MIT ViralMedia

MedRec: Patient Centered Medical Records Using A Distributed Permission Management System

Author: Nchinda Nchinda

Research Assistants: Nchinda Nchinda, Agnes Fury Cameron, Kallirroi Retzepi

Advisor: Andrew Lippman

MIT Media Lab 2018

Abstract

MedRec is a simple, distributed system for personal control of identity and distribution of personal information. The work is done in the context of a medical information distribution system where patients retain control over who can access their data. We create a network of trusted data repositories, the access to which are determined by a set of 'smart contracts'. These contracts are stored on a distributed ledger maintained by those who generate data. The distributed nature of the system allows unified access from diverse sources in a single application with no intermediary. This increases patient control while retaining a measure of privacy of both data content and source. MedRec is amenable to extensions for decentralized messaging and distribution of information to third parties such as medical researchers, healthcare proxies, and other institutions. The system is based on a blockchain that contains smart contracts defining user identity and distribution specifics.

Contents

List of Figures

1 Introduction

The major sites on the internet today represent the modern kingdom. Facebook, Google, Amazon, each control a massive domain. Each of these companies collects the data of millions of people in a completely opaque way. With so much information about people flowing through their servers these companies posses an enormous amount of control over the perception and dissemination of information.

This creates one of the major problems with the internet and digital media today. People interacting with these sites lack control over their own identity as well as information relating to and supporting it. As a person interacts with sites across the Internet they leave behind bits and pieces of their identity as metadata: phone numbers, credit cards, addresses, etc. If any of this information changes it's almost impossible to track down all the occurrences across the web and update it. Additionally, the more sites these bits are stored on the more likely they are to be stolen via hacks or other security breaches.

A person may have multiple identities, but once an identity is formed it cannot be changed. An identity is formed through a history of events which together make it unique. The defining parts of this identity cannot be destroyed. As a person's identity is spread among more data silos on the internet, they lose the ability to maintain and control their own identity. That power is then transferred to the owners of the data silos.

Many have spoken to this problem, and specifically its applicability in the realm of electronic medical records, in the past [1] [2] [3]. I propose a modern solution using a *blockchain* as the identity registry. A blockchain most generally is a distributed, append-only ledger of information that in at least one instantiation, bitcoin [4], has defended itself against all attempts to undermine or take it down. Additionally, it has been able to do this in a completely open, transparent, and distributed manner- the polar opposite from some of the titans of the modern Internet. A particularly salient aspect of blockchain technology is it allows every participant to maintain a self-sovereign identity. There is no single, central repository which can be taken down or corrupted. Instead information is stored in a distributed, ownerless system.

I also note that identity, like retirement plans and medical records themselves, are important aspects of life that most people avoid thinking deeply about. That means that an important aspect of the Medrec work is making it be something that people will want to use.

My thesis builds upon past work done in the Viral Communications Group on MedRec by Ekblaw et al. [5]. Version 2.0 of MedRec changes some of the fundamental designs of the original by leveraging technologies that have been invented since the first iteration was composed. The full system architecture, smart contracts, and majority of the implementation are my work. I was greatly assisted by two other research assistants in Viral Communications, Kalli Retzepi and Agnes Cameron. Kallirroi worked on implementing and styling the user interface, while Agnes worked on various components in the database manager and user interface and setting up a simulated provider database.

I propose MedRec v2.0 as a full system implementation for a distributed access management system. Medrec is designed for the management of medical records, but at its core it is simply an access management system. Its main goal is to give patients control over the distribution of their private data. By building such an identity control system for medical records, I simultaneously solve a social problem and provide a model for other applications.

This thesis will create the basic blockchain-based means for identity control: protection against identity fraud, recovery mechanism in case of fraud, a means to register contracts through which your information can be used by others, and a secure, distributed system for holding that information.

We, at Viral Communications, agreed the most prudent design of this project would be non-commercial, free, and open. This is a secondary goal of the system. Our rationale being so that MedRec can become widely used and developed. This avoids creating a multiplicity of commercial intermediaries that each have their own proprietary solutions.

In the following pages, I will give arguments in favor of my choice of blockchain, describe the general implementation, then go more in depth into how some edge cases are handled. Sections 1-4 are a background on the technologies used. Section 5-9 describe the four different components of MedRec. The remaining sections go more in depth into particular features I find interesting.

2 MedRec Concept

In this section I go deeper into the objectives of MedRec. MedRec ties together disparate health provider data sets with blockchain technology to provide patients with a unified access management system for their medical data. MedRec is a network, not a website or intermediary. It is not an additional data silo for patient data, but provides a means of accessing data that already exists in disparate data silos. Part of MedRec is software that sits on top of a provider's database and handles external requests for data. Healthcare providers are allowed to continue to maintain their patient records in whatever way they see fit. Each request is checked against an access list to ensure the requester is authorized to read/write the data. Before blockchain technology this access list would have had to be stored in a database managed by a trusted third party. Blockchain technology allows MedRec to store these access permissions in a distributed way, among multiple parties in the system. It also allows MedRec to manage changes to those access permissions in a decentralized way. Each patient is able to give others fine-grained read/write access to portions of their medical records. Furthermore patients can authorize others to act on their behalf as caretakers and manage access to their data for them.

Ethereum advanced the standard for smart contracts on blockchain based systems as a method for running Turing Complete code in decentralized systems. MedRec has an open architecture for contracts, users can expressly view the underlying code that governs their contracts. If they are skilled developers they may also make modifications to their contracts and release them onto the blockchain.

The smart contracts used in MedRec have an open architecture. This means they have the capacity to be inspected and modified by users [6]. Patients deploy their own contracts via their MedRec instance so they may choose to deploy contracts with different features. The smart contracts reside on a distributed blockchain, providing storage redundancy. Each agent is required to possess a public/private keypair and using their keypair is able to interact with the smart contracts and other members of the MedRec network.

These are the basic requirements for MedRec:

1. MedRec will not be a proprietary system. In a project designed to facilitate patient centered access to medical records, it would be discordant to design the system in a closed way. To fully allow comments and feedback to its design MedRec should be designed under a Free Software License. This way agents can know exactly what they are running and ensure for themselves their software is not mistreating their medical data.

2. Patients should have a singular portal to access their data across various provider databases. This portal should provide the same interface for patients regardless of the means by which providers are storing patient data. There should also be a clear process for new providers to be integrated into the system. This means MedRec has to be interoperable across and agnostic to a range of storage implementations by providers.

3. MedRec should provide granular access permissions for patients. A patient should be able to specify a specific medical record, such as blood pressure, x-ray results, prescriptions, as well as who should have access to that data. It should also distinguish between who is allowed to read the data, and who is allowed to make modifications to it.

4. MedRec should provide improved data quality and quantity for medical research. MedRec empowers users to safely share portions of their medical data with a third party, thus empowering researchers with a new dataset.

5. MedRec should incorporate a distributed method of access control data storage and retrieval. Even if the majority of nodes on the network are offline a user should still be able to view and modify their medical record access permissions. Raw data is still stored within provider's own databases, so if a hospital is destroyed all of their patient's medical data will likely still be lost.

6. MedRec should obfuscate any association between real world identities and agents in the system. Even revealing metadata about the relationship between a patient and a provider is harmful. For example: an employer may infer from a patient's ongoing relationship with a radiology specialist that they are sick and factor that into a hiring decision.

7. There should be a cryptographically secure association between data in the system and real patient data stored in a medical provider's database. Patient's and providers should be able to attest to the validity of the patient provider relationships stored on the blockchain.

8. Bitcoin and most blockchain based systems require the use of cryptocurrency to facilitate interactions with the blockchain. MedRec should bypass this and remove cryptocurrency management as a requirement for using a blockchain based protocol.

9. Only authorized agents should be able to make changes to a patient's data. Authorization should be a cryptographically secure process.

10. Patients as well as their caretakers are notified in real time of changes to records

11. All agents in the system there should have their identity securely authenticated for interactions with each other.

12. There should be separation of authority to minimize the impact of security breaches. It may be allowable for a hacked or otherwise compromised provider to generate false data for all their patients, but it should be impossible for them to harm the relationship their patients have with other providers. Similarly actions taken by a patient should not change the data of other patients nor compromise the relationship between other patients and their providers.

13. MedRec should be able to integrate nontraditional data sources. Wearable devices and mobile phones generate a wealth of day to day health data which could be managed by this system.

MedRec at its core is an distributed, accessible, reliable, and consistent access management system. However, these features enable a diverse range of applications. For example:

- Emergency Room - A patient with a medical identification tag can expedite the time it takes for an emergency room to identify exactly what is special about their medical condition. With a distributed access management system a patient can selectively grant all emergency rooms access to certain medical records. They could also create more expressive permissions, such as giving a second person the right to make decisions on their behalf during times of incapacitation.
- Patient Relocation - When a patient switches to a new medical provider, all of their medical records remain digitally accessible to them and (if they choose) to their new provider.
- Pharmacy - When visiting an out of network pharmacy, a patient can give the pharmacy temporary access to their prescription data. This allows the pharmacy to verify the veracity of prescriptions in real time against the issuing provider's database.
- Parental Custodians - A minor can be placed under the custodianship of their parents. A smart contract governing the minor's access permissions can be set to automatically grant them the ability to manage their own medical data when they become of age. The smart contract can be designed with a fixed date and in a way that is public to both the minor and their custodians.

