UPTEC F 17023
Examensarbete 30 hp
Juni 2017

Blockchain Technology and Smart Contracts

Privacy-preserving Tools

Jonatan H. Bergquist

Abstract

Blockchain Technology and Smart Contracts: Privacy-preserving Tools

Jonatan H. Bergquist

The purpose of this Master's thesis is to explore blockchain technology and smart contracts as a way of building privacy-sensitive applications. The main focus is on a medication plan containing prescriptions, built on a blockchain system of smart contracts. This is an example use case, but the results can be transferred to other ones where sensitive data is being shared and a proof of validity or authentication is needed. First the problem is presented, why medication plans are in need of digitalisation and why blockchain technology is a fitting technology for implementing such an application. Then blockchain technology is explained, since it is a very new and relatively unfamiliar IT construct. Thereafter, a design is proposed for solving the problem. A system of smart contracts was built to prove how such an application can be built, and suggested guidelines for how a blockchain system should be designed to fulfil the requirements that were defined. Finally, a discussion is held regarding the applicability of different blockchain designs to the problem of privacy-handling applications.

Handledare: Prof. Dr. Krcmar, Dr. M. Dapp, Dr. M. Buschle
Ämnesgranskare: Dr. Tjark Weber
Examinator: Dr. Tomas Nyberg
ISSN: 1401-5757, UPTEC F 17023

Acknowledgments

I would like to thank my supervisors Dr. Marcus Dapp of the Technical University of Munich and Dr. Markus Buschle of zeb.rolfes.schierenbeck.associates GmbH. Without their guidance this thesis would not have been possible. I would also like to thank my subject reader Dr. Tjark Weber of Uppsala University for providing valuable thoughts on research methodology, direction and for proof-reading. Finally I want to thank my parents, Prof. Dr. J. Bergquist and Dr. C. Bergquist for always encouraging my curiosity and for being my role models in research and most other topics of life.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Currently when someone is sick in Germany and most parts of the western world, he or she goes to see a doctor at a private or public health institution. The doctor then examines the patient and, if needed, prescribes medication, which the patient can buy at a pharmacy. The list of medication is written on a special piece of paper that only doctors are allowed to buy and thereafter stamped with a special stamp, proving that a doctor has prescribed the medications. The patient then goes to the pharmacy bringing the prescriptions with the pharmacist deciphers the handwriting and validates the stamp. The safety of the patient is reliant on the doctor asking and receiving the correct answer to, among other questions, what other medications the patient is taking. The patient also mustn't lose the prescriptions list or he or she will have to return to the doctor once again. The pharmacist has to be able to validate the stamp in a secure manner. The entire system is trusted to be safe against fraud because it is relatively difficult to forge a prescription. Trust is also put in the doctor's ability to correctly perform the anamnesis, the asking of questions at the beginning of a medical visit including questions about other drugs. When a patient becomes sick or suffers as a consequence of medical treatment, it is called *iatrogenic illness*. This can be injury upon examination, negligence, medical treatment errors and unintended effects of medication. Medication errors or *adverse drug events* are the most common cause of injury to hospitalised patients and are often preventable. (Bobb, Gleason, & Husch, 2004). Of adverse drug events, prescribing errors are the most common form of avoidable medication errors. (Hamid et al., 2016) In a study of 17808 prescriptions ordered during one week at a hospital, 1111 of those were shown

to contain a prescribing error. Of those 1111, about 2% were classified as "likely to have caused patient harm" (128 prescriptions) or "likely to cause need for monitoring" (214 prescriptions). (Bobb et al., 2004) There is however, a modernisation happening, in USA by April 2014, 70% of physicians were using electronic prescription methods and 90% of pharmacies were enabled to accept such prescriptions. Notably, the usage of so-called e-prescriptions has increased by at least 50 percentage points in 48 states from December 2008 until April 2014, (Gabriel MH, 2014). The change is also happening in Germany with the introduction of the E-Health Act in January 2015. Among other things that the act encompasses, is an electronic medication plan to be put to use by 2018 for all patients having three or more prescriptions. This is meant to lower harmful interactions between medications by informing doctors of what medications a patient is taking. However, in a study from July 2015 that aimed to evaluate the accuracy of medication plans showed that only 6.5% of all medication plans evaluated did not contain discrepancies. (Waltering, Schwalbe, & Hempel, 2015) This calls for an increased unification of existing analog and digital systems and for te development of a less error-prone model.
As the digitisation of public and private organisations evolves, customers and citizens are exposed to new kinds of vulnerabilities. Instead of passport theft or bank robbery, we are now worrying about hackers stealing identities or personal information, ("JPMorgan Chase Hacking Affects 76 Million Households," 2014). In such systems there is a large need of immutability, identification and redundancy; e.g. no one should be able to alter the prescriptions of a patient except for a doctor, there should be no doubt about the identification of all participants in the system, and there should not be a single point of failure. A traditional database system does only partly fulfil these requirements and alternative technologies should therefore be explored. In 2008 the basis for an immutable, cryptographically secured and distributed database system was laid with the introduction of Bitcoin and blockchain technology. (Nakamoto, 2008) Since the implementation of Bitcoin in 2009, other applications of blockchain technology have emerged, mainly in the financial sector but also with non-cryptocurrency related use cases. Blockchain as a stand-alone technology, along with recent advances in computer science regarding secure multi-party computation, was proposed by Zyskind et al. as a method for access-control and the removal of trusted third parties when dealing with personal data. (Zyskind,

Nathan, & Pentland, 2015) There has already been some exploration into the subject of using blockchain technology for digitalising existing processes in the health care industry. (Krawiec et al., 2016) Many high-level advantages to using blockchain for electronic patient records (EPRs), without any deeper technical analysis, are put forth in (Braxendale, 2016). Other applications than EPRs are explored in (Irving & Holden, 2016) and in (Nugent, Upton, & Cimpoesu, 2016). It seems however that the current solutions available are either inefficient because of how the consensus mechanism is used in the blockchains, or that they are not secure and rely on the trust of a third part (e.g. government, company etc.). In (van Dijk & Juels, 2010) a very strong argument is made that privacy-preserving cloud-computing can never be done using cryptography alone, but one must rely on "tamperproof hardware, distributed computing, and complex trust ecosystems."

1.2 Purpose and research questions

The goal of this Master thesis is to show how blockchain technology and smart contracts can be used to securely share and control personal information among parties who do not necessarily trust each other. This will be proven by a proof-of-concept application for the use case of electronic medical records (EMR). The results of the investigation will have clear applicability to many use cases. The research goal can be broken down into three research questions:

- **Research question 1: What are the requirements for storage of prescriptions, patient-, doctor- and pharmacy-profiles on a blockchain application for prescriptions?** To answer this question a literature-review will be performed, also a study of the existing frameworks and technologies for data-storage (on- and off-blockchain) will be executed. The latter part will be done by comparing different technologies with the requirements of the application.
- **Research question 2: How can the architecture of a blockchain application for privacy-preserving data-sharing between known, but not necessarily trusted, parties look like?** This research question is focused on the architecture of the blockchain application as a whole and attempts to evaluate it from a security

perspective. The question will result in a collection of the most privacy-critical parts of the application. If any parts are lacking, those parts will be rebuilt and evaluated again and if the requirements identified in research question 1 were enough to pass security tests, then no further development will be required.

- **Research question 3: How can a blockchain application for prescriptions handling be built in order to ensure that each patient has access control over prescriptions, that only certified doctors can prescribe medications and that pharmacies who sell medications perform controls over prescriptions?** This research question is focused on the *access control* part of the blockchain application. To answer this question a literature-review will be performed. Existing blockchain-based applications will be inspected and evaluated for requirements. The results of the investigation will be a table or a list of technical requirements with building blocks providing these requirements as well as, potentially, a schematic over the architecture connecting them. An argumentation will be made to motivate why the requirements are needed and in fact the most appropriate for the purpose of this thesis. Finally, the requirements will be implemented in the artefact. The results will be evaluated according to (Hevner, March, Park, & Ram, 2004)

1.3 Limitations

This PoC will strictly consist of the code necessary for the smart contracts, which define most of the operational logic and basic permissions management. Considering the time constraints of this thesis (approximately six months) and abundance of existing blockchains, no blockchain will be programmed. However, in Section 3.2.4 and in 2.2.3, there is a discussion of blockchains design considerations for the extension of the PoC. The thesis discusses cryptography used in blockchains and some additional encryption mechanisms are suggested for the PoC. These are however relying on existing technologies and implementations and are not part of the smart contracts code. One could also argue that other parties involved in the economics and regulation of health care such as insurance companies, the Ministry of public health or the medical products agency should be included. Although these types of users are not implemented with their specific requirements, in the PoC, they are considered and discussed in Section 5.2. Another highly relevant subject, important for the application of block-chain technology to handling of personal data such as in the medication plan, are legal considerations. Since this is a technical thesis, most legal requirements are not discussed but the ambition is that data privacy laws shall be honoured.

1.4 Related work

Since the start of this thesis (August 2016), much related work has been done, advances in blockchain technology and large open source efforts in development have been made. (Zyskind et al., 2015), presents ENIGMA, a blockchain-based solution for secure multi-party computations. They suggest using blockchains for permissions management and for storing pointers to encrypted data, while the actual data is hosted by a trusted, blind escrow service. (Kosba, Miller, Shi, Wen, & Papamanthou, 2015) lay the groundwork for a project called HAWK, a framework and compiler for writing privacy-preserving smart contracts. (Kish & Topol, 2015) Propose in *Nature Biotechnology*, the use of blockchain technology for managing patient data but do not discuss a specific implementation or technical discussion. (Azaria, Ekblaw, Vieira, & Lippman, 2016) design a modular system for storing electronic medical records on a blockchain, they suggest a Proof-of-Work system for incentivising the participation of doctors and hospitals in the system. Med-Vault ("Medical Records Project Wins Top Prize at Blockchain Hackathon," 2015) were present in the media but have not published any details regarding their blockchain-EMR.

To the best knowledge of the author, there have been no functional, electronic medical prescriptions based on blockchain technology built so far.

Chapter 2

Technical Background

In this section, we will give a brief introduction to, and explanation of some basic concepts in cryptography, blockchain technology and related concepts such as smart contracts. Thereafter we will give a description of existing and currently maintained, related blockchain platforms.

2.1 Cryptography

Cryptography provides techniques for transformation of data in order to render it useless for unintended receivers of the data. Useless, in this context, means the thwarting of two basic actions; extracting information from the data and injecting false data or altering the data. This is called the confidentiality- and the integrity-problem respectively. Additionally, one could imagine the case where a sender encrypts and sends a message only to later deny having sent it. Not being able to plausibly deny having sent specific data is another cryptographic goal, called non-repudiation. At its core, cryptography is the theory, but also to a large extent the practice, of preventing and detecting cheating or disallowed access to and usage of data. Where nothing else is stated, this section will be based on the book Handbook of Applied Cryptography by A. Menezes, P. van Oorschot and S. Vanstone. (Alfred J. Menezes, 1996)

Data encryption can be classified into three branches; unkeyed, symmetric-key and asymmetric-key, as shown in figure 1. Unkeyed primitives are functions that do not use a key to encrypt a message, e.g. arbitrary length hashing and permutations. Symmetric-key primitives use the same key for encryption and decryption whereas asymmetric-key cryptography uses the system of a public key and a private key (not equal to each other) which are both required for encryption and decryption.

