare environment, most patients meet physically with healthcare workers in accredited healthcare institutions. However, when the meeting is moved to a virtualized environment, this inherited trust is decreased. Furthermore, building up such a trust relationship is even harder if the caregiver is an AI health worker. The health domain, therefore, needs new solutions to enable an establishment of trust between patients and healthcare workers in a virtualized environment.

Blockchain is a maturing technology with properties that can provide trust within a virtualized health domain as it allows mutually mistrusting entities to interact without the presence of a central trusted third party. While initially intended for the financial domain, the addition of smart contracts allow for general purpose applications to be made. By creating and deploying smart contracts, we can build models that capture the authorization and the experience of a healthcare worker directly on the blockchain. These models can then be used to establish trust in a patient-caregiver encounter.

When it comes to building network applications that capture and nourish the complex relations of trust among different entities, a related area is the area of distributed database systems. More than a quarter of a century ago, the trust-management approach to the authentication and access control of distributed-systems was proposed in [3]. It addressed the inefficiency and inadequacy of traditional authorization mechanisms. However, a real renaissance in the trust-management approach of distributed database systems is happening with the introduction of blockchain technology. As described in [17], in the last decade, we witnessed a mutual influence and development between database technology and the blockchain technology. In particular, the blockchain technology has influenced the introduction of new functionalities in some modern databases such as immutability, privacy, and censorship resistance. Those blockchain functionalities are precisely the ones that we valued the most in this work.

Our contribution: We describe the design rationale, implementation and evaluation of VerifyMed - a proof-of-concept for transparently validating the authorization and competence of healthcare workers by using blockchain technology[1].

First, we identify the issues related to data sharing and trust establishment and maintenance in a virtualized healthcare environment. Second, we define the requirements that the proposed application has to meet. These requirements are mapped to solutions provided with blockchain and smart contracts. Last but not least, we present the proposed architecture, its implementation and evaluation. To our knowledge, this is the first proof-of-concept designed to enhance trust in a virtualized healthcare environment by utilizing blockchain technology.

We propose three types of evidences for building trust in a virtualized healthcare environment such as evidence of authority, evidence of experience and evidence of competence. Our design uses the public Ethereum blockchain platform, where transactions cost some amount of cryptocurrency. We evaluate the performance of our platform in terms of this cost.

2 RELATED WORK

Extensive research on the use-cases of blockchain within the health domain has been done in recent years [1, 10, 13]. The technology is generally proposed as a solution to the data management problems within the health domain, and it is shown as especially well suited for the data sharing problems. These problems relate to challenges with interoperability, security and mobility. The majority of the research on blockchain in healthcare has focused on managing electronic health records [10].

MedRec[2] is a proof-of-concept application that relies on the existing data-infrastructure within the healthcare domain. It uses blockchain as a public registry for data sharing and access control of Electronic Medial Records (EMRs). The registry is used to store a simple mapping between a pseudonymous patient identifier, healthcare providers and pointers to EMRs. Overall, this architecture allows patients and organizations to locate and access data from a range of providers given a patient's consent. **Ancile**[5] is a system for controlling access to EMRs, and tries to solve the same problem as MedRec. It improves the previous solutions by including a key-management mechanism for symmetric keys to encrypt the data stored at providers. The system is designed for a permissioned Ethereum-based blockchain, but does not specify the underlying platform further. Permissions for access and participation in the blockchain are governed through a distributed governance mechanism where a pool of voter nodes controls these permissions.

Reference [23] defines a set of metrics which can be used for the evaluation of blockchain applications within the health domain. That reference also describes some fundamental principles which should be applied when creating decentralized applications. Although the work is directed towards the American Health Insurance Portability and Accountability Act, the framework can be generalized to a set of specific requirements that can be applied to the European setting. References [4, 8, 15] focus on establishing trust in healthcare through blockchain, but they only present conceptual analysis and do not include a practical implementation, design or proof-of-concept.

2.1 Using blockchain for trust in healthcare

To ensure that evidence for a health-workers trust is credible, we can use a blockchain platform for their storage. Blockchains offer data storage, which is immutable and highly distributed, making them easy to access. Blockchain technology has been coined as a key enabling technology for better data-sharing and interoperability within the healthcare industry [9]. It can potentially enable patients and healthcare institutions to share, index and control access to data in a fully distributed manner. Blockchain platforms are also, by its nature highly available, and easily accessible by all participants in the blockchain network.