3 Background

Before continuing with the implementation details of MedRec, this section provides a background on blockchain technology. The choice to use a blockchain, and furthermore which among multiple blockchain technologies is one of the primary differences between MedRec 2.0 and the original implementation. The choice also provides the foundation for the MedRec Network, defining the bounds of what can be accomplished using the system and the medium by which agents in the system interact.

3.1 Bitcoin

The most used blockchain based system is Bitcoin. Bitcoin is an example of a distributed, decentralized system that is successfully operating on the global scale. All other blockchain based systems are manipulations or extensions of the capabilities afforded by using the Bitcoin network. Although Bitcoin is not used in this thesis, understanding it is a stepping stone to understanding the technologies that are.

Bitcoin is a cryptocurrency and consensus network used by millions of people around the world for sending monetary transactions. Unlike traditional fiat currencies, it is not backed by a central bank or government. Instead, every user, developer, and participant in the network has to agree to run the same rules coded into the software. Instead of storing the state of account balances in a central trusted entity, the entire network must come to consensus on the current state and every transaction that changes that state.

The problem that Bitcoin solves is generally called the Byzantine Generals Problem [7]. This is a problem that plagues distributed systems with unreliable links, such as the Internet and Bitcoin. A Byzantine Fault Tolerant system is one that solves the Byzantine Generals Problem. The problem arises when multiple parties cannot come to consensus about the state of the world due to untrustworthy communication. The parties involved cannot agree on any plans, as those plans as well as their confirmations may be lost in transit. This problem is easily solved in centralized systems where one party can unilaterally make decisions, but in distributed systems even the election of a leader is subject to failure.

Bitcoin solves this problem and allows nodes in the network to come to consensus by using a data structure called the blockchain. A blockchain is a list of blocks starting at some ”genesis” block and extending until the present time. Transactions are consolidated into batches called *blocks* and appended to the blockchain. The bitcoin blockchain contains all the transactions on the network going back to the first Bitcoin transaction in 2009. Different blockchains utilize different mechanisms to deter invalid blocks and fraudulent transactions. The one which Bitcoin uses, and the easiest to understand is *Proof of Work* (PoW). Nodes can opt in to be miners on the network, which have the ability to append blocks to the blockchain. Miners are allowed to choose which transactions to include and which block in the history of the blockchain to build off of. The rationale behind the name PoW is that adding a new block requires finding the solution to a very hard cryptographic puzzle. The puzzle is regularly updated in difficulty to maintain an approximately fixed generation time of new blocks. The entire Bitcoin network working together to find the solution to Bitcoin's puzzle targets ten minutes to find a solution, although mining is a poisson point process so this timing is not exact.

3.2 Ethereum

Ethereum is a blockchain based technology that is used in this project. Ethereum provides capabilities that allow for a much greater breadth of applications than those which Bitcoin could provide.

3.2.1 SmartContracts

Nick Szabo first proposed the idea of smart contracts in 1996 to bring traditional law and paper contracts into the digital age [8]. At the time smart contracts were a theoretical construct of codified representations of agreements between parties that could supercede the functionality of traditional paper based contracts. A smart contract could be a living document, almost an agent in itself that once authorized by its signers could autonomouly act to uphold the rules within its code. As of now smart contracts are still primitive and can only support basic if-this-then-that statements and are limited by the blockchain to storing small amounts of data. Even so, they are still being used for a broad class of applications ranging from identity management, provably fair gambling, fundraising, marketplaces, and access control.

The Ethereum Whitepaper [9] [10] and Nick Szabo's paper both provide substantial examples of ways in which smart contracts could change various social interactions. Smart contracts both empower new paradigms of interaction and introduce new design challenges.

3.2.2 Proof of Authority

Ethereum also introduced an alternative system of block validation called Proof of Authority (PoA) as an alternative to Proof of Work. While the main Ethereum public chain runs on PoW, there are multiple testnets and private chains running on PoA. PoA operates by a federated model, only certain nodes are allowed to make additions to the blockchain. This changes the security model, the system is more susceptible to fraudulent transactions by way of one of those authorities getting hacked. At the same time, it allows faster transaction confirmation times and the ability to remove the cryptocurrency aspect from blockchain technology. Private blockchains bridge the gap between public blockchains which are trustless and pseudonymous, and databases which are centralized. In networks where certain entities are already semi-trusted it may be prudent to use a private blockchain instead of public with those entities as the custodians.

4 Choice of Blockchain

Before choosing to use a private blockchain I looked at many other options. This section discusses not only why MedRec is implemented using a PoA blockchain, but what the shortcomings would be if it was implemented on another type of system. When implemented correctly blockchain based systems can provide:

1. Consensus among a globally distributed set of peers so they are in agreement on what is true and and what is not true
2. Efficient reconciliation between unharmonious states of reality to the most probable state
3. A distributed data storage and computation network
4. A system that runs without the need for a central server or trusted authority to resolve disputes or coordinate participants
5. Transparency to everyone involved of the current set of rules agreed upon

6. Access for anyone with an internet connection to participate

7. Pseudonymous identities- users are not inherently linked to their real world identities so de-anonymization requires targeted attacks

8. Personalized control over what amount of resources should be expended verifying a particular transaction before it can be considered valid

Deciding on a blockchain required analyzing what a potential MedRec would look like on each of those systems. A few analysis of alternatives used by other systems are below.

4.1 Bitcoin

Bitcoin is the original use of blockchain technology and it's still widely considered the most stable and secure implementation. This is the counterpoint to the fact that maintaining the security of Bitcoin's blockchain requires a large amount of energy and computational power. Data stored on the Bitcoin blockchain can generally be assumed to be indelible. In a version of MedRec based on this blockchain, data saved on the blockchain would remain accessible to the world in perpetuity, with no need for further action by users. Archiving services would be provided by Bitcoin miners. Unfortunately this design makes MedRec subject to the externalities of Bitcoin. In the future the network may change in a way that prioritizes the primary use of cryptocurrency and harms secondary uses such as data storage.

Bitcoin has a simple scripting language which is usually used to represent the inputs and outputs of transactions. It allows for arithmetic and comparative operations on bits, signature generation and verification, time locks, and call stack operations. Each transaction is also allowed to include up to 40 bytes of data. This is enough space to include a small hash or reference to data living off the blockchain such as a land title [11] or graduation certificate [12]. The data itself has to be managed by other means, creating more complexity as applications have to keep track of the location of the pointers on the blockchain and then query another service to finish fulfilling a request.

Additionally every transaction on the Bitcoin networks costs money and takes time. The current fee for a transaction on the network to get confirmed in about 10 minutes is currently $4, but this number can spike up during periods of heavy usage. Causes of the transaction influx can vary, but one side effect is a backlog of transactions. Heavy utilization of the blockchain gives miners an incentive to order the transactions placed into blocks, those paying higher transaction fees getting processed first. One project using the Bitcoin Blockchain for land title registry in the Republic of Georgia has an estimated 5-10 cents cost per registration [11]. This is feasible in a system where land ownership changes slowly, on the order of years, but not at the scale of MedRec.

To mitigate these costs and delays, applications using Bitcoin for storage tend to batch transactions so they can be submitted with a higher fee and get confirmed faster. The downside is batching transactions also causes delays. Overall, this would lead to a highly inconsistent user experience. For some of the use cases of MedRec a 10 minute delay is unacceptable, especially when an alternative exists. The strongest reason against using the Bitcoin network is changes to medical relationships should always be free. There should never be a case where a patient does not have the funds to add or remove a third party's right to view sensitive data.

4.2 Public Ethereum

The public Ethereum network has some of these same limitations. An Ethereum transaction on average takes 15 seconds to confirm and the average transaction fee is $.30. For both parameters this is a marked improvement over Bitcoin. But an optimal solution would get rid of the cost completely.

All transactions on the public Ethereum network are public perforce and must be stored by all the full nodes on the network. This causes implementation difficulties for a project such as MedRec which is dealing with sensitive data. Static analysis has been used to deanonymize users of Bitcoin [13]. Since data on the blockchain is persistent, it lacks retroactive identity confidentiality. At some point in the future an algorithm may be developed to correlate and deanonymize many users of a public blockchain. I have also done research previously [14] exploring real time deanonymization on the Ethereum network. By not relying on the inherent pseudonymity of public blockchain MedRec can defend against this attack.

Along with its powerful Turing complete programming language, the public Ethereum blockchain brings along all it's programs. A full node on the network must process all the transactions and blocks generated on the network. The Ethereum developers have advanced plans to mitigate this, but the overarching fact remains that for the system to remain operational some nodes must track everything that happens on the network. On a private blockchain I can model the growth of the system and experimentally stress test it. Nodes will not have to store any data that isn't required for the system to run. Previous work has shown how a single Ethereum application can cause crippling externalities to the rest of the network [15].