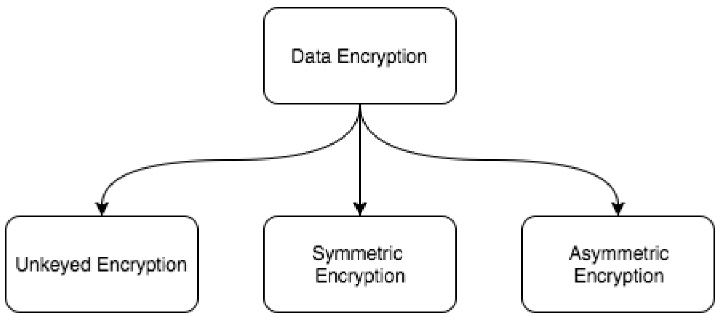

Figure 2.1: Division of types of cryptographic tools. Reproduction of Figure 1.1 in (Alfred J. Menezes, 1996)

2.1.1 Basic terminology

Denote the set $\mathcal{M}$ as the message space, consisting of strings of symbols from the alphabet of definition. The alphabet could be the binary, {0,1}, the English, or the hexadecimal, {0,1,...,9,A,B,...,F}. An element, $m \in \mathcal{M}$ is called a plaintext message, since it supposedly is written in plain text, visible to everyone. Additionally, denote the cipher text space as the set $\mathcal{C}$, consisting of elements of an alphabet of definition potentially different to the one used in $\mathcal{M}$. An encryption function is a bijection between $\mathcal{M}$ and $\mathcal{C}$, uniquely determined by an element in the key space $\{e \in \mathcal{K}\}$. The bijection is denoted E_e. Similarly, a decryption scheme can be defined as the bijection between $\mathcal{C}$and $\mathcal{M}$, determined by the decryption key $d \in \mathcal{K}$, the decryption function is denoted $\mathcal{D}_d$. To fully define the encryption/decryption operations, we need the encryption set $\{E_e : e \in K\}$ and the decryption set $\{D_d : d \ \in \ K\}$ such that: $\forall\ e \in K\ \exists\ d \in K$, where d is unique and $D_d = E_e^{-1}$. This allows for the decryption of a message to be written as: $D_d(E_e(m)) = m\ \forall\ m \in M$. Note that the key-pair $\{e, d\}$ can consist of the same two keys or not. It is assumed that $\mathcal{M}, \mathcal{C}, \mathcal{K}, \{E_e : e \in K\}$ is public knowledge and the only thing which is kept secret between the communicating parties is the key-pair $\{e, d\}$.

2.1.2 Hash functions, private- and public-key cryptography

The unkeyed primitive of most interest to the understanding of blockchains and privacy in the context of this thesis, is hashing. A hash function is a one-way function mapping an arbitrary length binary string into a fixed-length binary string. It is called a one-way function because it takes little computational resources to calculate it, but a very large effort, preferably an impossible amount, to retrieve the inverse of the function. An important characteristic to the hash function is its output length (normally n = 256 or 512). It is important because the longer the hash, the more possibilities there are for outputs. And a critical characteristic of a hash function is that there are no or few collisions, when both x and y produce the same output $h(x) = h(y)$. Finally, the hash function must be deterministic, that is, the same input produces the same output every time. The hash function which is considered the safest, i.e. most difficult to reverse or to in any way alter the contents of, is currently SHA-3. It fulfils all the criteria of an ideal hash function, and furthermore it is not susceptible to length-extension attacks. It was demonstrated in late February 2017 that collisions for SHA-3s predecessor SHA-1 were found, where a PDF could be altered to be hashed into a specific value. It took approximately $2^{63.1}$ computations, 6,500 years of CPU computation time and 110 years GPU computation time. The attack proof was done using a so-called *identical-prefix collision attack*, which reduced the computation time about 100,000 times compared to a brute-force attack.(Stevens, Bursztein, Karpman, Albertini, & Markov, 2017) Hashes are used in most password verification systems. There a hash of the password-input is compared to a hash stored in a database, instead of storing the passwords in plaintext. Hash functions are also a necessary basis for understanding proof-of-work, which will be explained in section /refconsensus .

To understand symmetric-key cryptography we will start by considering two parties, Alice and Bob, who wish to exchange messages over an insecure channel (public chat, megaphone etc.). It is not enough for them to hash their messages and send them, since it is computationally close to impossible to find m given $h(m)$. So first they agree upon an encryption scheme, i.e. a message space $\mathcal{M}$, key space $\mathcal{K}$, a set of encryption and decryption functions and ciphertext space $\mathcal{C}$. In the case of symmetric, or private-key cryptography, they agree upon a common key

e such that $\mathcal{D}_e(\mathcal{E}_e(m)) = m$. Then Alice encrypts her plaintext message m into $c = \mathcal{E}_e(m)$ upon which she sends it to Bob. Bob then decrypts it, $\mathcal{D}_e(c) = m$. The first apparent issue here is the initial agreement of the (shared) private key. Since it is commonly assumed that the encryption scheme is public knowledge, or at least can be found out, considering there is a finite amount of possibilities of encryption transformations available. This requires Alice and Bob to communicate over a secure channel (meeting in person for example), which is often unfeasible. This problem is called the key distribution problem and its solution in 1976 is considered to have altered the direction of modern cryptography.

The basis of asymmetric-key cryptography is the solution of the key distribution problem. At first, the idea of sharing a key through an insecure channel was proposed by Ralph C. Merkle (Merkle, 1978). However, the key-sharing algorithm is named after Whitfield Diffie and Martin E. Hellman and their introductory paper. (Whitfield Diffie, 1976) The idea of the so-called Diffie-Hellman exchange, or D-H ex-change, is according to the following protocol where two participants, Alice and Bob wish to share a secret:

1. Alice and Bob agree upon an encryption transformation and
2. Alice and Bob each generate two keys: pk_A, sk_A and pk_B, sk_B such that:
$$\mathcal{D}_{sk}(\mathcal{E}_{pk}(x)) = x$$
3. Both pk_A and pk_B are made public while sk_A and sk_B are secret.
4. Alice encrypts m using Bobs public key and an agreed upon encryption algorithm.
$$c = E_{pk_B}(m)$$
5. Alice then sends c to Bob over any channel.
6. Bob decrypts $D_{sk_B}(c) = m$ using his private key.

An actual implementation came the year after, in 1978, by Rivest, Shamir and Adleman (Rivest, Shamir, & Adleman, 1978), proposing the famous RSA-algorithm.

2.1.3 Digital signatures

To achieve authentication and non-repudiation using cryptography, digital signatures are used. In other words, to assure that a specific person/device has sent a specific message, it needs to be digitally signed, just like letters would be emprinted with a special seal and signed by hand of the sender in former times. A digital signature is a method for digitally signing data with, perhaps even more, certainty of identity than a handwritten letter. Formally, this authenticates the message sent, ensures that the sender cannot deny having sent it and also ensures senders identity.

There are essentially two types of digital signature algorithms, those that require the original message as input for the verification algorithm, and those that do not. In the latter case, the original message is recovered from the signature itself. *Digital signature schemes with appendix* rely on hashing algorithms and are more widely used than the alternative message recovery-type since they are less prone to existential forgery attacks.[1] *Digital signatures with message recovery* does not require a priori knowledge of the original message, for verification. It is especially suited for sending short messages since the message can be recovered from the signature itself.

Essential for understanding blockchain technology is the concept of Merkle trees, named after Ralph C. Merkle who owns the patent for it since 1979. (Described in Merkle, 1988) It is a data structure defined as: every non-leaf node carries the hash of the values of its child nodes, and a leaf node does not have any child nodes. To verify that a leaf node is a child of a given node, is a logarithmic-time computational problem, compared to a list, whose size is proportional to the number of nodes in the tree.

2.2 Blockchain Technology

There exists, to the knowledge of the author no single, formal definition of blockchain technology which is generally accepted. Many use Bitcoin as a starting point, explaining blockchain technology by its first application, cryptocurrency. However, there are

[1]An existential forgery attacks is when a message/signature-pair is created although the creator isn't the legitimate signer

systems that aren't captured very well by that definition, and that still are generally classified as blockchains. As for alternative definitions there is the one by Vitalik Buterin, the founder of Ethereum: *a blockchain is a magic computer that anyone can upload programs to and leave the programs to self-execute, where the current and all previous states of every program are always publicly visible, and which carries a very strong cryptoeconomically secured guarantee that programs running on the chain will continue to execute in exactly the way that the blockchain protocol specifies.*(Buterin, 2015) The definition of blockchain made by Buterin is not very rigorous or technical, and is certainly not identical to Bitcoin, but manages to include many characteristics of blockchain systems. An attempt to classify different blockchain technologies was made by (Okada, Yamasaki, & Bracamonte, 2017), but it fails to provide a satisfactory definition. A technical committee has been formed within ISO to define areas for standardisation. ("ISO/TC 307, Blockchain and electronic distributed ledger technologies," 2016) They have yet to publish a formal definition but do describe blockchain as: *a shared, immutable ledger that can record transactions across different industries, [...] It is a digital platform that records and verifies transactions in a transparent and secure way, removing the need for middlemen and increasing trust through its highly transparent nature.* IBM proposes a similar definition saying that a blockchain is *a shared, immutable ledger for recording the history of transactions.* ("How does blockchain work?" 2016) The following definitions are from the sources just mentioned or from ("Ethereum Homestead Documentation - What is Ethereum," 2016). A blockchain is a distributed computing architecture where a computer is called a node if they are participating in the blockchain network. Every node has full knowledge of all the transactions that have occurred, information is shared. Transactions are grouped into blocks that are successively added to the distributed database. Only one block at a time can be added, and for a new block to be added it has to contain a mathematical proof that verifies that it follows in sequence from the previous block. The blocks are thus connected to each other in a chronological order.

The above definition is a very wide one, encompassing almost all existing implementations of blockchains. Therefore a more detailed explanation for the readers understanding will be given below. First Bitcoin will be explained in section 2.2.1, and then the blockchain technologies that were created after that, in section 2.2.2).

2.2.1 Bitcoin - the first blockchain

Blockchain technology stems from the seminal white paper, (Nakamoto, 2008), outlining how the cryptocurrency Bitcoin could be constructed. Bitcoin solved a very important problem in the field of electronic money called double-spending, i.e. using the same electronic coin to pay for multiple things. Normally this is solved through a central authority, such as a bank or another trusted third party but Nakamoto proposed a time-stamp server, which ensures all transactions are appearing chronologically in the database. The author(s)[2] proposes a use of a Proof-of-Work (PoW, see section 2.2.5) algorithm for establishing consensus on which chain is the correct one. It establishes an incentive for users to be correct in the validation of transactions. Essentially, it makes it more expensive to fake a transaction than the potential gain. Without an appropriate algorithm for establishing consensus on the blockchain, there could be no trust in the blockchain-system of Bitcoin since anyone with access to the history of transactions (all nodes), could re-write history and publish it as the true one. In Bitcoin, users don't have accounts or account balances, but instead signs transactions using their private key. Each bitcoin is linked to a public key through an unspent transaction output (UTXO) and the user who possesses the corresponding private key is the owner and can control the usage of it. The UTXO is there because, in Bitcoin, coins sent to an address all have to be spent, even if the user actually doesn't want to spend the entire amount. It is however, possible to split a transaction. Assume that a certain address contains 3 XBT (abbreviation for the Bitcoin currency), and the owner of the private key to that address, i.e. the owner of the bitcoins, wants to pay 1 XBT to another address. The new 1 XBT transaction will use the entire 3 XBT as input and the 3 XBT are thereby spent. So, the change in this transaction, the 2 XBT, will be sent back to the same user but as a new input using a new address. A user who has taken part in payments, whether as payer or payee, will have a collection of addresses, all summing up to the total balance of her wallet. Because the first blockchain to be used was Bitcoin, there was at first no distinction between Bitcoin and blockchain. All initial applications for blockchain were within cryptocurrencies or financial processes. Many blockchain-based use cases are still within the financial

[2]It is to this date still unknown who is the person or persons behind the pseudonym Satoshi Nakamoto, even though multiple claims have been made by researchers, cryptocurrency protagonists and others.

sector, but the benefits of disintermediation of trust have proven to be useful in other areas as well. This was brought forth by the advent of Ethereum, a Bitcoin-like cryptocurrency but with added functionality for smart contracts.