Through the means of smart-contracts, we can create a distributed application running directly on a blockchain platform. These smart-contracts can be used to store the proofs of a health worker's authority, experience and competence while incorporating access control mechanisms that ensure that the published data is credible. The data uploaded via these contracts is immutable, resulting in the proofs of trust that are non-reputable. The non-repudiation property denies any change to the proofs once published, disabling health-workers to alter their proofs fraudulently. Through the use of a public platform, data can be easily available to patients and healthcare institutions, allowing them to validate proofs on-demand.

The Ethereum blockchain is the most popular blockchain platform to incorporate the concept of smart contracts [6]. As a consequence, it offers a rich suite of developer tools, enabling rapid prototyping and testing. It, therefore, offers a compelling value-proposition for creating proof-of-concept applications. Although a permissionless blockchain platform in nature, developers stand free to implement their own permission structure within smart contracts to limit how data can be published. Smart contracts on the blockchain can interact with each other, enabling the creation of complex architectures with a rich set of features.

2.2 Patient reported outcomes

One way to measure outcomes from a given treatment and clinical recommendations is through Patient Reported Outcomes (PROs) [21]. There are two standardized manners to measure PROs: Patient Reported Outcome Measures (PROMs) and Patient Reported

[1] As an online addition to this article, the source code and instructions for setting up the platform are available at https://github.com/jarensaa/transparent-healthcare

Experience Measures (PREMs) [12]. They, among other factors, can measure the functional status associated with a treatment or the healthcare which the patients have received. We have chosen the latter to capture the patient experience metrics related to the virtual interaction with the caregiver. These can, for example, be satisfaction rates for patients' experience with their treatment, the health-worker or the virtual setting of the healthcare institution. By creating a link from the healthcare worker to treatment to experience, we can create a model for healthcare worker experience (number of treated cases) and competence (PREMs).

3 USED CRYPTOGRAPHIC COMPONENTS

VerifyMed uses the Ethereum blockchain [22] to store data about trust relationships, treatments and evaluations within the health domain. To achieve this, we rely on many cryptographic primitives. Some are used directly, while others are key pieces to understand the underlying workings of the Ethereum blockchain and the tools used to interact with it. For the overall security of our platform, we followed the design and engineering principles of applied cryptography [19] and we assumed that all security features are inherited from the Ethereum blockchain. While proving the security of some particular and specific relations in our platform is an important issue, in this proof-of-concept stage of the development, it was out of the scope of our work. In that sense, since the purpose of this paper is not to be a tutorial for the cryptographic concepts and primitives that are used in blockchain, we refer an interested reader to some systematization of knowledge publications such as [16] and to follow the references there. Yet, we can say that VerifyMed uses the following cryptographic primitives:

- Cryptographic hash functions (Ethereum uses the NIST approved SHA-3 hash function [7]);
- Merkle trees (Ethereum uses a generalized form called Modified Merkle Patricia Trees);
- The ECDSA digital signature scheme [11].

3.1 Smart contracts in Ethereum

The main intention of the Ethereum blockchain platform is to enable the creation of general-purpose decentralized applications. The platform can be seen as a state-machine with a set of valid transitions triggered by transactions. Each individual transaction submitted to the blockchain alters the state through the function:

$$\sigma_{t\ 1} = \Upsilon(\sigma_t, T) \tag{1}$$

where σ (i.e. $\sigma_0, \sigma_1, \ldots$) is the *global state* for the Ethereum blockchain platform, often described as the *world state*. The function Υ is the *Ethereum state transition function*, which produces a new *world state* based on the current *world state* and a *transaction* T. To ensure that all nodes participating in the blockchain network can deduce the same *world state* σ, they must all agree to a fixed ordering of transactions $S = [T_0, T_1, T_2, ...]$. Given that such an ordering is shared and agreed upon, all nodes may deduce the same world state by using the transition function over all of these transactions:

$$S = [T_0, T_1, T_2, ...] \tag{2}$$

$$\sigma_t = \Upsilon(\Upsilon(\Upsilon(\sigma_0, T_0), T_1), T_2)... \tag{3}$$

The purpose of the blockchain ledger and consensus mechanisms is to allow the nodes in the network to agree to such a transaction order. The ledger follows a general structure where each block contains a set of ordered transactions which are cryptographically bound to the block via a root hash. With the introduction of blocks, we must alter the *world state* update function:

$$B_b = (..., (T_{b1}, T_{b2}, ...), ...) \tag{4}$$

$$\sigma_b = \Omega(B_b, \Upsilon(\Upsilon(\Upsilon(\sigma_{b-1}, T_{b0}), T_{b1}), T_{b2})...) \tag{5}$$

where σ_b is the *world state* after block B_b is processed. The block B_b contains the transaction set, along with the remaining data bound to the block. The *Block transition function* Ω combines the state changes from transactions and the block (e.g. rewards given to the miner of the block) and generates a new *world state* σ_b.