4.3 Alternative EVMs

One way of thinking of smart contracts is programs running in a distributed, sandboxed, virtual machine. Ethereum calls this the Ethereum Virtual Machine (EVM). The machine has properties characteristic of an operating system such as a stack, indexable storage, and an assembly style language.

Other alternatives to and implementations of the Ethereum EVM exist, allowing the same code to be run on a blockchain with different consensus parameters, target audience, or capabilities. Some of them, such as JP. Morgan's Quorum [16] also allow for private transactions. ErisDB has had support for private blockchains well before the Go-Ethereum client MedRec uses. Hyperledger Fabric has not yet reached version 1.0, but is being developed as an enterprise grade smart contract platform based on Node.js and Java.

I am using a private Ethereum blockchain since the Ethereum ecosystem is the most diverse and has the most public scrutiny and support. The Go-Ethereum client is the most used Ethereum client, developed primarily by the Ethereum Foundation. Over time I expect the capabilities offered by this client to approach those offered by other alternatives. Many of these alternative blockchain solutions use the same Ethereum Virtual Machine, providing easier portability between systems.

4.4 Proof of Authority

MedRec uses a Proof of Authority (PoA) blockchain developed as part of the Go-Ethereum client. PoA is used to create private blockchains, where only a federated set of nodes are allowed to make modifications to the shared ledger. The data stored on the blockchain can still be made public outside of the federated set of nodes maintaining the blockchain. As PoA changes the fundamental game theoretic security model of PoW blockchains from a financial incentive to trust based. The intrinsic cost of trust replaces the extrinsic financial cost. The security of such blockchains depends on a choice of federated nodes.

I decided to make providers the authorities on the blockchain. These entities already are entrusted with the safety and security of their patron's medical data. In many countries they are legally required to comply with strict standards around the protection and handling of patient medical records. This makes them ideal custodians of the access rights which govern MedRec. In MedRec medical providers are able to be voted in as voting authorities by the current set of authorities. An authority is able to form blocks and add them to the blockchain. Since the identities of all participants is known, regulation can be done external to the blockchain [17]. The penalties for abusing their abilities as an authority would be severe as they would likely be penalized under HIPAA provision.

Even as authorities, the attack surface for providers remains low. A provider has the ability to validate transactions and add them to the blockchain. Unlike PoW there is no external rate limiting mechanism such as computational power. A properly functioning implementation of a PoA client inhibits its own rate of block generation so the network is not needlessly flooded with data.

Since each transaction is public other providers are able to validate every transaction as well. A provider is incentivised not to generate fraudulent data because they will be easily identified. Furthermore the type of attacks on the system by providers is limited since their only duty is to confirm transactions. Providers cannot impersonate other actors in the system or generate false permissions for their patients as they do not have those private keys. The following describes impact of this choice of blockchain technology on the eight properties of blockchains given above.

4.4.1 Consensus

In MedRec the network needs to come to consensus on which human identities are allowed to access which documents. I defend my use of a blockchain for they are a useful tool for allowing multiple parties to come to agreement about the state of reality. MedRec unifies disparate provider databases using a shared blockchain to store who and how to access the medical data.

4.4.2 Cost of Trust

A limitation of blockchains is they require a tradeoff between efficiency and trust. Blockchain based systems are not inherently more trustworthy or decentralized than the alternatives. However, they allow systems to slide along the axis from complete centralization, with one person running a node validating transactions, to full decentralization, with every network participant running a node and checking each other's contributions. In MedRec I want to leverage the trust that already exists between patients and their medical providers while mitigating the effect a bad provider could have on the system. Using a Proof of Authority blockchains creates a network that is cognizant of the fact that the identities of the maintainers of the blockchain do not need to be anonymous. Furthermore, when they are not anonymous bad actors can be detected and removed. Thus providers are allowed to create blocks instantaneously, instead of after a computationally and time intensive process, increasing efficiency.

4.4.3 Storage Inefficiency

A limitation of blockchains is they require all maintainers to store and reference the entire blockchain. This provides an incentive to minimize the amount of data stored and is one reason I did not use a public blockchain, instead using a permissioned blockchain, unique to medical records. This mitigates the storage costs by restricting the data on chain to a single topic, access permissions. MedRec is configured to only store on chain what is necessary. Additionally, I and other users of MedRec can make predictions about the growth of users and derive the expected computational and storage costs.

Conversely the blockchain provides a mechanism for data sharing between nodes that are not online at the same time. This is necessary for patients to share information with each other without using a centralized institution as a repository of everyone's data. Even having a list of medical providers is not sufficient. A patient's client would have to go through and contact each one periodically to check for new data that had been left for them. I trade the cost of expending this time for the cost of redundantly storing information. The blockchain provides a distributed, yet central place to store information. One especially useful application is providers can all store their ip address in a single location for patients and third parties to query. Instead of querying every provider for a patient's data, a third party queries their local copy of the blockchain for the relevant provider and how to contact them.

Every patient must store some data on-chain. They execute a 9727 byte transaction to create an Agent contract. Then each relationship between a patient and provider requires a 5007 byte transaction. Updates to contracts require 220 bytes. Using these numbers we can make varying estimates on how many bytes need to be stored by every node.

Example 1. Assume 350 million users (approx. US population) who each form relationships with 5 different providers over their lifetime. Every person leaves the contracts as deployed and makes no further update transactions. This gives

$$3.5e^{8} * 9727 + 3.5e^{8} * 5007 * 5 = 12.17e^{12} bytes$$

or 12 Terabytes.

Example 2. Assume 7 million users (approx. Massachusetts state population) who each form relationships with 2 different providers over their lifetime. Every person makes 30 update transactions to add/remove third parties to their Relationship contracts. This gives

$$7e^{6} * 9727 + 7e^{6} * 5007 * 5 + 7e6 * 30 * 220 = 2.895e^{11} bytes$$

or 290 Gigabytes.

4.4.4 The Locus of Trust

There is no singular institution that is uniquely placed to manage access to all medical records. Furthermore I argue such an institution should not be created. Such an institution would be an extremely lucrative target for attack. Already the current landscape possesses a plethora of data silos in the form of individual provider databases lacking a common way of accessing their data. The primary objective is not to build yet another data silo, but to standardize access to all of them. A patient or provider using MedRec who gets hacked does not compromise any more information than they do using the centralized management tools that exist today. Notably when any agent in the system is compromised they only release information about themself and any medical data they were privy to. Additionally, operations on the blockchain are public, allowing providers to identify the actions of themselves, and their patients. In this was the power of the already distributed medical network, and is leveraged to build a more resilient system.

4.4.5 Transparency

Patients can see the full scope of their medical data as well as who they have shared their medical data with across multiple medical providers.

4.4.6 Security and Convenience

MedRec runs as a desktop application but I envision future implementations running on a mobile devices. Although not part of the current design, the blockchain permits applications to make tradeoffs on the axis between security and convenience. On one side is the current version of MedRec where every node is a *full node* which verifies and propagates all transactions and blocks. Along the axis are other varieties of nodes that adjust how much of the blockchain is stored and verified. *Pruned nodes* only keep a fixed number of the latest blocks. Other nodes may choose not to verify transactions, trusting that other nodes will do it. Still other nodes, known as *light nodes*, only download the blockheaders, resulting in massive space savings but relying more on the existence of trusted nodes elsewhere on the network to verify transactions. As MedRec has provider nodes with public identities maintaining the blockchain with full nodes, future applications running on mobile devices can use a light nodes without a detrimental effect to their security [18] [19].

4.4.7 Anonymity

Using the Ethereum blockchain provides pseudonymity. An ethereum account inherently reveals no information about the identity of its owner. But over time a person's transactions can be correlated to identify them [13] [20]. This normally would make an application such as MedRec which requires patient identities to remain anonymous unusable, but I developed a way to efficiently convert pseudonymous identities into truly anonymous ones. Note that users can still choose to identify themselves to others on the network, breaking their own anonymity. More details on this are provided in section 10.

4.5 Bootstrapping

Although they may become decentralized systems, blockchains need to be initially setup by a central institution. The bootstrapping of the PoA blockchain for MedRec was no different. Documentation on the technical capabilities of PoA and the progress of its implementation in go-ethereum remains scant at the time of this writing. At this point the go-ethereum blockchain is still considered a work in progress and in practice getting the blockchain to run initially was a finicky process.

There is a weak point in the decentralized nature of blockchains, which is introducing or *bootstrapping* new users into the network. The previous sections discussed various methods of ensuring consistency, availability, and correctness of data on the blockchain, but left open the question of how new users join the network. Even with a decentralized network, blockchains, including MedRec, generally retain a centralized aspect to the network in the form of bootnodes. A bootnode is a node on the network with the sole purpose of matchmaking new users with nodes already on the network. They are akin to DNS root authorities and Bittorrent trackers. These nodes are hardcoded into the MedRec client. When MedRec is operational they should at minimum be set to a few well known public healthcare providers on the network and at maximum all providers. Only one of these providers needs to be non-malicious in order for a user to safely join the network. A completely malicious set of bootnodes would prevent anyone from using the network.