2.2.2 Post-Bitcoin blockchains

Some properties of Bitcoin have been abstracted and rebuilt into what is now called blockchain technology or distributed ledger technology. While still maintaining the main properties of Bitcoin, new blockchains are often more flexible in their applications and what actions they allow. It is a technology very much under development where new approaches and applications are being published frequently, most often through white papers published by start-ups or a group of corporate researchers. Still the basics of blockchain remain the same, it is a distributed, time-stamped database with consensus-establishing peers. Blockchain technology is characterised by the following traits:

- Distributed: Nodes are considered equal in the sense that they all have a full copy of the entire history of the database. There can also be less equal nodes, also called lightweight nodes, which only have a couple of the last blocks stored locally. Generally, communication between nodes is done over the Internet with private-key cryptography.
- Time-stamped: Since every block of transactions is hashed into all the subsequent blocks, it becomes increasingly difficult to change history the further away in time the current block is. The blockchain at hand becomes a provably correct auditing tool.
- Consensus: Nodes establish one truth about which version of the database is the correct one through a consensus-algorithm. This serves to validate transactions as well as to discourage for example double-spending attacks. The type of consensus-algorithm being used is highly dependent on the structure and purpose of the blockchain. See section 2.2.5.

2.2.3 Permissions and specialisation

As the development of blockchain technology progressed past Bitcoin, two different options developed as to who should be allowed to participate in the validation and observing of the network. The dichotomy is essentially between permissioned and permissionless blockchains, although there is in some cases some flexibility for hybrid solutions to be implemented. A blockchain which exists openly on the internet is called *permissionless*, classic examples of such are Bitcoin and Ethereum. This type of structure is what was defined in the Section 2.2. However, the more actions that are allowed, the more possibilities to hack the blockchain there are. This was seen during the infamous DAO-hack where approximately USD50 million were siphoned from an ether fund. (Buterin, 2016). Also, since the data on the blockchain is open to anyone who wishes to join the network, data has to be kept completely anonymised (as not completely successfully attempted by Bitcoin,Vasek and Moore, 2015) if it's necessary to keep it private. Since, in some cases, it is not possible to anonymise all the data or it is simply not desirable that everyone can participate in a network, *permissioned* blockchains developed.

The principle of permissioned blockchains is that there is a regulation of who is allowed to join and participate in the network. This can be done by a consortium of companies, governmental agencies or other organisations, either by inviting new members one be one, or by predefining a set of criteria. The benefits, besides the increase in privacy, include the potential for more flexibility in adapting the network, better scalability and faster transactions. Sometimes, depending on the consensus algorithm at play, permissioned blockchains can be more susceptible to unintended changes of its history. In other words, the speed, privacy and scalability are sometimes being traded for immutability and censorship-resistance. (Mattila, 2016). This is because a permissioned blockchain doesn't necessarily require a PoW-consensus algorithm, but can use one with less resource expenditure, thus making the process of concurrency easier.

Blockchains can also be created more or less flexible, or specific, in what actions are permitted on them. For example, Bitcoin and most *coins*, is an example of highly specialised chains with one purpose - to safely transmit the tokens of the

Table 2.1: Generalised vs. Specialised blockchains and Permissioned vs. Permissionless. Original source: "Monax - Blockchain explainer," 2016

	Permissionless	Permissioned
General purpose	Ethereum	Monax's eris-db
Specialised	Bitcoin	Multichain

cryptocurrency. On the other hand, there is Ethereum, with a virtual machine built in, as well as the possibility to deploy smart contracts in a turing-complete manner. Ethereum was explicitly created to allow for the creation of decentralised applications (DApps), and has at the time of writing this thesis, about 360 applications listed on http://dapps.ethercasts.com/. Ethereum is, however, permissionless and isn't the right platform for all DApps, necessarily. In table 2.1 the matrix of permissionless/permissioned and generalised/specialised blockchains is shown with examples of a blockchain or platform for each category. A generalised blockchain is one which is not optimised for performing one specific task, in opposition to a specialised one that is. Both Ethereum and Bitcoin are permissionless, but on the permissioned spectrum, there is the multi-purpose eris-db from Monax and the specialised Multichain platform. eris-db is a blockchain client containing a permissions layer, an implementation of the EVM and uses by default Tendermint consensus, although that can be modified. Tendermint is a Proof-of-Validation (PoV) algorithm, where scarce tokens are deposited and are threatened to be deleted if voting is dishonest (see Section 2.2.5). Eris-db is different from MultiChain in that, MultiChain is a fork from the Bitcoin Core source code and is in many ways optimised to work with the Bitcoin network. Multichain is specialised because it is optimised to perform transactions, wheras Monax is built to supply a great number of services. It does not, however, mean that it is not possible to build customised services on Multichain, or high-performance payment systems on Monax, just that it is made easier by design. It doesn't use PoW, but has a type of algorithm where the maximum amount of votes that a miner can cast during a specific timeframe is limited. ("Ethereum Homestead

Documentation - What is Ethereum," 2016)

2.2.4 Smart Contracts and Ethereum

The name smart contracts is arguably a misnomer since they are in fact neither smart nor contracts in the common sense. Smart contracts are, in the context of blockchain, simply logic that is published on a blockchain, can receive or perform transactions like any address (transactions may be rejected or require special arguments to function) and that can act as an immutable agreement. The purpose of the smart contracts is to act as a "computerised transaction protocol that executes terms of a contract" (Szabo, 1994) and was first coined by cryptographer Nick Szabo. The basic idea, and the source of the contract-part in the name, is that certain parts of contracts can be included in software in such a way that the breach of them is either expensive or impossible. Smart contracts are often confused with Ricardian contracts (Griggs, 2015), which is the digital recording and connection to other systems of a contract at law. This is not what is meant by smart contracts, since they do not need to be legal in any way, nor connected to outside systems. One could however, imagine value in the connection of smart contracts with Ricardian ones to "outsource" functionality of legal contracts to smart contracts.

According to Szabo, contracts need to have a couple of characteristics to be defined as truly smart contracts. These characteristics are: visibility, online enforceability, verifiability and privity. Visibility (Szabo uses the term observability) means that participants in the contract should be able to see each other's performance of the terms of the contract, or to be able to prove the fulfilment of their own terms to other participants. It is also referring to the visibility of actions taken by the logic in the contract; a Point-Of-Sale screen showing the amount to be paid to the customer but omitting the fact that data is being saved from the credit card is an example of such a hidden action. Online enforce-ability refers to making certain that the terms of a contract are being fulfilled. The measures that can be taken in order to achieve this can be categorised into proactive and reactive ones. Proactive measures seek to make it technically impossible to breach terms or to allow either party to drop out of the contract should there be a valid breach on another part. Reactive measures deter malicious behaviour through reputation or enforcement, but also by recovering potential assets after breach of contract. Smart contracts also need to be verifiable, or auditable, should there be a conflict. Lastly, smart contracts should be as private as possible, meaning that knowledge and control of data involved in a smart contract should only be available to participants if necessary.

One might notice that the objectives of smart contracts just mentioned; visibility, online enforceability, verifiability and privity, results in two separate directions. Privity is exerting a controlling force over the contracts, wanting to minimise openness to outside parties. Diametrically opposed, there are the other three objectives, visibility, enforceability and verifiability, who require access to contractual data to be handed out to participants or auditors. Therefore an optimum must be found where as little information and control as possible is given to external parties, yet the possibility to verify, observe and enforce is still available. In 1997, before blockchain technology and advances in zero-knowledge proofs as well as secure multi-party computations, Szabo's solution to the optimisation problem was to trust an intermediary, a third party, such as an auditor. (Kosba et al., 2015), (Szabo, 1997), (Zyskind et al., 2015)

The Ethereum platform is a general blockchain, with a virtual machine (Ethereum Virtual Machine, EVM) to run smart contracts. Since the environment exists only on the blockchain in the form of a virtual machine, the smart contracts are completely isolated from network, file-system or other processes on the node machines. A high-level, Turing-complete language was created to write smart contracts with on Ethereum. However, that language, Solidity, has now become standard also for other platforms with smart contract capabilities. Solidity is similar to JavaScript in syntax, but is written in a completely different style. After a contract has been written in Solidity, it is compiled into EVM bytecode and then deployed at a specific Ethereum address. To deploy and interact with smart contracts on Ethereum however, a special JavaScript RPC-library is used alongside a web API.[3] Because smart contracts programming started with Ethereum and Solidity, it is still a discipline under development. The Solidity language has a number of known peculiarities and a list of changes to come, meaning that code being written now may not be fully functional with the next

[3]See http://www.ethdocs.org/en/latest/contracts-and-transactions/accessing-contracts-and-transactions.html

update. There are a couple of programming best practices that are specific to smart contracts development, gathered in the (relatively) short time that Solidity has been in use. There are two main reasons behind the extra considerations of security that should be taken into account for smart contracts development; Solidity contracts are likely to process the ownership of valuable tokens, items or rights to something; the execution of smart contracts occurs on a blockchain, meaning that all participants can observe it and the source code for it. Common security guidelines that have been gathered during the (relatively) short time that Solidity has been used are:

Damage control	If possible, the amount of tokens stored in a smart contract should be limited since, if the source code, the platform or the compiler should contain a bug the tokens may be stuck in the contract.
Modularity	Smart contracts should be kept as tiny and simple as possible. Local variables and length of functions should be limited to keep the contracts as readable as possible. The more modular the contracts are, the easier it is to improve a system of smart contracts.
Checks-Effects	Functions should perform precondition checks at the first step of the algorithm. Then, as a second step, changes to state-variable should be made. Finally interactions with other contracts should occur.

2.2.5 Consensus algorithms

Consensus algorithms are of the highest relevance to blockchain technology since the purpose of Bitcoin was to transfer value in an unregulated, distrusting environment, where a sure way of validating transactions was needed. The goal of the consensus algorithm is to ensure a single history of transactions exists and that that history does not contain invalid or contradictory transactions. For example, that no account is attempting to spend more than the account contains, or to spend the same token twice, so-called double-spending. In Table 2.2, different important consensus algorithms are compared to each other. Below, a brief introduction to a few of them is given, but for more details, the reader is referred to (Back, 1997), (Nakamoto, 2008), (Fischer, 1983), (Tendermint, 2017).