The main differentiating factor of the Ethereum blockchain platform in comparison to the popular Bitcoin platform is the expressiveness of the *world state* and the ability of users to create smart contracts to utilize this expressiveness. Smart contracts allow users to append their own programs to the blockchain ledger. These programs act like an additional state-machine on top of the existing infrastructure, with their own set of valid transaction types.

The composition of the Ethereum blockchain platform follows the same fundamental principles of the Bitcoin blockchain platform. However, a fundamental understanding of details related to five different concepts in Ethereum is required and we urge the reader to study reference [22] for the following Ethereum concepts: 1. Accounts; 2. Smart contracts; 3. Transactions; 4. Costs, and 5. The Ethereum ledger construction.

4 DATA SHARING AND TRUST ESTABLISHMENT IN VIRTUALIZED HEALTHCARE ENVIRONMENT

A fundamental problem within the health domain is the low capability to share data between healthcare institutions and services. This problem has multiple underlying root-causes, where each can be addressed with a different individual solution. References [10, 14] have defined four healthcare industry requirements where blockchain can be a significant contributing factor to improvement. Here we explain the same requirements but from a perspective of a healthcare worker.

(1) **Interoperability:** Data is not organized in a way that is easily shareable and transferable between institutions. In particular, the data related to the healthcare worker is stored in fragmented data stores where formats and access methods vary from organization to organization. Building applications that can access and integrate all this data together to form an evidence for trust is therefore challenging.
(2) **Security:** As more digital health data is produced and shared, higher security requirements are imposed. Data providing an evidence for the experience of a healthcare worker is stored in context to patients. Security mechanisms such as access control are therefore patient-centered, making it difficult to access in the context of a health worker without manual intervention.
(3) **Data sharing:** Due to interoperability and security requirements, it is difficult for patients and healthcare workers to gain access to all their data in a unified view. As healthcare workers change employers, their data documenting

their work-history does not follow them. Thus, the evidence of their work-history becomes increasingly fragmented between different organizations over time.

(4) **Mobility:** A patient traveling between countries, changing services or switching their health domain should be able to transfer his/her data from one health institution to another. Likewise, practitioners should be able to transfer data related to their experience, credentials and practice between institutions. The inability to share the work history of healthcare workers, may limit their ability to move across borders and jurisdictions. Gaining formal certifications and licenses can thus take a long time, reducing the overall efficiency of the healthcare workers and increasing the costs related to recruitment and on-boarding for healthcare institutions.

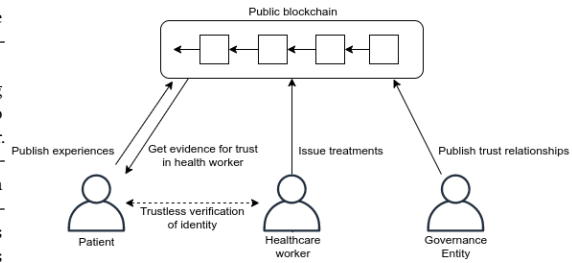


Figure 1: Interacting with the blockchain to gain trust in a health worker

4.1 Trust in a virtualized environment

There is an inherent trust relationship between a patient and healthcare worker in the setting of a physical meeting in a healthcare institution: The patient often trusts that the person in front of him/her in a white coat is an authorised medical professional and the healthcare worker trusts that the patient is whom he/she claims to be, often verified with physical ID [18]. The same trust relationship could be extended into a virtualized environment when the patient is talking with a practitioner that the patient already knows from a previous physical setting, and the healthcare worker knows that the patient is who he/she claims to be. Although in a virtualized healthcare environment where the virtual interaction is the first meeting, this same principle cannot be used. Thus, there is a need for establishing such trust relationships in a virtualized healthcare environment.