Past research on Bitcoin [21] and Ethereum [22] has analyzed the practicality of an attacker assuming control over a node's connection to the rest of the network. These attacks are generally called eclipse attacks. The effect is while the victim believes they are well connected to the network, all peering connections from the victim's node lead directly to an attacker who is able to manipulate their interaction with the blockchain. For example by delaying transactions. Using a PoA blockchain adds another mechanism for mitigation of these attacks. One way is by requiring all nodes to maintain a connection to at least one provider node at all times. The current set of provider nodes can be hardcoded into each version of the MedRec software as the nodes used for bootstrapping.

5 Implementation

MedRec comprises four components: Ethereum PoA Blockchain, The Smart Contracts, The User Client, and the Database Manager.

6 Ethereum PoA Blockchain

For demonstration and research purposes, I setup a blockchain spanning three different computers on Amazon AWS. Each computer supplied a provider node, and each node was initialized as a signer in the genesis block of a PoA blockchain.

Signers in MedRec also offer a service known as an ether faucet. Every transaction on the network requires ether. The fulfillment service of requests for ether is the ether faucet. but for user convenience this is obfuscated as much as possible to MedRec users. Every time a user makes a transaction, MedRec automatically makes a request to a provider for enough ether for the transaction. This notably and visibly manifests itself when a patient is creating their Agent contract or when a provider is proposing themself to be a signer. In both cases the User Client includes a *sponsor* field where a provider's unique id must be placed. That provider is used to obtain enough ether to proceed. Following that, the sponsor of a patient's transactions can be inferred and does not need to be requested explicitly. Providers who are accepted to be signers automatically generate eth as their node validates blocks.

7 Smart Contracts

There are three main types of smart contracts in MedRec. These are written in Solidity and run on my PoA blockchain. Additionally I implemented a couple of other contracts, derived from these, as a demonstration of the extensibility and applicability of the Solidity language.

7.1 Agent

The Agent-and its derivative the AgentGroup-contracts represent actors on MedRec. They are analogous to the *SummaryContract* in MedRec v1. Individual patients interact with the system using the Agent contracts as a proxy while multiple actors can form an AgentGroup. These contracts hold data about the entities they represent. For patients, the Agent contract stores a list of Patient-Provider relationship contracts. The AgentGroup is provided as a convenience to give permissions to a large set of agents at once. For example, a group of emergency rooms from different providers may join into one AgentGroup allowing a patient to give them all access to basic blood work data in one step. The members of the AgentGroup can change without any further action on the part of the patient. Smart contracts are extensible and customizable, exact details such as how new members are added to an AgentGroup can be established on a case by case basis by different groups.

In MedRec Agent contracts are only created by patients, they are not needed for providers, third parties, or other actors who do not need to modify access to their own personal data. Agent contracts exist primarily as a recovery mechanism. Pointers to all of a patient's provider relationships are kept in one place on the blockchain so that they can be found if the patient has to recover their account from the recovery seed. The recovery seed generates a private/public keypair, which derives an Agent contract, which derives pointers to the patient's relationships. Using Agent contracts also provides the ability to store small amounts of data on the blockchain associated with a particular address-this will likely prove useful in future work on MedRec.

7.2 Relationship

This contract represents the relationship between all parties who have access to a particular set of patient data. Each contract has exactly one patient and one provider associated with it. The contract also contains an unbounded list of *viewers*, third parties who also have access to the patient's data. Viewers share an ethereum account with the patient who then adds it the contract. The relationship between patient, provider, and viewer is stored on the blockchain. Additionally, off the blockchain providers store more granular access permissions. Permissions for each viewer are stored in a small database on the provider's computer. Via the MedRec interface patients can customize permissions, eg. Permitting family members to exclusively view information about a recent surgery.

A possible place of confusion is why I decided to segregate permissions from the other data stored on the blockchain. Placing all permission details on the blockchain was a more convenient and cohesive storage- all relationship data would be in one place. The initial system design operated this way for this reason. When evaluating the scalability of the system I realized permission data could significantly affect the storage needed to maintain a medrec node. Putting information on the blockchain requires it to be stored by every full node in the system. Fine grained management of permissions stored on the blockchain would introduce significantly more transactions per second and bytes to be processed and stored by every node.

Having the ethereum addresses of all parties be public allows them to send signed messages to each other and confirm participation in the same relationship. The relationship contracts also provides a mechanism for the patient and viewers to identify which provider in the MedRec network posses some of the patient's data, instead of querying all of them for updates. Once the provider is identified, however, then it's possible for the requester to retrieve information about specific facets of a patient's medical history. Permissions are not stored on the blockchain because they do not have to be.

MedRec allows patients to have complete control over access to their medical records. Access permissions can be specified to start on a certain day and last for a certain amount of days. These specifications are specified on the granularity level of both per viewer and per permission. This can be used, for example, to provide 1 day prescription history access to a pharmacy. Data is retrieved directly from the provider's database, providing credibility. Meanwhile the patient can trust that after 24 hours the pharmacy will be locked from examining future prescription results.

7.3 Agent Registry

The last major contract used by MedRec is the Agent Registry. The Agent Registry has a single instantiation which is referred to by every node on MedRec. One capability it provides is a single reference point for providers to put their name and ip address. The UserClient and DatabaseManager leverage this to lookup a provider via their ethereum account and retrieve the ip address for direct communication. Additionally, the Agent Registry contains the logic which manages the authorities of the blockchain. Inherently, the PoA implementation we use has a simple 50% activation threshold for adding or removing authorities. Creating a smart contract abstraction over that permits for a fully customizable method of consensus. The current implementation of MedRec retains the 50% threshold, A potential future upgrade to the Agent Registry might place a small number of large and well known providers as *essential* authorities which cannot be voted out of the network except by each other.

8 User Client

This is the user facing component of the application. All patients, providers, researchers, etc. will interact with MedRec through this interface.

One major design distinction between MedRec v1.0 and MedRec v2.0 is the move from a web to desktop application. This is necessitated by a change in the way user data is stored in the system. The rationale for this change comes from the security risks posed by MedRec 1.0's implementation, which hashed user usernames/password combinations together and stored them on the blockchain. Furthermore, the username/password directly, deterministically derived a private key for signing transactions in the system. Though a valid method of storage, recent work has shown that these hashed credentials are crackable [23]. Users do not pick sufficiently secure passwords to prevent their private key from being determined by brute force.

As MedRec is a distributed network, the user credentials on a central server, as other webapps have the option to do. Thus, I propose that the user stores their credentials locally. Whilst the initial load on the user is somewhat greater— requiring the download and installation of an app, rather than a pure web interface- it allows us much greater freedom in the distribution of notifications, and means that the user has a local means of storing and browsing their own, up-to-date records. I would also propose that, whilst a user might be reluctant to install an app on their computer, they are likely to have fewer issues with a phone equivalent, and this could provide the more successful platform for distributing users' data. This would also provide offline access to data without the user having to have downloaded the files of their own accord, which could be invaluable in emergency situations, where internet access may not be available.

We chose the Electron API to develop the front-end. Electron is a cross-platform framework that uses web languages (JS, HTML, CSS) to build native applications. Effectively Election ports web applications to a desktop environment. The Electron frontend interfaces with both local and remote servers via websockets. The local server holds the user's private keys and other credentials, while remote servers are hosted by providers that the user has a relationship with.

I (along with my co-researchers) have written a desktop application which runs locally and