Bitcoin solved the consensus problem by, for each new block announcing a *target*, which the hash of the previous block, the current block and a variable *nonce* has to equal less than. Since the output of the hashing function is evenly distributed, it's impossible to create a block such that it with certainty will be easy to reach the target. Therefore, there is a race between the mining computers in the network to find the right nonce. Once a target is reached, the mining computer broadcasts that block to the network and other participants validate the transactions. If enough validating nodes find the transactions to add up, they agree upon that block being added to the chain. This procedure is called *proof-of-work* (PoW). Since the goal is, not to give too much power to a single person or organisation[4], a limited resource has to be chosen which will be spent upon voting for the validity of a block. In PoW, that resource is computing power.(Cynthia Dwork, 1992). Since computing power is getting cheaper and more available with Moore's Law and cloud computing, the difficulty of the hashing problem is regulated according to the frequency with which the previous problems were solved. A common critique of PoW is however, that the "waste" of computing power also means a large waste of energy. There are miners who only mine in winter, and use the exhaust heat from the mining farm to warm up their house. ("Hotmine Inc." 2016). What this essentially means is that miners are forced to pool resources into what can ultimately be a handful of giant Bitcoin farms, thus having centralised the decentralised network. Additionally, Bitcoin does not have a very high throughput of transactions since the block time stays constant at about 10 minutes and block size as well (about 1 MB). The energy waste and throughput are two reasons why alternatives have emerged. The most relevant for this thesis are Proof-of-Stake (PoS) and Tendermint which are very similar. Neither uses computing power as a scarce resource, but rather the ownership of the inherent tokens of the blockchain. The principle is that owners of tokens put a certain amount of tokens at "stake" by betting on the version of the blockchain that they believe is the correct one. This will increasingly incentivise validators to behave according to the rules depending on how much they possess. Validators in the Tendermint consensus algorithm are nodes who take turns proposing blocks of transactions and then vote on them. If a block fails to get enough votes, the

[4]A Sybil-attack is when an attacker gains control of the network tokens and can redirect them to a specific account

protocol moves to the next validator to propose a block. To successfully commit a block, there are two stages that need to be passed: *pre-commit* and *pre-vote*. A block is committed when more than 2/3 of validators pre-commit for the same block on the same round. As long as no more than 1/3 of validators are byzantine, it is impossible for conflicting blocks to be committed at the same height of the blockchain. Tendermint can be modified to act as a Proof-of-Stake algorithm by assigning different "weights" to the votes of different validators. In PoS, there is an attack, or a problem, called the nothing-at-stake-attack. The core of it is that there is no reason why a validator couldn't bet on all different proposed versions, thus being certain to win. The Ethereum wiki-page explains it as: *an attacker may be able to send a transaction in exchange for some digital good (usually another cryptocurrency), receive the good, then start a fork of the blockchain from one block behind the transaction and send the money to themselves instead, and even with 1% of the total stake the attacker's fork would win because everyone else is mining on both.* ("Ethereum GitHub Wiki - Proof of Stake FAQ," 2017)

Consensus algorithm	Resource being used	Benefits	Drawbacks	Examples
Proof-of-Work	Computing power	Trustless, immutable, highly decentralised	Energy consumption, transaction throughput.	Bitcoin, Litecoin.
Proof-of-Stake (PoS)	Ownership of fixed amount of tokens	Efficient in energy and throughput, scalable	Nothing-at-Stake problem. I.e. voting for different forks at the same time	NXT
Delegated PoS	Ownership of scarce tokens + peer reputation (elections for delegates)	Allegedly more efficient than PoS	Voter apathy in elections can lead to excessive centralisation and reduced robustness	BitShares
Tendermint (Proof-of-Validation) (Tendermint, 2017)	Security deposit of scarce tokens subject to burn if voting dishonestly	Gives the benefits of proof-of-stake without almost any of its draw-backs	Nothing-at-stake problem still persists over long periods of time	Eris-Db ("Monax - Blockchain explainer," 2016)
Proof-of-Authority (PoA)	Selected authorities are randomly selected to validate transactions	Efficient, doesn't require any inherent tokens or economic value	The corruption of authorities is a large possibility, relies on authorities being well-selected and controlling eachother	Parity PoA

Table 2.2: Consensus algorithms for usage in blockchains. Adapted from source: (Mattila, 2016) with addition of Proof-of-Authority

Chapter 3

Implementation

In this chapter, a design for a PoC electronic medication plan (EMP) using smart contracts and blockchain technology is proposed. First, the three different user archetypes are described along with user stories in order to provide functional specifications for the PoC. Further descriptions of the PoC such as quality attributes and how to set the system of smart contracts up with a blockchain are also given. Thereafter a schematic of the prototypical interactions on the blockchain is shown along with the interactions between the smart contracts. The suggested solution to the described problem uses the decentralised, trust-less and immutable properties of blockchain technology as well as permissioning in the smart contracts. To be noted is, however, that no security or privacy liabilities outside of the blockchain have been resolved with this implementation. Some of the larger off-chain issues are mentioned in the discussion.

3.1 User stories and requirements

The implementation outlined in this thesis is limited to three archetypical users: patients, doctors and pharmacies. In order to describe the various functional requirements that users have on the application, user stories were written and are shown in Table 3.1 below. The different user types are thereafter defined more in detail. An effort was made to simplify the user stories and requirements to the bare minimum, while still keeping the PoC at a viable level of usability and security.

As a ...	I want/need to ...	Traceability
Patient	Be able to see what prescriptions I have so that I know what medicine to take.	1.1
	Identify myself in a cryptographically secure manner upon accessing my personal information on the blockchain.	1.2
	Be able to share information on my medication plan with Doctors and Pharmacies on the blockchain.	1.3
Doctor	Be able to see and alter the prescriptions of my patients so that I can correctly treat them and avoid medication errors.	2.1
	Verify the patient-account identity before prescribing any medications or in any way altering the medication plan.	2.2
	Be able to see what prescriptions a Patient has gotten from other Doctors, so that I can control the safety of my patients.	2.3
	Identify myself in a cryptographically secure manner upon accessing Patient information on the blockchain, so that no unauthorised entities can access it.	2.4
Pharmacy	Be able to verify a patient's prescription so that I know that she/he isn't trying to purchase unintended pharmaceuticals.	3.1
	Identify myself in a cryptographically secure manner upon accessing Patient information on the blockchain, so that no unauthorised entities can access it.	3.2

Table 3.1: User stories defining functional requirements and guiding development of EMP PoC.

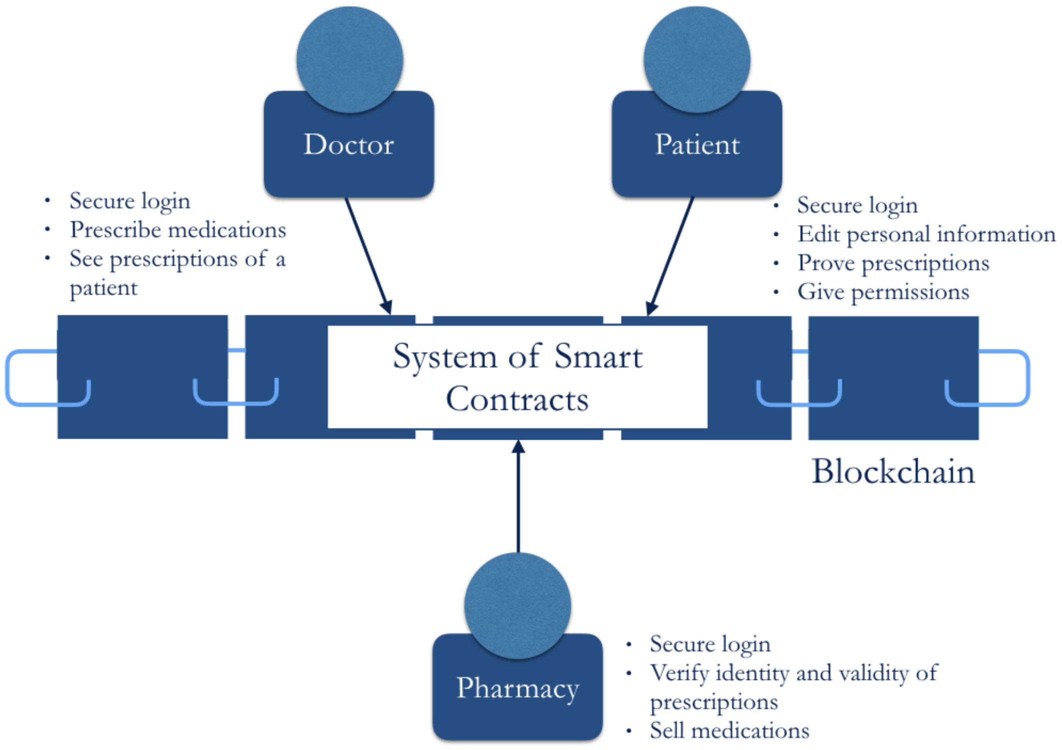

Figure 3.1: Overview of different users and their interactions with the blockchain and system of smart contracts which exist on the blockchain.

Patients are assumed to be private persons, seeking medical care at one of many healthcare providers, in this thesis simply called Doctors. Doctors are assumed to be certified, medical professionals, in possession of a state-issued license and authorisation to practice medicine. Pharmacies are defined as only those commercial or state-owned outlets possessing the legal right to sell prescription medications to patients. The requirements are described in a less formal way in Figure 3.1, where the different users are shown interacting with the blockchain, on which the smart contracts reside. In bullets next to them are the actions they need to be able to perform.
There are some requirements that apply to the general system and not just to one user specifically. Some of them are described in part by the user stories, but for the sake of exhaustiveness and application to users not in the system, they are explicitly written below.

R1. It must be impossible, for a non-admin account, to connect prescriptions to the identity of a patient, doctor or pharmacy without the consent of the user in question.

R2. Only those permitted to should be allowed to connect to the network.

R3. There must be an immutable traceability built into the system, where it is possible to see:

1. Who prescribed a certain medication
2. If a medication was sold after it having been prescribed
3. Where it was sold

Immutable traceability means that there must be a history of changes made to prescriptions and that it must be made very difficult, if not impossible, to alter it post ex.

R4. Smart contracts must be exchangeable without needing to re-move the entire system or change addresses to contracts with which humans interact.

The design of the final PoC was based on the requirements and user stories mentioned above.

3.2 Design of the PoC

In this section, the design, based on the user stories and requirements from the previous section, is described. First of all, the core of the PoC-logic, the smart contracts, is explained. Thereafter, a proposal for deployment on a blockchain is given, along side a clarification of the functionality which that would provide. Names of contracts and functions are written in `this font` from here on to facilitate reading.

3.2.1 Design overview

In Figure 3.2, a high-level overview is given of the EMP as designed in this thesis. It goes from more abstraction at the top ("Software system level"), to lower level of abstraction in the bottom ("Contract level"). For the sake of relevance to the code written for the PoC, only the "Smart contracts"-component is explained in detail in Figure 3.4. The process of starting the application is shown in Figure 3.3 as a UML-like activity diagram. It describes the initial set-up and deployment of the contracts by an admin.

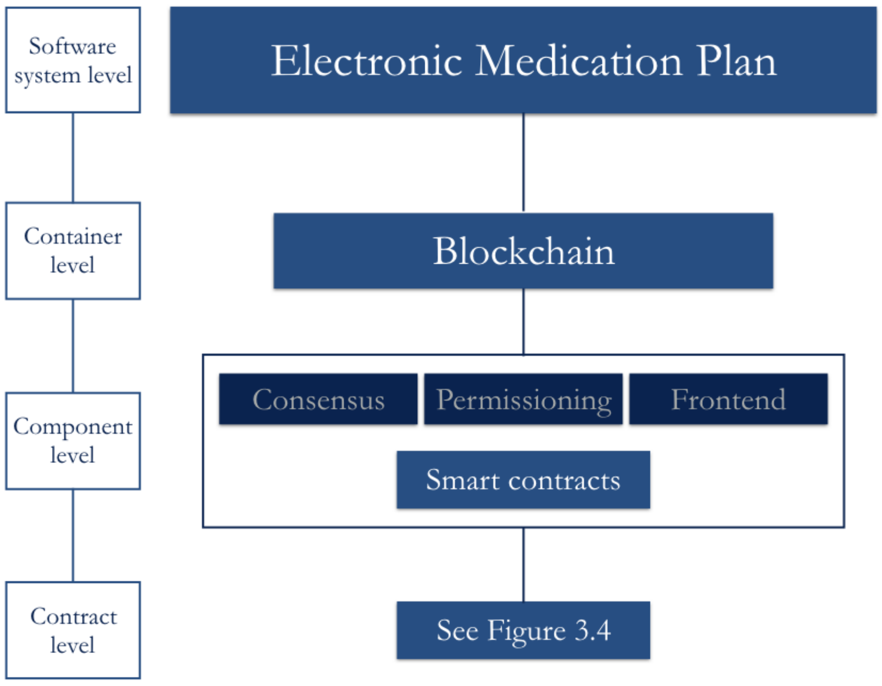

Figure 3.2: High-level system overview of the EMP PoC. The figure is not exhaustive and is simplified for sake of relevancy. Visualisation technique based on (Brown, 2016)

3.2.2 System of smart contracts

The proposed architecture of the system of smart contracts is based on the design principle of having different types of contracts to perform different classes of tasks. To classify the contracts, a model called "The Five Types Model" is used (“Monax - Solidity explainer: The Five-types model,” 2016), although not all of the five models are actually used in the PoC. The model divides contracts into:

1. **Database contracts**	Contracts for storage of data with basic read-, write- and get-functions are called database contracts. They can also include permissions-checking.
2. **Controller contracts**	One step up in the layer of abstraction are contracts for controlling database contracts. For example to perform batch reads/write operations. They can also act on multiple database contracts.
3. **Contract managing contracts**	These contracts are needed to control and manage the actions and existence of other contracts. They should also handle the communication between contracts.
4. **Application logic contracts**	Any contract that is implementing application-specific tasks through controllers is an application logic contract.
5. **Utility contracts**	Some small, generic functions can be outsourced into utility contracts that are highly specialised. For example, a utility contract could hash data or perform another operation.