To enable trust in a virtualized world, the trustee must be provided with an evidence. This evidence is the ground that justifies a trust relationship between the trustee and the trusted. In the context of the patient-caregiver relationship, we can define the following three major evidences that could enhance trusts:

(1) **Evidence of authority:** The healthcare workers must be able to show that they have formal credentials allowing them to practice as healthcare workers. They need a formal license, and their background must be legitimate and approved.
(2) **Evidence of experience:** The healthcare workers will have the possibility to verify their experience required to deal with the specific health issue of the patient. As specialization increases, this evidence will increasingly be an essential ground for trust.
(3) **Evidence of competence:** In addition to being experienced within the medical problem in question, the healthcare workers should be able to show that they have previously delivered positive experiences to other patients. Thus, a metric for patients satisfaction is another crucial evidence.

By making these evidences available to the patients, the grounds for trust between the patient and the healthcare worker can be established. However, designing such a solution is not trivial due to the major requirements defined within the health domain: interoperability, security, data sharing and mobility. These requirements make it challenging to create an application that works on top of the existing organizational structure where data is fragmented over different organizations with a diverse set of formats.

Making data about healthcare workers' authority, experience and competence transparent, available and immutable can be perceived as a privacy issue for healthcare workers. However, this structure also has some significant advantages for the healthcare worker. Due to the availability of data, turnover, on-boarding and mobility processes can be simplified due to the ability of employers to perform efficient background checks related to their profession. They can also have better visibility, providing a major incentive to provide better care.

5 DESCRIPTION OF THE ARCHITECTURE

Our proposed system architecture is designed to store Evidences of Authority, Evidences of experience and Evidences of competence on the public Ethereum blockchain platform. To enable these evidences to hold any legitimacy, the application incorporates a concept of governance, where a set of stakeholders cooperate to create a trusted environment on the blockchain via smart contracts. Patients use this trusted environment to gain evidence for trust in a health worker, and publish their own experiences once the patient and health worker interaction is completed. While the high-level view, as shown in Figure 1, is simple, the underlying system design is of high complexity. The first contributor to increased complexity is the real-world trust relationships within the healthcare system. While our top level model depicts a single *governance entity*, no such entity exists in the real world. The trust relationships within the healthcare industry include a broad set of different organizational entities. These entities hold specific responsibilities, and they can only together create overall trust in the healthcare system. Our system architecture includes multiple organizational entities, and we capture the trust relationships between them on the blockchain.

Other requirements relevant to our platform include patient privacy, prevention of fraudulent patient evaluations and scalability considerations. These quality attributes can only be addressed through architectural choices, furthermore contributing to complexity. After describing our overall system architecture and our choices, we will break this down into procedures and subsections to show how the architecture addresses these requirements.

5.1 Modeling evidence for trust

5.1.1 Evidence of authority.

The first evidence for trust in a healthcare worker is the evidence of authority. This evidence consists of the formal credentials which allow the healthcare professional to practice. By providing this evidence on a blockchain, patients or any other interested stakeholder can access it freely. If the patient can confirm the link between a healthcare worker and the evidence of authority on the blockchain, he/she should be able to trust that the healthcare worker has formal authorization. In practice, we choose to model the evidence of authority as two different statements which the healthcare worker wants to prove:

(1) The healthcare worker is currently in possession of a valid License for Health Personnel in the area he or she operated and is thus formally qualified to practice.
(2) The healthcare worker is formally associated with an authorized healthcare facility.

Both of these statements cannot be fulfilled by the healthcare worker alone. They are instead statements of trust from other organizational entities that are deemed trusted themselves. This structure of entities and their trust relationships quickly serves as the foundation of trust in the system. We define the following stakeholders that create one hierarchy of trust:

- **Authorities** are top-level healthcare authorities responsible for the formal authorization of healthcare institutions, educational facilities and other organizations who provide healthcare related services. Organizations with such authorities are usually the national health directorates. These organizations organize themselves via a distributed governance protocol.
- **License Issuers** are organizations that are responsible for the formal authorization of healthcare workers. They are responsible for background checking of the applicant and use their documented experience and performance to decide if the healthcare worker is fit to hold a License. If that is the case, they choose to issue such a license and thus establish a trust relationship with the healthcare worker. Such organizations are often units within a national health directorate.
- **License Providers** are authorized healthcare facilities responsible for the practice of the healthcare worker on a day-to-day basis. These facilities are under continuous evaluation by the authorities and have to ensure the competence of their associated healthcare workers. Such organizations can include hospitals or clinics.
- **Treatment Providers** are health service providers who are responsible for facilitating the interactions between patients and healthcare workers. They hold the main responsibility for authenticating patients and for storing data related to the interaction. Such stakeholders may be similar to license providers, like hospitals or clinics. It can also include services such as e-health platforms and secondary consultation services.
- **Licenses** are the components that represent the healthcare workers within our trust model. A license can only be created by a license issuer, and it is tied to credentials (keys) in possession of the health worker. Once issued, it may be transferred between License Providers, License Issuers and associated with additional Treatment Providers if these stakeholders agree to these movements.