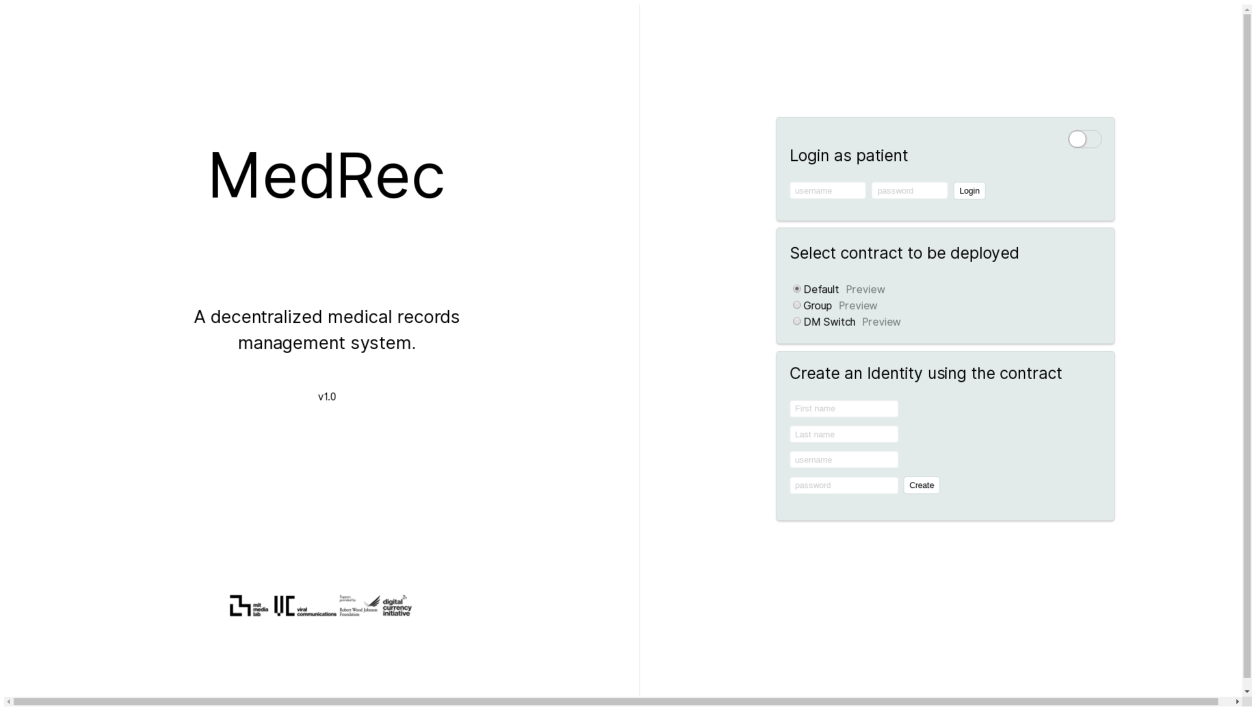
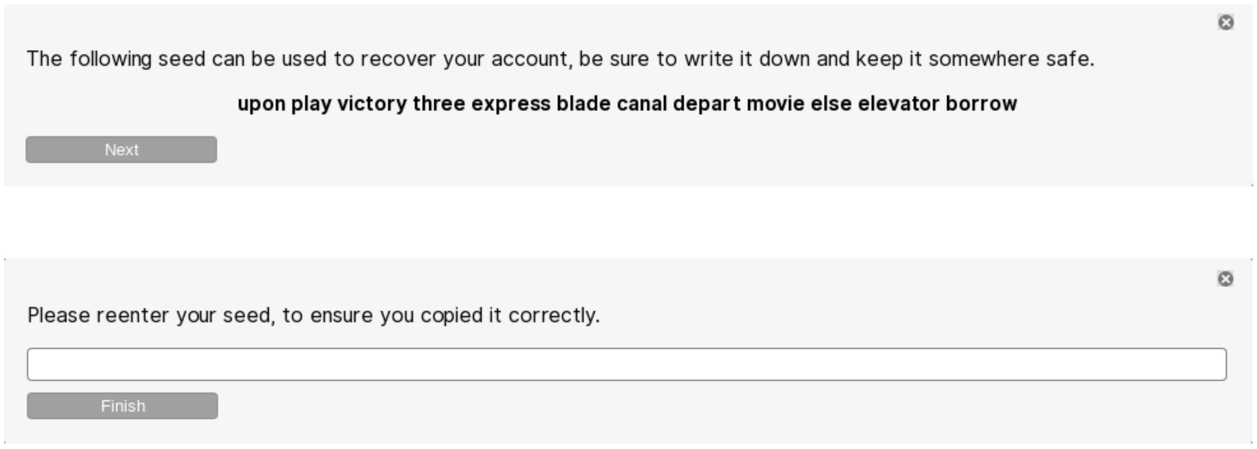


Figure 1: MedRec welcome page

make queries to the other portions of the system, the Ethereum Blockchain and the Database Manager. The majority of the visual styling and user experience was done by Kallirroi. MedRec uses a Proof of Authority (PoA) Blockchain to allow participants in the system to come to consensus about the state of the network. Unlike the first version of MedRec, which was based on PoW, no knowledge of how to transact using cryptocurrency is required for users to participate in the system. Users will interact with a familiar web interface while behind the scenes, the application logic translates their commands into calls to provider databases and smart contracts. Figure 1 is a screenshot of the first screen a user sees when they open the application. Users are invited to create an account and login using a username/password combination as they might do for a website.

By making a downloadable application instead of a web application I make a tradeoff in favor of user security at the expense of convenience. Users have to possess more storage space so they can hold the medical records that they access. They also need enough space to house the MedRec application, including the blockchain with everyone's access permissions. With these costs come the benefit of the ability to access medical records offline. Even if a patient's provider becomes


The following seed can be used to recover your account, be sure to write it down and keep it somewhere safe.

upon play victory three express blade canal depart movie else elevator borrow

Please reenter your seed, to ensure you copied it correctly.

Figure 2: The user is prompted to record their recovery seed

unresponsive, the patient can still control their own medical data. I consider this to be a necessity for any system that claims to be decentralized.

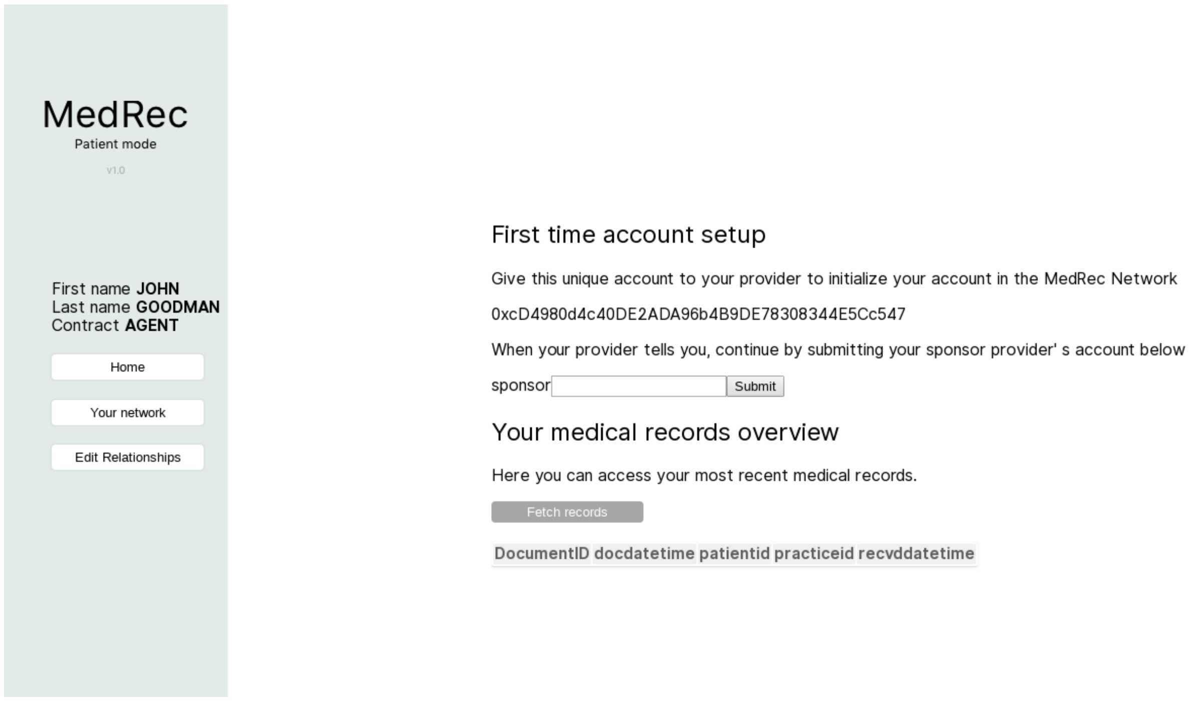
MedRec uses a local database to cache some information from the blockchain and store user settings. Users can create an account and then login with a username and password via their MedRec instance. This login mechanism is not necessary, all a person really requires to interact with the system is their 64 byte private key. The username password abstraction is to enhance the usability and familiarity of the interface. This login will only work on the computer the user account was created on as it's only stored in the local database. Multiple users can create accounts on and use one instance of MedRec. Passwords are salted and hashed before storage so that other users on the same system cannot scry others' password. I considered three different algorithms for hashing passwords: PBKDF2, bcrypt, and scrypt. I chose Scrypt even though it's the newest of the algorithms because it has been designed to be more resistant to brute force attacks [24]. Unlike the other two, Scrypt is memory hard as well as computationally hard for an attacker to brute force. Due to this Scrypt is also the hash algorithm of choice for multiple popular cryptocurrencies including Dogecoin and Litecoin.

Account recovery is done via recovery seeds [25]. When a user creates an account they not only generate a private key, but a 12 word recovery phrase or seed. This seed should be written down and kept secret similar to a passport or birth certificate. When a user signs in with this key they can execute all functions on their account, including generating a new private key and recovery seed. The recovery seed allows a user to manage their account in the case they forget their password or their primary computer is damaged. A hash of the recovery seed is stored on the blockchain with a tag associating it with the user's account. The seed does not need to be salted since they are verifiably unique to each private key. When a user logs into a MedRec instance with their recovery seed the software can query the blockchain to check which account the seed belongs to. This makes MedRec truly decentralized. Even though access is the easiest from their primary computer, a user can easily move to another device and still securely access their medical data.

Figure 2 illustrates the modal windows shown to a user while creating their account. Unless the user is able to repeat their seed in the prompt they will not be able to continue through account creation. This makes it impossible for the user to just click through the seed generation modal and forces them to make a conscious effort to save (or not to save) their seed.