The contracts in the PoC are the following:

`PatientInfoDb` - A database contract to store information about patients. The information stored for each patient is prescription and the prescribing doctor for each respective prescription. A `Patient`-struct is created for each patient to contain the corresponding data as well as a `next`- and a `prev`-attribute so that a doubly-linked list can be used to iterate over them.

`PermissionsDb` - A database contract to store permissions. The different permissions that can be modified are:

`perms`	0	- Patient permissions, e.g. only allowed to read info related to one's own address

1 - Pharmacy permissions, e.g. allowed to read info about patient who permitted it

2 - Doctor permissions, allowed to add patients, add prescriptions and read info about patients

`patientDoctorConsent` When a patient wishes to grant a specific doctor or pharmacy the right to prescribe, sell medication or add patient as customer a `consentCode` can be added.

`prescripPatientConsent` When a doctor attempts to prescribe a specific medication, the existence of the patient-prescription tuple is checked against this variable.

`Permissions` - Controller contract acting like interface with the PermissionsDb-contract.

`Cmc` - The contract-managing-contract is simply named `Cmc` and contains a collection of the different contracts. All other contracts must be connected to the `Cmc` or inherit from the class CmcEnabled.

`CmcEnabled` - Base class for contracts that are used in a cmc system.

`Patient` - Application logic contract for handling requests from patients such as retrieving prescriptions, changing consent-level for a certain prescription-doctor tuple etc.

`Doctor` - Application logic contract for handling requests from doctors. These include adding a new patient, prescription or confirming that a certain medication has been prescribed.

`Pharmacy` - Application logic contract for handling requests from pharmacies. This is ultimately only to confirm a prescription and will be called when a pharmacist wishes to sell a prescription to a patient.

`InfoManager` - Application logic contract with which users interact. It also checks all permissions and provides one point of contact for a user. See Figure 3.4.

`InfoManagerEnabled` - Base class for contracts that only allow the InfoManager to call them. Note that it inherits from CmcEnabled.

`ContractProvider` - Interface for getting contracts from Cmc.

To implement the application in a real-life scenario, one would need to (even before the steps in Figure 3.3) set up a blockchain such as the one provided by Monax. The setting up consists of each individual node generating a private/public key-pair. The developer then starts her original node, creating a so-called genesis-file which contains necessary information for the blockchain configuration. The developer can the register which private keys should be validators and they can connect to the blockchain. Then the steps from Figure 3.3 continue. The validators are configured, the blockchain is started and transactions can begin. Then the developer deploys the contract managing contract and then all the other contracts. She then registers them with the contract managing contract and sets permissions for the patients, doctors and pharmacies.

3.2.3 Data and variables on the blockchain

Given requirement R1., plaintext data with which a user can be identified cannot be stored on the blockchain. This would allow any participant on the blockchain to see all medications of a specific person. However, if the specificity is reduced, and the person is no longer identifiable, the value in the information address 0x3a5f29... has been prescribed medication A, B, C, is very low. Thus, plaintext prescriptions are assumed to be stored in the smart contracts.

3.2.4 Supporting infrastructure and governance

The PoC is not complete with just the smart contracts. There is also a blockchain needed, as well as a structure for handling keys. Additionally, there needs to be a type of either distributed or centralised consensus established, on who is allowed to join the network as a doctor or a pharmacy.

The suggestion for this PoC, is that a PoS-, Tendermint or PoA-consensus algorithm is used on a permissioned blockchain. The details of the algorithms are explained in

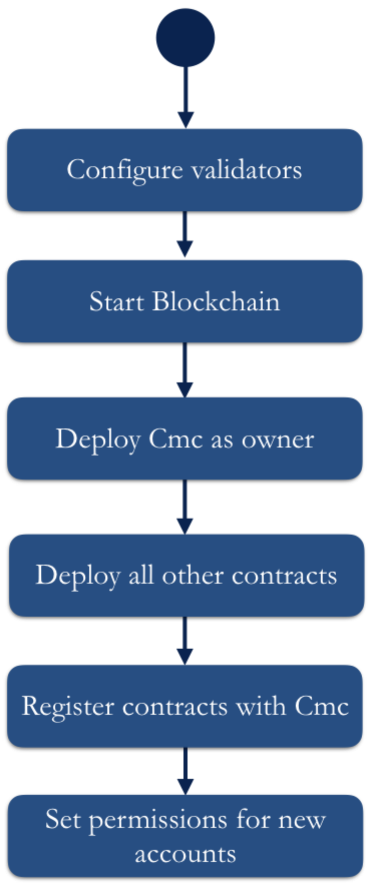
Configure validators
Start Blockchain
Deploy Cmc as owner
Deploy all other contracts
Register contracts with Cmc
Set permissions for new accounts

Figure 3.3: Diagram of how the starting-up activity for the system works.

Section 2.2.5. It is wasteful and inefficient to use a Proof-of-Work consensus algorithm in a permissioned blockchain. Therefore a PoW is not recommended in this PoC. It would however make sense should the infrastructure cost (validating nodes would require constantly running computers running special software) prove to be too high. In such a case, a similar system could be deployed to the Ethereum public blockchain. A weakness in the security in that case would be that a layer of security has been lost, no proof is available that a doctor is a doctor, and since more people can access the contracts, vulnerabilities are more likely to be exploited.
A Proof-of-Stake algorithm is a more efficient alternative should there be some type

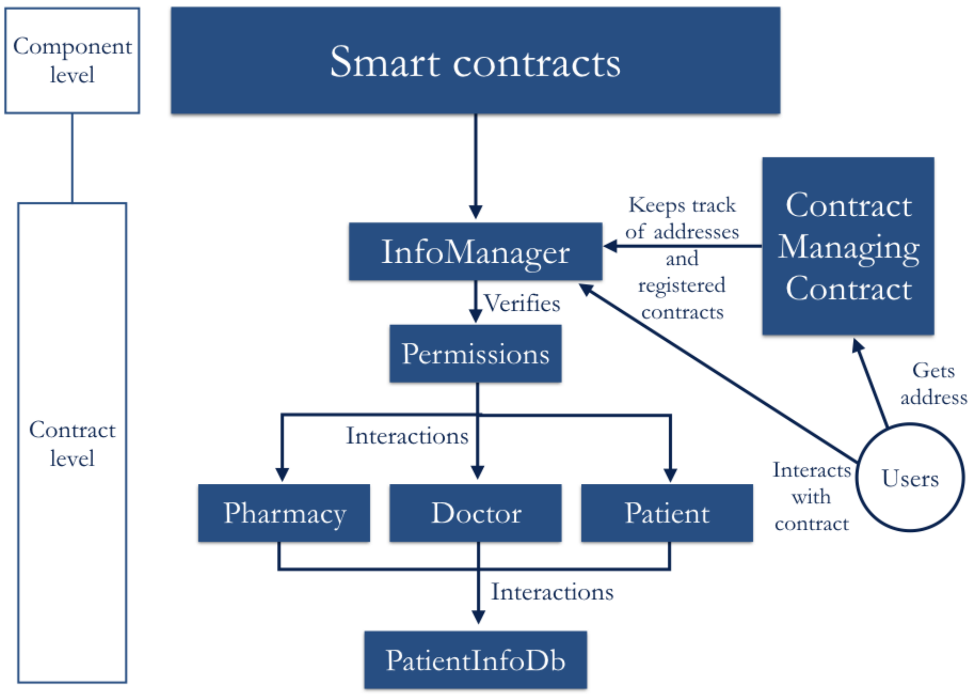

Figure 3.4: Overview of the system of smart contracts for the EMP PoC. Not all smart contracts are included for sake of clarity and relevance. Visualisation technique based on (Brown, 2016).

of value in the inherent tokens. Otherwise no one would voluntarily validate blocks. One can however imagine that some premium from health care insurance companies is payed out based on how much validation a doctor or pharmacy has provided. The premium or payment could also come from a governmental operator. The same logic can be applied to Delegated PoS. A Proof-of-Authority consensus model is feasible, should it be possible to find trustworthy authorities to validate the transactions. If doctors and pharmacies are liable to alter the transaction history, then one or more governing bodies would need to be appointed. These could be as small as independent data-centers locked away in a server room underground with additional security measures like multiple keys required to enter. Or they could be insurance companies operating under EU law, being audited by third parties. This, however, avoids the purpose of the project a bit, to not have a single point of failure or a single trusted authority. Although it does not strictly have a single trusted third party, it does have multiple parties that could collude with each other.
No additional validation besides the permissioning layer on the blockchain and the control mechanisms of the smart contracts is needed, strictly speaking. Although it is advisable that a structure, such as a Decentralised Autonomous Organisation, or another type or organisational structure is set up to validate prescriptions and on-board users. Also, audits of doctors or pharmacies should be formalised and required. In the event of an audit it must be possible to analyse specific prescriptions and actions on the blockchain. The blockchain would consist of, for the largest part, non-validating nodes, patients, and otherwise doctors and pharmacies who could be required to have full nodes. The best fitting, available platform for this is the Eris-db by Monax, who supply a fairly well-developed, though not bug-free, open source blockchain ecosystem. The core is also using the EVM, which means Solidity, and the smart contracts developed for this PoC, can be used without alterations. Eris-Db provides the permissioning layer and integrates with Inter-Planetary-File-System (IPFS), which could be very useful should larger amounts of data wish to be stored, such as entire medical health records or other personal information.

To start an Eris-Db blockchain, a docker-container needs to be set up to ensure that the environment in which the whole package resides is compatible. Once that is set up, keys for the first users need to created, since they need to be included in the genesis json-file. Some type of end-to-end encrypted messaging service should be used when transmitting keys. When designing the genesis file, users can be configured to have different privileges. For the EMP PoC, a number of cloud instances (through Amazon Web Services, Digital Ocean or other) could be set up to act as (Tendermint) validators. This allows patients, doctors and pharmacies to have lightweight client nodes, meaning that they do not need to consecrate devices to always be running to ensure the continuation of the blockchain.

To avoid fraud and enforce Know-Your-Customer-regulations, it would be necessary to have some type of authority to control that users aren't **a)** registering multiple patient accounts to potentially hide prescriptions from doctors and pharmacies, or **b)** registering as a doctor without actually being a licensed doctor, or **c)** registering as a pharmacy without actually being in possession of a pharmacy. The first issue could be solved by having a simple, encrypted storage of who has been registered already, kept by the organisation issuing the accounts. The second and third problem need to be addressed in cooperation with the responsible state department for the country at hand. The record does actually not need to be kept private, since knowing which doctor is registered for an account would not lead to any security issues, as long as there are enough users. It would (hopefully) though be seen as a positive, knowing that a doctor or pharmacy is adopting new technology.

Chapter 4

Evaluation

In this chapter, the artefact presented in Chapter 3 is evaluated using a descriptive evaluation method as proposed in (Hevner et al., 2004). Additionally manual, functional testing was carried out on the system using the online compiler provided by the Ethereum foundation (https://ethereum.github.io/browser-solidity/). Development of the smart contracts was done in Solidity.