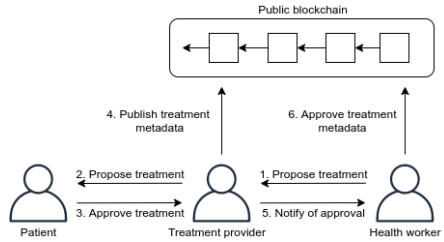


Figure 2: A model for generating evidences for the experience of health workers

Together, these stakeholders interact and build a complete trust hierarchy. This model is captured via smart contracts deployed to the blockchain ledger, storing data about stakeholders and their trust relationships.

5.1.2 Evidence of experience. The second evidence for trust in healthcare workers is their experience. Depending on the context in which the patient meets a healthcare worker, experience within a relevant field may be of high importance to ensure that the healthcare worker can deliver the care required. The metrics for experience come in either qualitative or quantitative forms. The qualitative evidence can be conveyed through certifications, while the quantitative evidence can be deduced from metrics such as the number of a specific treatment performed by the healthcare worker or the number of specific problems addressed. To model the evidence of experience, we choose to focus on quantitative metrics.

The goal of our model is to expose the number of treatments performed by the healthcare worker to the patient. Evidence of authority is created by a formal model for creating an evidence. In contrast, evidence of experience is generated through patient and healthcare worker interactions. Each new interaction resulting in a treatment thus forms evidence for future patients who want to interact with the healthcare worker. Figure 2 shows our model for publishing treatment information on the blockchain. During the patient and healthcare worker interaction, the treatment provider is responsible for conveying information about treatments recommended by a healthcare worker. Once approved by a patient the full content of the treatment is stored at the treatment provider. Metadata about the treatment is published to the blockchain, which is in turn approved publicly by the healthcare worker, thus forming a public link from the healthcare worker to the treatment. Over time, this process will generate a public log capturing metadata about treatments performed by a health worker, which serves as the evidence of experience.

5.1.3 Evidence of competence. While a quantitative metric like a number of treatments can be evidence for experience, it does not represent the quality of these treatments. Patient Reported Experience Measures (PREMs) is a standardized way to measure the outcome of an encounter. By summarizing these outcomes into qualitative metrics which is published on the blockchain, we can

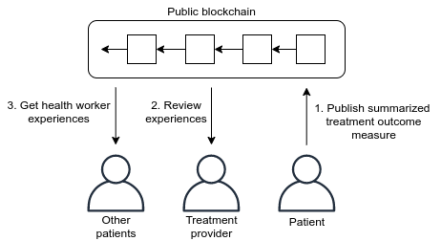


Figure 3: A model for generating evidences for the competence of health workers

measure the quality of a treatment. This general process is shown in Figure 3, where the patient interacts directly with the blockchain to publish a summarized outcome measure related to a treatment they have gone through. Since these treatments are linked to a healthcare worker, we can use them as a proxy for evaluating their competence. As a log of treatment metadata with corresponding outcome measures is built on the blockchain, it serves as evidence of competence.

5.1.4 Access control in smart contracts.

The implemented access control scheme can be described as a Role Based Access Control (RBAC) scheme where accounts interacting with the blockchain must hold a certain role within our distributed application to perform such an action. Examples of access control policies in the blockchain component of our architecture include: **1.** Only existing authorities may interact with the distributed governance protocol; **2.** Only existing authorities may add trust in a treatment provider, license provider or license issuer; **3.** Only treatment providers trusted by an authority may add treatments; and **4.** Evaluations can only be created by the patient who is the subject in a treatment.

6 IMPLEMENTATION DETAILS

A fully working proof-of-concept application was developed for assembling metrics, finding faults with the architecture, and for testing stakeholder workflows. During the application development process, we tried to keep usability for administrators in mind, where we tried to make the process of administering as easy to run as possible. This will allow further development of the application to be easy, and allows third parties to easily test and set up the application. All the code for the software application is in a github repository[2].