This model is completely different from that used in the previous iteration of MedRec. MedRec v1.0 hashed together the username and password of users and placed them on the blockchain in a registry contract. This provides universal access to a user's MedRec account, similar to my use of recovery seeds. The key difference is that the user is effectively setting their own seed. This method of key derivation from a user chosen seed is called a brainwallet. Past work presented at DEFCON [23] has shown brainwallets to be a completely unsafe for practical wallet generation. Inherently humans do not provide enough entropy to create a secure key from a user chosen phrase. The recovery seeds generated by MedRec are 12 words that are simply an alternate encoding and checksum for a private key. No information nor security is lost by using them, with the improvement that they are easier to write down and remember. While on a personal device the user can use their username and password. If the password is forgotten, the recovery seed can be used to restore the account. Additionally, the move to local accounts increases the barrier to entry for attacks against users. There is no capability for running a general attack against all the users of the entire system, an attacker must run target attacks to obtain the salted and hashed password for each user they want to compromise and run brute force attacks to decrypt that data.

8.1 Patient Interface



First name **JOHN**
Last name **GOODMAN**
Contract **AGENT**

First time account setup

Give this unique account to your provider to initialize your account in the MedRec Network

0xcD4980d4c40DE2ADA96b4B9DE78308344E5Cc547

When your provider tells you, continue by submitting your sponsor provider' s account below

sponsor

Your medical records overview

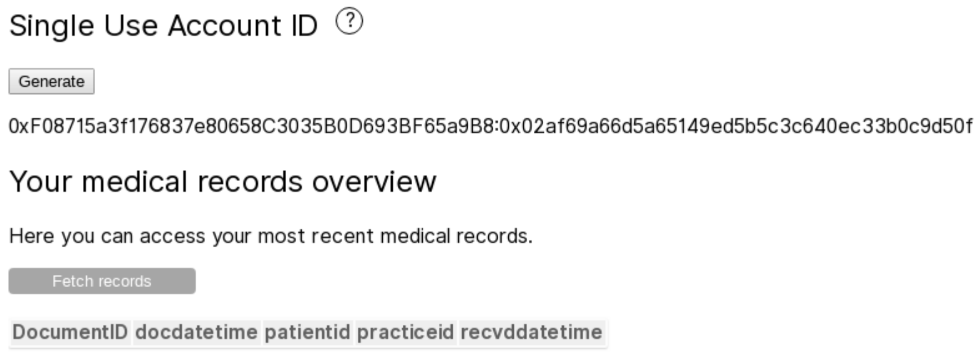
Here you can access your most recent medical records.

DocumentID	docdatetime	patientid	practiceid	recvddatetime

Figure 3: Patient post login screen

The patient homepage generates unique account ids for relationships and gives a summary of the medical records a patient possesses access to. The homepage interface takes two slightly different forms dependent on whether the patient has created an Agent contract yet.

Tee first view is shown in Figure 3. The process for the patient to finish joining the MedRec network is as follows. The provider creates an entry in their local database with a unique identifier for the patient. Then, they send that identifier to the patient via some communication mechanism, like email. This identifier could be something as simple as a name and date of birth. The patient sends this information to their provider along with their ethereum account. The provider's MedRec instance handles verifying the unique identifier and storing the patients ethereum account. This account is used to sign future patient requests for data. The patient's MedRec application also requests enough ether from the provider to make an Agent contract and proceeds to make the Agent contract. The reference implementation of MedRec developed by the Viral Communications Lab has a version of this more suitable for tests and demos. In this version the patient shares their



Single Use Account ID

0xF08715a3f176837e80658C3035B0D693BF65a9B8:0x02af69a66d5a65149ed5b5c3c640ec33b0c9d50f

Your medical records overview

Here you can access your most recent medical records.

DocumentID	docdatetime	patientid	practiceid	recvddatetime

Figure 4: Returning patient login screen

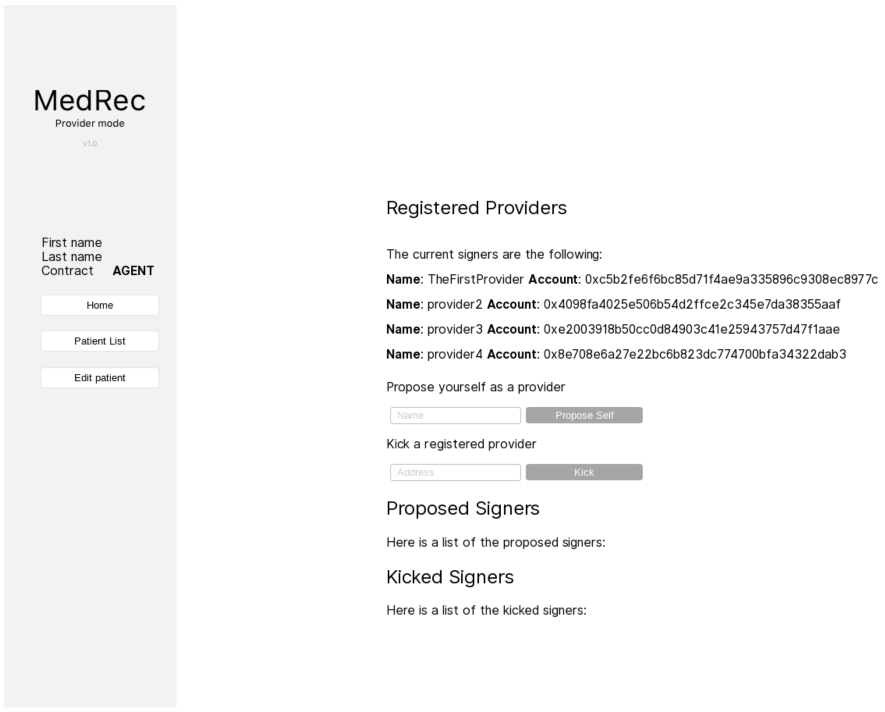
ethereum account with their provider so the latter can link it with the patient's unique identifier in the provider database. After this link is established on the provider side the patient can continue on to create their Agent contract. After entering the provider account, the app looks up the ip address of the provider using data stored in the smart contracts. Then it directly sends a request to the provider for enough ether to create the Agent contract. It is impractical to expect patients to have this kind of back and forth interaction in a production system, however. A production implementation would have the provider create the database link first then send the patient their unique id via email. The provider's account may also be provided via email or via an in app search mechanism.

The second view appears when MedRec detects an Agent contract deployed to the blockchain for the user's ethereum account. Clicking the generate button creates a new unique identifier for the account holder to share. This identifier is generated by third parties (which could be patients) and given to patients to include in a relationship contract. Inclusion in a Relationship contract permits a third party to view the medical data of the associated patient stored by the associated provider. The identifier is a concatenation a single use ethereum public key and the patient's primary ethereum account. When a third party wishes to be privy to data in a patient relationship, they share this string of characters with a patient. They use the first half to 1) encrypt the identity of their provider and store it in the Relationship contract where the third party can reference it and 2) store the single use pubkey in the Relationship contract so the provider knows to accept signed requests for data from that pubkey. When a patient wishes to create a new relationship with a provider they share this string with the provider. The provider then saves the second half of the string, the primary ethereum account. This account is associated with both the patient's Agent contract and Relationship contracts and allows the provider to verify the patient in future requests.

The UserClient also provides an interface for editing and viewing relationships. There are three levels of edits: provider, viewers, and permissions. The top level is creating Relationship contracts between the patient and different providers. The middle level is sending transactions adding third parties as viewers to a particular relationship contract. The interface actually simplifies some of the functionality available in the Relationship contract. The Relationship contract allows for multiple viewers to be grouped together and given permissions as a unit. The lowest level does not require any interaction with the smart contracts. Permissions for viewers are only stored at the provider so at the bottom level MedRec makes updates by contacting the provider directly.