4.1 Description of evaluation criteria

An IT artefact can be evaluated according to the criteria: "functionality, completeness, consistency, accuracy, performance, reliability, usability, fit with the organisation, and other relevant quality attributes." (Hevner et al., 2004) However, given the novelty of blockchain technology, the scope of the thesis and the extension of the PoC, the artefact cannot be evaluated according to all criteria. (Hevner et al., 2004) says a descriptive evaluation can be used only in the case where the technology is especially innovative and other methods may not be feasible. The EMP fits those criteria and will therefore be evaluated using two methods from the descriptive evaluation theory.

1. **Informed Argument** Use information from the knowledge base (e.g., relevant research) to build a convincing argument for the artefact's utility.
2. **Scenarios** Construct detailed scenarios around the artefact to demonstrate its utility.

4.2 Fulfilment of evaluation criteria

The EMP PoC fulfils all the functional criteria shown in Table 3.1. A motivation for how each of the user stories is satisfied is exposed in Table 4.1. For details on the code, the reader is directed to the appendix where the full source code can be found. Besides the functional criteria, additional requirements were defined to cover non-functional aspects of the PoC. These are evaluated in an argumentation based on the theory seen in Chapter 2 in the thesis.

4.2.1 Potential security and privacy exploits

The most important non-functional requirement on the PoC, is the security of the patient data. The requirement is R1 in list 3.1. The details of how this is designed and some reasoning regarding security in the PoC can be found in Section 3.2.3. However, something which is not covered by the solution proposed is the publicity of the medications. Hypothetically, should the system be implemented for only a small amount of people, and assuming an attacker could know who those persons were, it could be possible to match blockchain address with a physical identity. For example, it is not terribly difficult to, based on demographics, medical statistics and some social hacking to find out what type of illness a person is suffering from. Knowing what medications are normally used to treat that or those conditions, and finding a similar combination on the blockchain, the physical identity is connected to the blockchain account address. On the other hand, if there are a very large amount of patients, the data is also valuable. Perhaps not as valuable for malicious attackers as for data scientists, pharmaceutical companies or insurances, data is the new gold and should perhaps not be given away so easily. But it still does not violate requirement R1.

Another consideration that needs to be made is that of what takes place before logging on to the blockchain client. Should the IP address of a user be traceable to an account address, then all privacy claims would be flawed. Therefore, great care must be taken when building the surrounding infrastructure, as it cannot be assumed that a large amount of people will use Tor.[1].

[1]For more details on secure(ish) browsing, see https://torproject.org/

User	Traceability	Motivation for fulfilment
Patient	1.1	A registered Patient can access prescriptions by calling the function `getPrescriptions()` through `InfoManager`
	1.2	Fulfilled by the infrastructure of the blockchain and permission levels, both in the component layer and in the contract layer. (See 3.2 and 3.4)
	1.3	A patient can consent to the access to specific prescriptions of Doctors and Pharmacies by calling the function `setConsent()` or `setPrescripConsent()`.
Doctors	2.1	Verified and registered Doctors can fulfil the requirement by calling functions `addPrescription()`, `checkPrescription()` and `getPrescriptions()` in `InfoManager`.
	2.2.	Verification of the identify can be done through a secure messaging service such as Whisper, on the blockchain. The patient can also show a prescription to the Doctor on the Patients device, that the Doctor can also access, thus proving the Patient identity.
	2.3	The Doctor can perform this by calling function `getPrescriptions()` for the right patient account address.
	2.4	Fulfilled by the infrastructure of the blockchain and permission levels, both in the component layer and in the contract layer. (See 3.2 and 3.4)
Pharmacy	3.1	A Pharmacy can verify a Patient's prescription by simply calling the `purchase()` which will trigger a call to the permission database contract.
	3.2	Fulfilled by the infrastructure of the blockchain and permission levels, both in the component layer and in the contract layer. (See 3.2 and 3.4)

Table 4.1: Evaluation of user story acceptance based on the EMP PoC.

There are at least two layers of permissioning in the PoC, because it is a permissioned blockchain, and because the smart contracts contains logic which is independent from the blockchain layer. One could also imagine the application of any number of other security measures a user would have to go through to be granted access and permissions on on the blockchain. Thereby the requirement R2. is satisfied.
Because the implementation proposes the usage of a blockchain, and as long as more than two thirds of the validators on that blockchain are benevolent, the record of the events stated in the requirements; prescription, purchase time and place, can be considered very safely and immutably stored. An additional explicit logging of events can be built into the smart contracts, perhaps triggering a completely separate system to create redundancy in the keeping of records. This satisfies requirement R3.
The smart contracts are built in modular fashion and with a specific contract-managing contract. This means that any updates to be made to the system of smart contracts simply need to make a function call to the `Cmc`. The users will not experience any changes, or the applications communicating with the blockchain application will not experience any changes since the address of the interface contract, the `InfoManager`, remains the same.

4.3 Outcome

Given the considerations taken to functionality, completeness and performance (in terms of security and privacy), the artefact and the reasoning behind it is a strong argument for the usage of such an application for an EMP.

Chapter 5

Conclusion

5.1 Summary of results

In this thesis, a proof-of-concept application was built to function as an electronic medication plan in a completely decentralised way. In order to achieve a peer-to-peer network secure enough to store personal information a system of smart contracts developed in the Solidity programming language in combination with a permissioned blockchain architecture and common cryptographic tools was proposed. The resulting artefact was then (n.b. post-ex) evaluated according to established design research criteria and found to fulfil all the necessary requirements. Although the evaluation criteria were fulfilled, it is important to take notice that no claims on the security outside of the PoC can be made, based on this thesis. There are ways a private key can be stolen, note that most cases of credit fraud or are not performed by hacking into a computer, but by simply resetting an email-account's password over the phone.

As for the research questions, they have all been answered by the theory summarised in the thesis, as well as the architecture of the application and the reasoning around it. The research questions are shortly summarised here:

- **Research question 1: What are the requirements for storage of prescriptions, patient-, doctor- and pharmacy-profiles on a blockchain application for prescriptions?**
- **Research question 2: How can the architecture of a blockchain application for privacy-preserving data-sharing between known, but not necessarily trusted, parties look like?**

- **Research question 3: How can a blockchain application for prescriptions handling be built in order to ensure that each patient has access control over prescriptions, that only certified doctors can prescribe medications and that pharmacies who sell medications perform controls over prescriptions?**

Research question 1 is answered in Section 3.1 and the tables therein. Additionally, the background of those requirements is exposed i Chapter 2.
Research question 2 is answered in Section 3.2.1, where the design of the implementation is shown. One possible architecture of an electronic medication plan is explained, based on blockchain technology and a system of smart contracts.
Research question 3 is answered through the design of the system of smart contracts, as well as the discussion and design choices made in Section 3.2.3 and in Section 4.2.1.

5.2 Discussion

5.2.1 Generalisation and extension into other domains

The objective of this thesis was to solve a very specific problem in the health care sector, however, one can not fail to realise the potential cross-over effects it could have on other industries. It is essentially an application which lets users register information and then in a highly controlled manner share it with distinctive partners. Consider a group of banks and perhaps even other regulated entities who are highly dependent on information about their customers. This is basically every company or organisation which provides a financial service, or an internet provider, or mobile phone operator. All of them need to know personal information on their customers, by law. Now let us apply the concept of the PoC to this situation. A customer registers and encrypts information on the blockchain using slightly modified smart contracts. When a bank or other organisation needs to have access to that information, they simply make a request to the customer who can reject or accept the request. What makes it different from a normal centralised server is that there is immutable traceability, and more importantly, customers could have the power and right to question the usage of personal information by corporations. Selling of information would have to be consensual on a much clearer level than it is in most systems today.

Another area of application could be in a large corporate setting, where information-sharing is essential but very difficult when dealing with secret or sensitive data.

5.2.2 Future work

Blockchain technology is constantly evolving, both in the private and public sector. Much progress was made during the six months in which this thesis was written, and a research community has begun to form. The introduction of blockchain technology-related topics at research conferences [1] and the launch of blockchain-focussed peer-reviewed research papers [2] are critical efforts in encouraging further research. There is no doubt that much more technological development is needed, both in academia and innovation coming from the industry. The benefits and problems it solves need to be communicated and an understanding for blockchain outside of cryptocurrencies needs to be spread. In the context of this thesis, there are areas of interest which could be further developed. For example, introducing a functionality for automatically controlling medications against a database of known harmful or unintended interactions. Or creating a completely blockchain-based version of electronic health records, including medication plans, vaccinations, etc.
There are also a large potential for fundamental research within distributed computing based on blockchain technology, specifically consensus algorithms and lightweight protocols for Internet-of-Things applications. With regards to the emerging decentralised economic system that is cryptocurrencies, there are many important questions related to game theory. Proof-of-Work solves the consensus problem at an increasingly high computational cost in the Bitcoin system, it would be a very important finding should that problem be solved in a provably secure manner, without the energy cost and the drawbacks of Proof-of-Stake.

[1]51st annual Hawaii International Conference on System Sciences (http://hicss.hawaii.edu/tracks-51/internet-and-the-digital-economy/)
[2]The aptly named *Ledger*, (http://ledgerjournal.org/)

Bibliography

Alfred J. Menezes, S. A. V., Paul C. van Oorschot. (1996). *Handbook of applied cryptography* (5th ed.). CRC Press. Retrieved from cacr.uwaterloo.ca/hac

Azaria, A., Ekblaw, A., Vieira, T., & Lippman, A. (2016). Medrec: Using blockchain for medical data access and permission management. In *2016 2nd international conference on open and big data (obd)* (pp. 25–30). doi:10.1109/OBD.2016.11

Back, A. (1997). A partial hash collision based postage scheme. Announcement. Online Multimedia. Retrieved from http://www.hashcash.org/papers/announce.txt

Bobb, A., Gleason, K., & Husch, M. e. a. (2004). The epidemiology of prescribing errors: The potential impact of computerized prescriber order entry. *Arch Intern Med*, *164*, 785–792.

Braxendale, G. (2016). Bitcoin technology and the NHS. *Digital Health*.

Brown, S. (2016). *Visualise, document and explore your software architecture - software architecture for developers*. Leanpub. Retrieved from https://leanpub.com/visualising-software-architecture

Buterin, V. (2015). Visions, part 1: The value of blockchain technology. Retrieved from https://blog.ethereum.org/2015/04/13/visions-part-1-the-value-of-blockchain-technology/

Buterin, V. (2016). Critical update re: Dao vulnerability. Blog. Retrieved from https://blog.ethereum.org/2016/06/17/critical-update-re-dao-vulnerability/

Cynthia Dwork, M. N. (1992). Pricing via processing or combatting junk mail. In *Crypto '92 proceedings of the 12th annual international conference on advances in cryptology* (pp. 139–147). Springer-Verlag.

Ethereum GitHub Wiki - Proof of Stake FAQ. (2017). Retrieved from https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQ

Ethereum Homestead Documentation - What is Ethereum. (2016). Retrieved from http://ethdocs.org/en/latest/introduction/what-is-ethereum.html

Fischer, M. J. (1983). The consensus problem in unreliable systems (a brief survey). In *International conference on foundations of computation theory*.

Gabriel MH, S. M. (2014). *E-prescribing trends in the United States*. Office of the National Coordinator for Health Information Technology.

Griggs, I. (2015). Retrieved from http://iang.org/papers/intersection_ricardian_smart.html

Hamid, T., Harper, L., Rose, S., Petkar, S., Fienman, R., Athar, S. M., & Cushley, M. (2016). Prescription errors in the national health services, time to change practice. *Scottish Medical Journal*, *61*(1), 1–6. PMID: 27101837. doi:10.1177/0036933015619585. eprint: http://dx.doi.org/10.1177/0036933015619585

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design science in information systems research. *MIS Quarterly*.

Hotmine Inc. (2016). Retrieved from http://en.hotmine.io/

How does blockchain work? (2016). Retrieved from https://www.ibm.com/blockchain/what-is-blockchain.html

Irving, G. & Holden, J. (2016). How blockchain-timestamped protocols could improve the trustworthiness of medical science. *F1000 Research*.