The full application is created as four independent services. These services together simulate the architectures shown in Figure 1, 2 and 3. Figure 4 shows how these services interact during runtime. The contract-deployer service is a short-lived service responsible for deploying the smart contracts to the blockchain and export the keys used for their deployment.

6.1 Setup with Docker

A `docker-compose.yml` file is provided in the root of the project. This file is designed to automatically build and start all services in the correct order. Running `docker-compose up` in the project root should be sufficient.

[2]https://github.com/jarensaa/transparent-healthcare

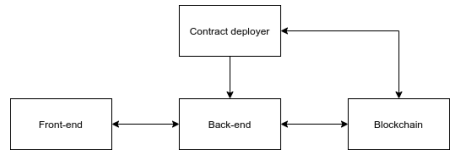


Figure 4: An overview of the run-time presence for our implemented services

6.2 User interface

Once the platform is up and running it offers a user interface with the following sections:

(1) Section for navigating to the panels relevant to the authority stakeholder;
(2) Section for navigating to the panels relevant to the License Provider and license issuer stakeholder;
(3) Section for navigating to the panels relevant to the Treatment Provider stakeholder;
(4) Section for navigating to the panels relevant to the healthcare worker stakeholder;
(5) Section for navigating to the panels relevant to the patient stakeholder;
(6) Section for navigating to the panels relevant to key management, relevant to all stakeholders;
(7) Selection button for selecting the current active Ethereum keypair (ECDSA keypair) to be used for actions in the UI, relevant for all stakeholders;
(8) Toggle for *admin mode*: This gives access to an account which is the first default authority, thus, giving a baseline allowing the user to expand the hierarchy from there. This account has a initial balance of 100ETH, which can be sent to other accounts so they are able to create transactions.

6.3 Key management

The key management is performed via a panel shown in Figure 5. This panel allows users to create and view keypairs. These keys are either stored on the server or locally, it depends on the intent. One can also use the panel to send Ether from one account to another. The panel has the following sections:

(1) The button to click to access the key management panel.
(2) The panel to send Ether from one account to another.
(3) The section to view current keys of all formats. This shows fields such as address and balance. If a local key is shown, the private key will be used. If present, an access token to use the key on the backend is shown.
(4) A card which can be clicked to create new keys with a selection of types.

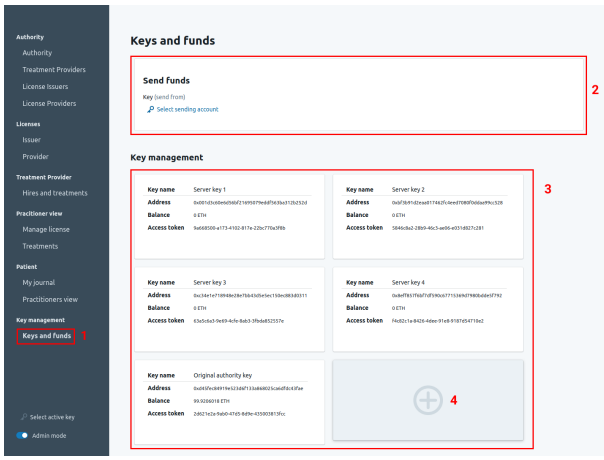


Figure 5: Overview of the key management panel

6.4 Initial results of simulated use of the platform

We have performed numerous simulated runs for different workflows of the platform. The simulations were performed with a single-node Ethereum network configured to simulate the real-world behaviour of the public Ethereum network. Configuration parameters for the blockchain were: Block gas limit: 9.991.391; Block generation time: 20s.

The simulations showed the following costs (in gas denominations) for different smart contract invocations: Add another address as a authority - 179013; Remove an authority - 149040; Vote on a proposal to add or remove a authority - 73686; Enact a proposal to add or remove a authority - 64297; Trying to enact a proposal without a majority vote in place - 28045; Enact proposal to remove an authority - 45332; Add trust in a registered treatment provider - 93707; Remove trust in a registered treatment provider - 22909; Add trust in a registered license issuer - 48863; Remove trust in a registered license issuer - 18829; Add trust in a registered license provider - 48906; Remove trust in a registered license provider - 15965; Register address as a treatment provider - 85959; Create a new treatment - 200118; Register as license issuer - 71059; Issue a new license to address - 88538; Approve movement of license to a new license issuer - 23040; Register as license provider - 86036; Approve movement of license to a new license provider - 38019; Propose movement of license to a new license provider - 46059; Propose license provider movement - 46092; Approve published treatment for a given patient - 102721; Submitting an evaluation - 143669.