8.2 Provider Interface

Providers can use their interface to connect with patients on the MedRec network and administrate signers for the blockchain. Figure 6 shows the homepage for a logged in provider. The homepage lists all the current signers stored in the smart contracts. If the provider is not yet a signer they can



Registered Providers

The current signers are the following:

Name: TheFirstProvider **Account**: 0xc5b2fe6f6bc85d71f4ae9a335896c9308ec8977c

Name: provider2 **Account**: 0x4098fa4025e506b54d2ffce2c345e7da38355aaf

Name: provider3 **Account**: 0xe2003918b50cc0d84903c41e25943757d47f1aae

Name: provider4 **Account**: 0x8e708e6a27e22bc6b823dc774700bfa34322dab3

Propose yourself as a provider

Kick a registered provider

Proposed Signers

Here is a list of the proposed signers:

Kicked Signers

Here is a list of the kicked signers:

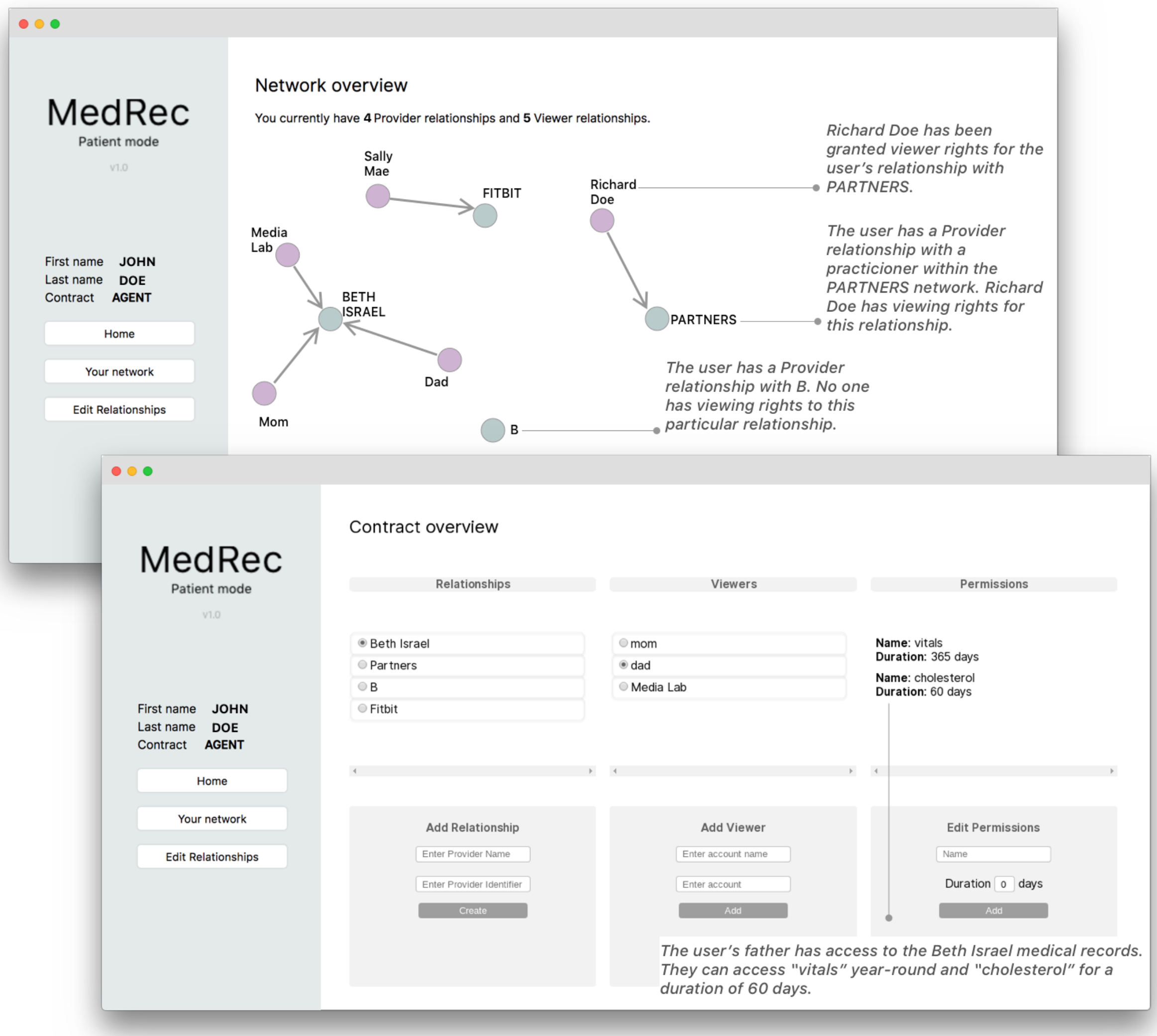
Figure 5: Provider post login screen

propose themself as one as well as view other open proposals. If they are a signer they can create and vote on proposals to kick a current signer.

Figure 7 shows the provider's interface for adding patient accounts. The patient account is linked to whatever the unique id for the patient is inside the provider's database. This form of this unique id will depend upon the provider. As a proof of concept the interface also shows an example list of patients that have already been added.

8.3 Language Choice

I developed the User Client as a desktop application to gain access to the portability and rich design options available to websites. Since the User Client runs in a browser it can more easily be form fitted to a variety of displays, namely desktop monitors and mobile phones. For a production build, I use Electron to package the app into an extremely lightweight version of the Chromium browser. This converts the webapp into a more familiar looking desktop application. Standard web



First name JOHN
Last name DOE
Contract AGENT

You currently have **4** Provider relationships and **5** Viewer relationships.

Richard Doe has been granted viewer rights for the user's relationship with PARTNERS.

The user has a Provider relationship with a practicioner within the PARTNERS network. Richard Doe has viewing rights for this relationship.

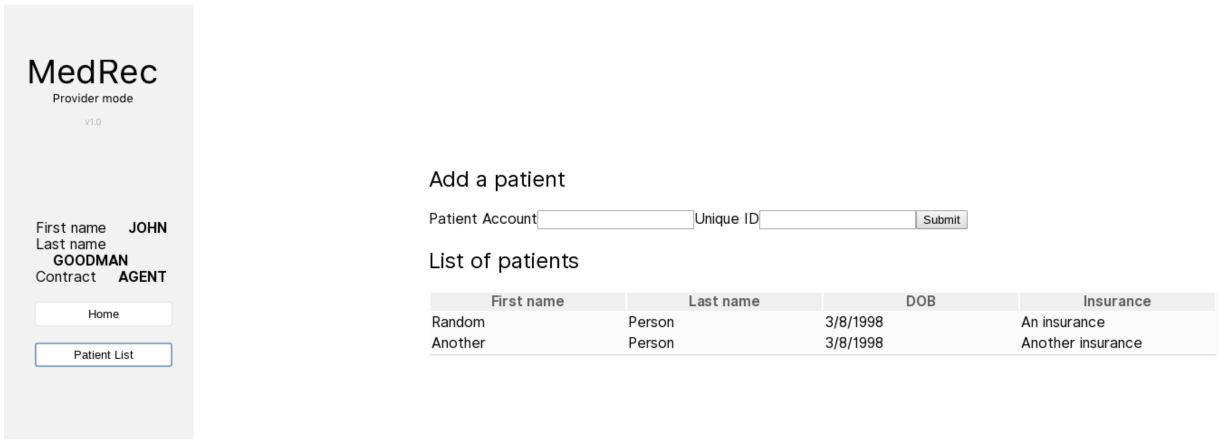
The user has a Provider relationship with B. No one has viewing rights to this particular relationship.

Relationships	Viewers	Permissions
● Beth Israel	○ mom	**Name:** vitals **Duration:** 365 days
○ Partners	● dad	**Name:** cholesterol **Duration:** 60 days
○ B	○ Media Lab	
○ Fitbit		

Duration 0 days

The user's father has access to the Beth Israel medical records. They can access "vitals" year-round and "cholesterol" for a duration of 60 days.

Figure 6: Top window: Patients are presented with a visualization of their relationships. Bottom window: The page patients use to edit their relationships.



First name JOHN
Last name GOODMAN
Contract AGENT

First name	Last name	DOB	Insurance
Random	Person	3/8/1998	An insurance
Another	Person	3/8/1998	Another insurance

Figure 7: Provider patient addition screen

components, JavaScript, React, HTML, and Sass were used to construct the application.

8.4 RPC

Remote Procedure Calls (RPC) are commands issued by one computer to invoke functions on another. The protocol is transport agnostic, and can be applied to TCP, HTTP, Websockets, and other protocols for sending data over the internet. MedRec uses RPC in two ways, between the client and a local patient database, and between the client and a remote provider database.

Locally RPC calls are used to interface between portions of the User Client written in JavaScript and the DatabaseManager written in GoLang. The DatabaseManager is essential for writing user data to the host filesystem. This includes usernames, passwords, and recovery seeds.

The User Client also contacts other, remote instances of MedRec. These calls are used to execute operations on provider nodes. One such operation is a request from the patient to retrieve all medical records held by the provider. Making a call from the User Client may directly execute a function on a server owned by a different MedRec user. For this reason I had to be careful defining the capabilities achieved by these calls and ensure authentication is built into requests. More about the available functions can be found in the section in the following section on the Database Manager..

9 Database Manager

The Database Manager component of MedRec interacts with the underlying filesystem and facillitates communication between MedRec nodes. It is primarily written in GoLang, although some interactions with the Ethereum network require calling out to helper scripts written in Javascript. Agnes was essential in implementing user authentication and interfacing between MedRec and an underlying database of medical records. In the following subsections I describe the components of the Database Manager.

9.1 Middleware

Every RPC request received by the Database Manager is modified by two or three layers of middleware before it gets processed by the destination function. The whitelist layer uses the request origin ip address to reject unauthorized requests. This first line of defense prevents remote computers from calling sensitive functions that access personal data on a MedRec node. Another middleware adds a header which allows browsers (via the User Client) to exchange data with the Database Manager. An additional middleware outputs basic information about each received request to a terminal console.

9.2 Local RPC

Certain functions are only intended to be called locally from a User Client on the same machine running the Database Manager. Most of these calls deal with user account creation and management. All except the last are called from the patient side of the application. MedRec makes use of the following locally restricted calls:

- *SetWalletPassword*(Password)
  - This function sets a global variable in the Database Manager with the password protecting the user's Ethereum private key. This variable is stored in memory until the application closes.