ISO/TC 307, Blockchain and electronic distributed ledger technologies. (2016). Retrieved from https://www.iso.org/committee/6266604.html

Jpmorgan chase hacking affects 76 million households. (2014). *The New York Times*, http://nyti.ms/1rQi4vG.

Kish, L. J. & Topol, E. J. (2015). Unpatients - why patients should own their medical data. *Nature Biotechnology*, *33*, 921–924. doi:10.1038/nbt.3340

Kosba, A., Miller, A., Shi, E., Wen, Z., & Papamanthou, C. (2015). Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *Berkeley-simons secure computation workshop*.

Krawiec, R., Housman, D., White, M., Filipova, M., Quarre, F., Barr, D., … Tsai, L. (2016). Blockchain technology: Opportunities for healthcare. This white paper was developed in response to the Department of Health and Human Services' Office of the National Coordinator for Health Information Technology (ONC) ideation challenge.

Mattila, J. (2016). The blockchain phenomenon. the disruptive potential of distributed consensus architectures. BERKELEY ROUNDTABLE ON THE INTERNATIONAL ECONOMY (BRIE).

Medical Records Project Wins Top Prize at Blockchain Hackathon. (2015). Retrieved from http://www.coindesk.com/medvault-wins-e5000-at-deloitte-sponsored-blockchain-hackathon/

Merkle, R. C. (1988). A digital signature based on a conventional encryption function, *p369*.

Monax - Blockchain explainer. (2016). Retrieved from https://monax.io/explainers/blockchains/

Monax - Solidity explainer: The Five-types model. (2016). Retrieved from https://monax.io/docs/tutorials/solidity/solidity_1_the_five_types_model

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. Online Multimedia. Retrieved from https://bitcoin.org/bitcoin.pdf

Nugent, T., Upton, D., & Cimpoesu, M. (2016). Improving data transparency in clinical trials using blockchain smart contracts [version 1; referees: 3 approved]. *F1000 Research*, *5*(2541). doi:10.12688/f1000research.9756.1

Okada, H., Yamasaki, S., & Bracamonte, V. (2017). Proposed classification of blockchains based on authority and incentive dimensions. In *2017 19th international conference on advanced communication technology (icact)* (pp. 593–597). doi:10.23919/ICACT.2017.7890159

Rivest, R., Shamir, A., & Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *CACM*, *21*(2), 120–126. Retrieved from http://doi.acm.org/10.1145/359340.359342

Stevens, M., Bursztein, E., Karpman, P., Albertini, A., & Markov, Y. (2017). *The first collision for full sha-1*. Retrieved from https://shattered.io

Szabo, N. (1994). *Smart contracts*. Retrieved from http://virtualschool.edu/mon/Economics/SmartContracts.html

Szabo, N. (1997). Formalizing and securing relationships on public networks. *First Monday*, *2*(9). Retrieved from http://ojphi.org/ojs/index.php/fm/article/view/548

Tendermint. (2017). Retrieved from https://tendermint.com/intro van Dijk, M. & Juels, A. (2010). On the impossibility of cryptography alone for privacy-preserving cloud computing. In *Proceedings of the 5th usenix conference on hot topics in security* (pp. 1–8). HotSec'10.

Vasek, M. & Moore, T. (2015). There's no free lunch, even using bitcoin: Tracking the popularity and profits of virtual currency scams. In *International conference on financial cryptography and data security* (pp. 44–61). Springer.

Waltering, I., Schwalbe, O., & Hempel, G. (2015). Discrepancies on medication plans detected in german community pharmacies. *Journal of Evaluation in Clinical Practice*, *21*(5), 886–892. doi:10.1111/jep.12395

Whitfield Diffie, M. E. H. (1976). New directions in cryptography. *IEEE Transactions on Information Theory*, *22*(6), 644–654.

Zyskind, G., Nathan, O., & Pentland, A. S. (2015). Decentralizing privacy: Using blockchain to protect personal data. In *Security and privacy workshops (SPW), 2015 IEEE* (pp. 180–184).

Appendix A

```
pragma solidity ^0.4.11;


// Base class for contracts that are used in a cmc system.
contract CmcEnabled {
    address CMC;

    function setCMCAddress(address cmcAddr) returns (bool result){
        // Once the cmc address is set, don't allow it to be set again, except by the
        // cmc contract itself.
        if(CMC != 0x0 && msg.sender != CMC){
            return false;
        }
        CMC = cmcAddr;
        return true;
    }

    // Makes it so that CMC is the only contract that may kill it.
    function remove(){
        if(msg.sender == CMC){
            selfdestruct(CMC);
        }
    }

}

// Base class for contracts that only allow the infomanager to call them.
// Note that it inherits from CmcEnabled
contract InfoManagerEnabled is CmcEnabled {

    // Makes it easier to check that infomanager is the caller.
    function isInfoManager() constant returns (bool) {
        if(CMC != 0x0){
            address im = ContractProvider(CMC).contracts("infomanager");
            return msg.sender == im;
        }
        return false;
    }
}

// The Contract managing contract.
contract CMC {

    address owner;

    // This is where we keep all the contracts.
    mapping(bytes32 => address) public contracts;

    modifier onlyOwner { //a modifier to reduce code replication
        if (msg.sender == owner) // this ensures that only the owner can access the function
            _;
    }
    // Constructor
    function Cmc(){
        owner = msg.sender;
    }

    // Add a new contract to Cmc. This will overwrite an existing contract.
    function addContract(bytes32 name, address addr) onlyOwner returns (bool result) {
        CmcEnabled cmce = CmcEnabled(addr);
        // Don't add the contract if this does not work.
        if(!cmce.setCMCAddress(address(this))) {
            return false;
        }
        contracts[name] = addr;
        return true;
    }

    function getContract(bytes32 name) constant returns (address addr) {
        return contracts[name];
    }

    // Remove a contract from Cmc. We could also selfdestruct if we want to.
    function removeContract(bytes32 name) onlyOwner returns (bool result) {
        if (contracts[name] == 0x0){
            return false;
        }
        contracts[name] = 0x0;
        return true;
    }

    function remove() onlyOwner {
        address im = contracts["infomanager"];
        address ime = contracts["infomanagerenabled"];
        address perms = contracts["permissions"];
        address permsdb = contracts["permissionsdb"];
        address patient = contracts["patient"];
        address patientinfodb = contracts["patientinfodb"];
        address doctor = contracts["doctor"];
        address pharmacy = contracts["pharmacy"];


        selfdestruct(owner);
    }

}
// The info manager
contract InfoManager is CmcEnabled {

    // We still want an owner.
    address owner;

    // Constructor
    function InfoManager(){
        owner = msg.sender;
    }

    // Attempt to prescribe a new medication to a patient
    function addPrescription(address patAddr, bytes32 prescription) returns (bool res
        ) {
        if (prescription == 0x0 || patAddr == 0x0){
            return false;
        }
        address doctor = ContractProvider(CMC).contracts("doctor");
        address permsdb = ContractProvider(CMC).contracts("permsdb");
        if ( doctor == 0x0 || permsdb == 0x0 || PermissionsDb(permsdb).perms(msg.sender) < 1) {
            // If the user doesn't have the right to prescribe, return false
            return false;
        }

        // Use the interface to call on the doctor contract. We pass drugHash and the patient address along as well.
        bool success = Doctor(doctor).addPatient(msg.sender, patAddr, prescription);

        // If the transaction failed, return the token to the sender
        if (!success) {
            msg.sender.send(msg.value);
        }
        return success;
    }

    function addPatient(address patAddr, bytes32 prescription) returns (bool res) {
        uint8 requestCode = 2;
        if (patAddr == 0x0){
            return false;
        }
        address doctor = ContractProvider(CMC).contracts("doctor");
        address perms = ContractProvider(CMC).contracts("perms");
        if ( doctor == 0x0 || perms == 0x0 || Permissions(perms).checkPerms(msg.sender) < 1 || Permissions(perms).checkConsent(msg.sender, requestCode, patAddr)) {
            // If the user doesn't have the right to prescribe return false
            //msg.sender.send(msg.value);
            return false;
        }

        // Use the interface to call on the doctor contract. We pass drugHash and the patient address along as well.
        bool success = Doctor(doctor).addPrescription(msg.sender, patAddr, prescription);

        // If the transaction failed, return the token to the sender
        if (!success) {
            msg.sender.send(msg.value);
        }
        return success;
    }

    //function getPrescription

    //function confirmPrescription

    //function add

    //function checkPrescripConsent

    //function setConsent

    //function setPrescripConsent

    // Buy/check-out prescribed medication. Should be called by pharmacist
    function purchase(address patAddr, bytes32 drugHash) returns (bool res) {
        if (drugHash == 0){
            return false;
        }
        address pharmacy = ContractProvider(CMC).contracts("pharmacy");
        address permsdb = ContractProvider(CMC).contracts("permissionsdb");
        if ( pharmacy == 0x0 || permsdb == 0x0 || PermissionsDb(permsdb).perms(msg.
            sender) < 2) {
            // If the caller doesn't have permission to buy prescribed medication,
                return false.
            return false;
        }

        // Use the interface to call on the doctor contract
        bool success = Pharmacy(pharmacy).purchase(patAddr, drugHash);

        // If the transaction succeeded, pass the token back to the caller.
        if (success) {
            msg.sender.send(msg.value);
        }
        return success;
    }