Combined with historical data of the price of Ether vs. USD, in Figures 6, 7, 8 and 9, we show the simulated costs for several smart contracts such as creating a treatment or evaluation of a treatment. In Figure 7, we show that the current cost for creating a treatment and getting it approved is around 1 USD, and the cost for evaluating a treatment, as shown in Figure 9, would be around 0.5 USD. However, these prices may increase dramatically if network congestion

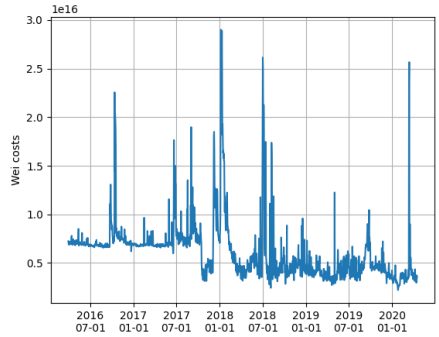


Figure 6: Cost in wei for creating a treatment and getting it approved by a health worker.

reaches similar levels as in January 2018, when a dramatic cost increase was observed.

7 CONCLUSIONS AND FUTURE FORK

We presented the design rationale, modelling and implementation of VerifyMed - a robust blockchain platform for transparent trust in a healthcare domain. To our knowledge, this is one of the first blockchain solutions addressing this specific problem. It is based on the Ethereum blockchain. Our platform is released as an open source code in github. The open source includes also user guides for setting up and running the platform.

We envision three user entities for this trust enhancing platform: governance entities, healthcare workers and patients. For each of these entities, the platform offers easy and intuitive user interfaces.

We have performed numerous simulated use case scenarios with the platform and showed the modest cost of the platform services for an extended simulated period of four years.

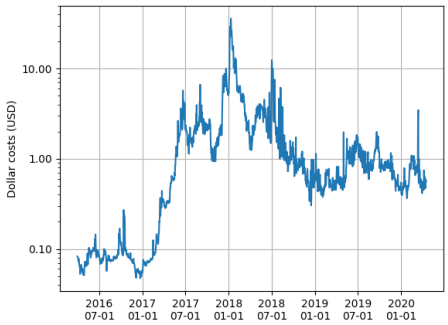


Figure 7: Cost in USD for creating a treatment and getting it approved by a health worker.

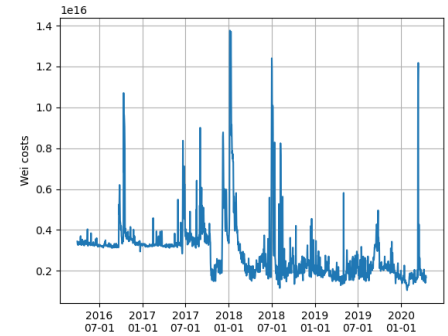


Figure 8: Cost in wei for evaluating a treatment

Figure 9: Cost in USD for evaluating a treatment

Future work: Our platform in the future updates will enrich the current trust model by including more trust requirements such as 1. The caregiver must trust that the patient exists; 2. The caregiver must trust the authenticity of the data that the patient is willing to share and 3. A third party (e.g. a insurance company) must be able to trust the patients claim that care provision has taken place.

REFERENCES

[1] Cornelius C Agbo, Qusay H Mahmoud, and J Mikael Eklund. 2019. Blockchain technology in healthcare: a systematic review. In *Healthcare*, Vol. 7. Multidisciplinary Digital Publishing Institute, 56.

[2] A. Azaria, A. Ekblaw, T. Vieira, and A. Lippman. 2016. MedRec: Using Blockchain for Medical Data Access and Permission Management. In *2016 2nd International Conference on Open and Big Data (OBD)*. 25–30.

[3] Matt Blaze, Joan Feigenbaum, and Jack Lacy. 1996. Decentralized trust management. In *1996 IEEE Symposium on Security and Privacy*. IEEE, 164–173.

[4] Angelo Capossele, Andrea Gaglione, Michele Nati, Mauro Conti, Riccardo Lazzeretti, and Paolo Missier. 2018. Leveraging blockchain to enable smart-health applications. In *2018 IEEE 4th International Forum on Research and Technology for Society and Industry (RTSI)*. IEEE, 1–6.

[5] Gaby G. Dagher, Jordan Mohler, Matea Milojkovic, and Praneeth Babu Marella. 2018. Ancile: Privacy-preserving framework for access control and interoperability of electronic health records using blockchain technology. *Sustainable Cities and Society* 39 (2018), 283 – 297. https://doi.org/10.1016/j.scs.2018.02.014

[6] Monika Di Angelo and Gernot Salzer. [n.d.]. Characterizing Types of Smart Contracts in the Ethereum Landscape. In *Proc. 4th Workshop on Trusted Smart Contracts, Financial Cryptography 20*. Springer.

[7] Morris J Dworkin. 2015. *SHA-3 standard: Permutation-based hash and extendable-output functions*. Technical Report.

[8] Eric Funk, Jeff Riddell, Felix Ankel, and Daniel Cabrera. 2018. Blockchain technology: A data framework to improve validity, trust, and accountability of information exchange in health professions education. *Academic Medicine* 93, 12 (2018), 1791–1794.

[9] William J Gordon and Christian Catalini. 2018. Blockchain technology for healthcare: facilitating the transition to patient-driven interoperability. *Computational and structural biotechnology journal* 16 (2018), 224–230.

[10] Anton Hasselgren, Katina Kralevska, Danilo Gligoroski, Sindre A. Pedersen, and Arild Faxvaag. 2020. Blockchain in healthcare and health sciences—A scoping review. *International Journal of Medical Informatics* 134 (2020), 104040.

[11] Don Johnson, Alfred Menezes, and Scott Vanstone. 2001. The elliptic curve digital signature algorithm (ECDSA). *International journal of information security* 1, 1 (2001), 36–63.

[12] Charlotte Kingsley and Sanjiv Patel. 2017. Patient-reported outcome measures and patient-reported experience measures. *Bja Education* 17, 4 (2017), 137–144.

[13] Tim K Mackey, Tsung-Ting Kuo, Basker Gummadi, Kevin A Clauson, George Church, Dennis Grishin, Kamal Obbad, Robert Barkovich, and Maria Palombini. 2019. 'Fit-for-purpose?'–challenges and opportunities for applications of blockchain technology in the future of healthcare. *BMC medicine* 17, 1 (2019), 68.

[14] Thomas McGhin, Kim-Kwang Raymond Choo, Charles Zhechao Liu, and Debiao He. 2019. Blockchain in healthcare applications: Research challenges and opportunities. *Journal of Network and Computer Applications* 135 (2019), 62 – 75.

[15] Peter B Nichol and Jeff Brandt. 2016. Co-creation of trust for healthcare: The cryptocitizen framework for interoperability with blockchain. *Research Proposal. ResearchGate* (2016).

[16] M. Raikwar, D. Gligoroski, and K. Kralevska. 2019. SoK of Used Cryptography in Blockchain. *IEEE Access* 7 (2019), 148550–148575.

[17] M. Raikwar, D. Gligoroski, and G. Velinov. 2020. Trends in Development of Databases and Blockchain. arXiv:cs.DC/2003.05687

[18] Graham Scambler and Nicky Britten. 2013. System, lifeworld and doctor–patient interaction: Issues of trust in a changing world. In *Habermas, critical theory and health*. Routledge, 53–75.

[19] Bruce Schneier. 2007. *Applied cryptography: protocols, algorithms, and source code in C*. John Wiley & sons.

[20] Christoph Thuemmler and Chunxue Bai. 2017. *Health 4.0: Application of Industry 4.0 Design Principles in Future Asthma Management*. Springer International Publishing, Cham, 23–37. https://doi.org/10.1007/978-3-319-47617-9_2

[21] Theresa Weldring and Sheree MS Smith. 2013. Article Commentary: Patient-Reported Outcomes (PROs) and Patient-Reported Outcome Measures (PROMs). *Health services insights* 6 (2013), HSI–S11093.

[22] Gavin Wood. 2014. Ethereum yellow paper. *Internet: https://github.com/ethereum/yellowpaper, [version 7e819ec - 2019-10-20]* (2014).

[23] P. Zhang, M. A. Walker, J. White, D. C. Schmidt, and G. Lenz. 2017. Metrics for assessing blockchain-based healthcare decentralized apps. In *2017 IEEE 19th International Conference on e-Health Networking, Applications and Services (Healthcom)*. 1–4.

Jens-Andreas Hanssen Rensaa

VerifyMed - Application of blockchain technology to improve trust in virtualized healthcare services

NTNU
Kunnskap for en bedre verden