```

```
        // Set the permissions for a given address.
    function setPermission(address addr, uint8 permLvl) returns (bool res) {
        address permsdb = ContractProvider(CMC).contracts("permissionsdb");
        if ( permsdb == 0x0 ) {
            return false;
        }
        uint8 userPerm = PermissionsDb(permsdb).perms(addr);
        if(userPerm < 3) {
            return false;
        }
        return PermissionsDb(permsdb).setPermission(addr, permLvl);
    }

    function setConsent(address addr, uint8 consentCode) returns (bool) {
        address patient = ContractProvider(CMC).contracts("patient");
        if(patient == 0x0) {
            return false;
        }
        return Patient(patient).setConsent(msg.sender, addr, consentCode);
        }
}

    // Interface for getting contracts from Cmc
contract ContractProvider {
    function contracts(bytes32 name) returns (address addr) {}
}

// Permissions database
contract PermissionsDb is CmcEnabled {

    struct consentedPatientPrescriptionTuple {
        address doctor;
        bytes32 prescription;
    }

    struct consentPatientCode {
        address docOrPharm;
        uint8 consentCode;
    }
    mapping (address => uint8) public perms;
    mapping (address => consentPatientCode) public patientDoctorConsent;
    mapping (address => consentedPatientPrescriptionTuple) public
        prescripPatientConsent;

    // Set the permissions of an account.
    // Permissions:
    // 0 - Patient permissions, e.g. only allowed to read info related to one's own
        address
    // 1 - Pharmacy permissions, e.g. allowed to read info about patient who shared
        it's address
    // 2 - Doctor permissions, allowed to add patients, add prescriptions and read
        info about patients
    function setPermission(address addr, uint8 perm) returns (bool res) {
        if(CMC != 0x0){
            address permC = ContractProvider(CMC).contracts("perms");
            if (msg.sender == permC ){
                perms[addr] = perm;
                return true;
            }
            return false;
        } else {
            return false;
        }
    }
    // Allow patients to consent to other users to perfom actions on their behalf
        such as adding prescriptions
    // Permissions:
    // 0 - no permission
    // 1 - addr, being a pharmacy, can sell prescribed medication to sender
    // 2 - addr, being a doctor, can prescribe medication for sender
    // 3 - addr, being a doctor, can add sender as patient with a prescription
    function setConsent(address patAddr, address docOrPharmAddr, uint8 consentCode)
        returns (bool res) {
        if(CMC != 0x0){
            address patientC = ContractProvider(CMC).contracts("patient");
            var cons = consentPatientCode(docOrPharmAddr, consentCode);
            if(consentCode < 4 && msg.sender == patientC){
                patientDoctorConsent[patAddr] = cons;
                return true;
            } else {
                return false;
            }
        }
        return false;
    }

    function setPrescripConsent(address patAddr, address docAddr, bytes32
        prescription) returns (bool res) {
        if(CMC != 0x0){
            address patientC = ContractProvider(CMC).contracts("patient");
            var consPP = consentedPatientPrescriptionTuple(docAddr, prescription);
            if(msg.sender == patientC){
                prescripPatientConsent[patAddr] = consPP;
                return true;
            } else {
                return false;
            }
        }
        return false;
    }
}

// Permissions
contract Permissions is InfoManagerEnabled {

    // Set the permissions of an account.
    function setPermission(address addr, uint8 perm) returns (bool res) {
        if (!isInfoManager()){
            return false;
        }
        address permdb = ContractProvider(CMC).contracts("permsdb");
        if ( permdb == 0x0 ) {
            return false;
        }
        return PermissionsDb(permdb).setPermission(addr, perm);
    }

    function checkPerms(address addr) returns(uint){
    }

    function checkConsent(address docAddr, uint8 consentCode, address patAddr)
        returns(bool){
    }
}



// The patientInfo database
contract PatientInfoDb is CmcEnabled{

    // List element
    struct Patient {
        address prev;
        address next;
        // Data
        bytes32[] prescriptions; //Array of prescriptions;
        address[] responsible; // Array of responsible person for corresponding
            prescription
        bool init;
    }

    uint public size;
    address public tail;
    address public head;
    mapping (address => Patient) public patientList;
    //mapping (address => bytes32[]) public prescriptions;

    // Add a new patient with a prescription. This will overwrite an existing
        prescription. 'internal' modifier means
    // it has to be called by an implementing class.
    function addPatient(address respAddr, address patAddr, bytes32 prescription)
        returns (bool) {
        if(CMC != 0x0) {
            address doctor = ContractProvider(CMC).contracts("doctor");
            if(msg.sender == doctor) {
                if(patientList[patAddr].init == false) {
                    bytes32[] presc;
                    presc.push(prescription);
                    address[] resp;
                    resp.push(respAddr);
                    Patient memory pat = Patient(0,0,presc,resp,true);
                    patientList[patAddr] = pat;
                    // Two cases - empty or not.
                    if(size == 0){
                        tail = patAddr;
                        head = patAddr;
                    } else {
                        patientList[head].next = patAddr;
                        patientList[patAddr].prev = head;
                        head = patAddr;
                    }
                    size++;
                    return patientList[patAddr].init;
                }
            }
        }
        return false;
    }

    function addPrescription(address resp, address patAddr, bytes32 prescription)
        returns (uint result) {
        if(patientList[patAddr].init == false) {
            //return code 0 means patient not found
            return 0;
        } else if(prescription == 0) {
            //return code 1 means no prescription given
            return 1;
        } else {
            //return code 2 means prescription added to patient
            patientList[patAddr].prescriptions.push(prescription);
            patientList[patAddr].responsible.push(resp);
            return 2;
        }
    }

    function _getPrescriptionByIndex(address patAddr, uint8 index) returns (bytes32)
        {
        if(CMC != 0x0) {
            address doctor = ContractProvider(CMC).contracts("doctor");
            address patient = ContractProvider(CMC).contracts("patient");
```

```
            if(msg.sender == doctor || msg.sender == patient) {
                if(patientList[patAddr].init == false) {
                    //patient not found
                    return 0x0;
                } else {
                    return patientList[patAddr].prescriptions[index];
                }
            }
        }
    }

    function getNumOfPrescriptions(address patAddr) returns (uint256){
        if(CMC != 0x0) {
            address doctor = ContractProvider(CMC).contracts("doctor");
            address patient = ContractProvider(CMC).contracts("patient");
            if(msg.sender == doctor || msg.sender == patient) {
                if(patientList[patAddr].init == false) {
                    //patient not found
                    return 0;
                } else {
                    return patientList[patAddr].prescriptions.length;
                }
            }
        }
    }

    //requires the caller to know the hash of the prescription, confirms whether
        prescription exists or not
    function confirmPrescription(address patAddr, bytes32 prescription) returns (bool
        ){
        if(CMC != 0x0) {
            //Ensure caller is doctor or pharmacy
            address doctor = ContractProvider(CMC).contracts("doctor");
            address pharmacy = ContractProvider(CMC).contracts("pharmacy");
            address patient = ContractProvider(CMC).contracts("patient");
            if(msg.sender == doctor || msg.sender == pharmacy) { //Ensure sender is
                doctor or pharmacy
                if(patientList[patAddr].init == false || prescription == 0) {
                    return false;
                }
                for (uint i = 0; i<patientList[patAddr].prescriptions.length; i++) {
                    if(patientList[patAddr].prescriptions[i] != 0) {
                        return true;
                    }
                }
            } else if(msg.sender == patient || msg.sender == doctor) { //If sender is neither doctor nor pharmacy, check if sender is patient
                if(patientList[patAddr].init == false || prescription == 0) { //only allow patient to confirm
                    return false; // proprietary prescriptions
                }
                for ( i = 0; i<patientList[patAddr].prescriptions.length; i++) {
                    if(patientList[patAddr].prescriptions[i] != 0 && i<1000) { //Check if i:th
                        return true; //prescription is the requested one.
                        } //Cap at 1000 if list has length>1000
                }
            }
        }
        return false;
    }

    function isPatient(address patAddr) returns (bool){
        if(patientList[patAddr].init == true) {
            return true;
        } else {
            return false;
        }
    }

    //function getPrescriptionByIndex(address patAddr, uint8 index) returns (bytes32) {

    //}

    // function getAllPrescriptions(address patAddr) returns (bytes32[] prescriptions){
    // if(patientList[addr] != 0x0) {
    // return false;
    // }
    // return patientList[patAddr].prescriptions;
    // }



}
// Patients
contract Patient is InfoManagerEnabled {
    // Consent levels:
    // 0 - no permissions
    // 1 - addr, being a pharmacy, can sell prescribed medication to sender
    // 2 - addr, being a doctor, can prescribe medication for sender
    // 3 - addr, being a doctor, can add sender as patient with a prescription
    // Set level of consent for a specific address
    function setConsent(address patAddr, address addr, uint8 consentCode) returns (
        bool success) {
        if (!isInfoManager()){
            return false;
        }
        address permissionsdb = ContractProvider(CMC).contracts("permissionsdb");
        if ( permissionsdb == 0x0 ) {
            // If the user sent a token, we should return it if we can't prescribe.
            return false;
        }

        // Use the permissionsdb-interface to call on the permissionsdb contract to
            set consent level
        success = PermissionsDb(permissionsdb).setConsent(patAddr, addr, consentCode);

        return success;
    }
    //set prescription consent
    function setPrescripConsent(address patAddr, address docAddr, bytes32
        prescription) returns (bool res) {
       if (!isInfoManager()){
            return false;
        }
        address permissionsdb = ContractProvider(CMC).contracts("permissionsdb");
        if ( permissionsdb == 0x0 ) {
            //Can't set consent.
            return false;
        }

        // Use the permissionsdb-interface to call on the permissionsdb contract to
            set consent level
        res = PermissionsDb(permissionsdb).setPrescripConsent(patAddr, docAddr,
            prescription);

        return res;
    }

    function _getNumOfPrescriptions(address patAddr) internal returns (uint256) {
        if (CMC != 0x0) {
            address patientinfodb = ContractProvider(CMC).contracts("patientinfodb");
            if(patientinfodb != 0x0) {
                PatientInfoDb pat = PatientInfoDb(patientinfodb);
                uint256 len = pat.getNumOfPrescriptions(patAddr);
                return len;
            }
         }
    }
    function getPrescriptions(){

    }

}

// A Doctor
contract Doctor is InfoManagerEnabled {

    // Register a Patient with address and (optionally) prescription
    function addPatient(address docAddr, address patAddr, bytes32 prescription)
        returns (bool res) {
        if (!isInfoManager()){
            return false;
        }
        address patientInfodb = ContractProvider(CMC).contracts("patientInfodb");
        if ( patientInfodb == 0x0 ) {
            // If the user sent a token, we should return it if we can't prescribe.
            msg.sender.send(msg.value);
            return false;
        }

        // Use the interface to call on the patientInfodb contract. We pass msg.value
            along as well.
        bool success = PatientInfoDb(patientInfodb).addPatient(docAddr, patAddr,
            prescription);

        // If the prescription failed, return the Token to the caller.
        if (!success) {
            msg.sender.send(msg.value);
        }
        return success;
    }

    // prescribe a medication, drughash for patient with address patAddr
    function addPrescription(address docAddr, address patAddr, bytes32 prescription)
        returns (bool res) {
        if (!isInfoManager()){
            return false;
        }
        address patientInfodb = ContractProvider(CMC).contracts("patientInfodb");
        if ( patientInfodb == 0x0 ) {
            // If the user sent a token, we should return it if we can't prescribe.
            msg.sender.send(msg.value);
            return false;
        }

        // Use the interface to call on the patientdb contract. We pass msg.value
            along as well.
        uint result = PatientInfoDb(patientInfodb).addPrescription(docAddr, patAddr,
            prescription);

        if(result == 0 || result == 1) {
            return false;
        } else if(result == 2) {
            return true;
        } else {
            return false;
        }
```

```
}

function confirmPrescription(address patAddr, bytes32 drugHash) returns (bool) {
    if (!isInfoManager()){
        return false;
    }
    address patientInfodb = ContractProvider(CMC).contracts("patientinfodb");
    if ( patientInfodb == 0x0 ) {
        msg.sender.send(msg.value);
        return false;
    }

    // Use the interface to call on the patientdb contract. We pass msg.value
        along as well.
    return PatientInfoDb(patientInfodb).confirmPrescription(patAddr, drugHash);

    }
//Check if
function checkPrescripConsent(address patAddr, bytes32 prescriptions) internal
    returns (bool) {

}

function isPatient(address patAddr) returns (bool) {
    if (!isInfoManager()){
        return false;
    }
    address patientinfodb = ContractProvider(CMC).contracts("patientinfodb");
    if ( patientinfodb == 0x0 ) {
        return false;
    }

    return PatientInfoDb(patientinfodb).isPatient(patAddr);
}

// //get all prescriptions of a patient
// function getAllPrescriptions(address patAddr) returns (bool res){
// if (!isInfoManager()){
// return false;
// }
```

```
    // address doctordb = ContractProvider(DOUG).contracts("Doctordb");
    // if ( doctordb == 0x0 ) {
    // // If the user sent a token, we should return it if we can't prescribe.
    // msg.sender.send(msg.value);
    // return false;
    // }

    // // Use the interface to call on the doctordb contract. We pass msg.value along
         as well.
    // bool success = DoctorDb(doctordb).getAllPrescriptions.value(patAddr)(msg.
        sender);

    // // If the prescription failed, return the Token to the caller.
    // if (!success) {
    // msg.sender.send(msg.value);
    // }
    // return success;
    // }
}


//A Pharmacy
contract Pharmacy is InfoManagerEnabled {
    event PrescriptionSold;

    //Confirm prescription
    function confirmPrescription(address patAddr, bytes32 prescription) returns (bool
        ) {
        if (!isInfoManager()){
            return false;
        }
        address patientInfodb = ContractProvider(CMC).contracts("patientInfodb");
        if ( patientInfodb == 0x0 ) {
            msg.sender.send(msg.value);
            return false;
        }

        // Use the interface to call on the doctordb contract.
        return PatientInfoDb(patientInfodb).confirmPrescription(patAddr, prescription)
            ;

    }

    function purchase(address patAddr, bytes32 prescription) returns (bool res) {
        //call infomanager function purchase
    }
}
```