  - MedRec calls this function when a user logs in. Without holding the password in memory, a provider would have to manually enter in their password to unlock their account hundreds of times per hour as patients request ether from the provider faucet as described in Section 6.

- *GetUsernames*()
  - This function returns a list of all the usernames stored locally on the machine.
  - *GetUsernames* was used to display a list of usernames on the User Client login page. The user could select their username from the list without entering it in manually.

- *GetUserDetails*(Username)
  - This function takes a username and returns the user's First and Last name.
  - MedRec uses this function after login to welcome the user with their name.

- *NewUser*(Firstname, Lastname, Username, Password, Seed)
  - This function saves a Username as a key to the other arguments. The Password is salted and hashed before saving. The Seed is encrypted with the password before saving. Firstname and Lastname we deem to be about as revealing as the Username so they are left unencrypted.

- *GetSeed*(Username, Password)
  - This function decrypts the appropriate user's seed and returns it. An error is returned upon decryption failure.
  -

- *DeleteUser*(Username, Password)
  - This function deletes a user.

- *SaveKeystore*(Keystore, Username)
  - This function saves a serialized version of the public and private keys for a user.
  - MedRec derives multiple public keys from a single private key in a deterministic way. When a new key is generated the updated keystore is saved. All data in the keystore is encrypted with the user's password.
- *GetKeystore*(Username)
  - This function retrieves the saved keystore.
  - MedRec calls this function at login to move the keystore from storage on the filesystem to memory in the browser.
- *AddAccount*(UniqueID, Account, Username, Password)
  - This function correlates an patient's ethereum account with their unique ID in the provider's database. It also generates a unique ethereum account for the provider to use for their Relationship contract with the patient. The generated account is called a delegate account and is described in greater detail in a succeeding section.
  - This function is used by providers. It creates a mapping between patient ethereum accounts and the unique id used for the patient in a provider database. When a patient requests to use the ether faucet the Database Manager checks to see if an entry exists for them in them mapping from account to unique ID.

AddAccount

9.3 Remote RPC

Each of these calls are made between MedRec Instances- either from patient to provider or provider to provider. They are submitted with both a timestamp and a signature of the timestamp. The timestamp is checked to ensure it is no more than 10 seconds in the past to prevent duplicate requests. The signature and timestamp are used to derive the signing account. Depending on the function this account is either used to verify identity as a patient or provider. If verification succeeds the appropriate information is retrieved or the processing is done.

- *GetProviderAccount*()
  - This function returns the unique provider account that has been generated for particular patient.
  - When a patient is creating a new Relationship contract they need to know what provider account to put in the smart contract. During this process they enter the provider's main account. MedRec automatically uses the AgentRegistry to lookup the provider ip and send a signed request to their provider calling this function.
- *ChangeAccount*(Account)
  - This function can be called by a patient to change the Ethereum account the provider has on file for them in case their account is compromised.
- *PatientDocuments*(PatientID)
  - This function retrieves from a provider all available data stored for a particular patient. What data is available is chosen by the provider. Third parties who have access to a patient's data may also call this function.
- *PatientFaucet*(Account)
  - This function is called by patients to request more ether from their provider. This ether can then be used in transactions editing Relationship contracts. The caller also specifies via the arguments the provider account they are requesting ether from. As a provider may have hundreds of accounts, of which only one has ether this expedites the process.

  - As described in a succeeding section on anonymity, this function does not fulfil the ether request directly. Instead it randomly selects a provider from the current set of signers and passes the request on to them using the *ProviderFaucet* function.

- *ProviderFaucet*(Account)
  - This function verifies the caller is a provider and then sends ether to the account specified in the function arguments.
  - This method is only and automatically called after a patient requests ether using the *PatientFaucet* function.

- *AddPermission*(AgentID, ViewerGroup, Name, StartTime, DurationDays)
  - This function adds a new permission. The patient is given by AgentID; the permission benefactor by ViewerGroup; and the permission name, start time, and duration by the corresponding arguments.

- *RemovePermission*(AgentID, ViewerGroup, Index)
  - This function deletes a permission.

- *SetPermissionDuration*(AgentID, ViewerGroup, Name, DurationDays)
  - This function updates a permission duration.

- *SetPermissionStartTime*(AgentID, ViewerGroup, Name, StartTime)
  - This function updates a permission start time.

- *GetPermissions*(AgentID, ViewerGroup)
  - This function returns an array of all the permissions for an AgentID, ViewerGroup combination. The index of a permission in the array is used during permission deletion.

- *CheckPermission*(AgentID, ViewerGroup, Index)

- This is a convenience function returns a boolean answering the question of whether the permission specified using the arguments exists and has not expired.

10 Anonymity

Patients have some anonymity in the system as their identity is only known to those organizations with whom they share their identity with. Yet this is still not enough protection, as an unintended side effect of using a single smart contract to represent a patient (the Agent contract) allows anyone who discovers a patient's Agent contract to decipher all other relationships with providers. One example of a case this would be harmful is when a patient shares portions of their medical records with both their employer and a gynecologist. The employer could detect this association and find a pretense to discharge their employee to avoid paying health insurance.

We solve this in two steps. The first step is disassociating each patient identity from provider identities. The solution to this is simple, each provider makes a new ethereum account for each patient provider relationship. We call these patient specific accounts *delegate accounts*. In practice these accounts would be an extra column of data stored along with a patient's name in a database. This method allows a patient to have public relationships without revealing who the individual members of those relationships are.

The second step is more complex and it solves the problem with metadata that the first step leaves open. Even though a patient is not interacting directly with their provider's main account, that main account is still being used to provide the patient with ether for their transactions. Over time this correlation could be used to associate patient with providers. At the same time there needs to be some means of correlation allowed so that providers can track how much ether has been sent to their patients and detect abuse. This has a solution. Instead of fulfilling patient ether requests themself, a provider asks another provider to do it for them. Remember that in MedRec providers also act as authorities and receive ether in exchange for validating blocks. This gives them effectively an infinite amount of ether to share with patients. Providers can track ether requests by their patients, but this information is not leaked onto the public blockchain.

A critique of this design is it does not use zero-knowledge proofs to guarantee the privacy of a patient's identity. The primary advancement of zero-knowledge proofs in the field of crypto-currency is the ability to exchange coins between two parties without revealing to a third party that the transaction ever happened. This is done using zk-SNARKS [26]. One proposal for an application in MedRec is a form of zero-knowledge proof, each patient encrypts information about the viewers of their medical relationships so a viewer can prove they have access to data from a medical relationship without knowing who either the patient or provider are. However, zeroknow-ledge proofs have their own limitations which we would like to avoid. One implementation in the cryptocurrency ZCash [27] utilizes a trusted setup phase. Although the ZCash team took extreme precautions to insure this phase was not compromised, the fact remains that if someone hijacked the secret keys used in this phase they could generate arbitrary amounts of money without any-one knowing. Analogously in MedRec a person could give themself permissions to read arbitrary patient records. Additionally creating a ZK proof is time and memory consuming, with the latest implementation of of ZCash requiring 7 seconds and 40MB of RAM to prove [28]. Again, this does not completely rule out the use of zk-SNARKs, but leaves open the question of is there an alternative.

A more recent advancement by Narula et. al is in zkLedger [29] which also uses zk-SNARKs for privacy preserving transactions. zkLedger does not require a trusted setup phase. The drawback is transaction creation and verification times are dependent upon the number of nodes in the net-work. Creation and verification time of transactions scales linearly with nodes. While zkLedger requires approximately 90ms for 4 participants, it requires 400 for 20 participants. zkLedger achieves this by requiring every transaction to encrypt and include entries for every participant on the network, not just the intended recipient for the transaction. For a system like MedRec which is designed to scale to millions of nodes, the approach used by zkLedger is clearly infeasible.

11 Human Meaningful Names

There are three properties of a naming system which can be difficult to get simultaneously- Human Meaningful, Secure/Unique, Decentralized. Zooko's Triangle is a conjecture that it's impossible to achieve all three. Although that conjecture has since been proven false, achieving these remains complicated in practice. In MedRec I balance between the three in two different ways.

It's generally convenient for signers to be able to easily and uniquely identify each other with a human readable name. To achieve this, new signers are allowed to propose themselves to the list with a name of their choosing. Logic in the smart contract managing the proposals prevents multiple signers from choosing the same name. Therefore I have human meaningful and unique names, in a pseudo-decentralized system.

I also allow patients to associate names with the providers and other entities they interact with via MedRec. These names are human meaningful, but remain linked to the entity's public account, so they remain unique even if the user chooses to refer to two people by the same name. This naming is completely centralized, it's only meant for one user to keep track of their own relationships.

12 Handling Sensitive Data

One aspect of the system that I hope to integrate is a fluid handling of sensitive data that is appropriate to the context for which it is required. For example, if you are in receipt of a prescription for methadone, you may well not want all your care providers to access this information, though it will be essential for some care providers to have access to this data, especially as it would prevent double prescriptions across disparate healthcare providers.

Resolving this issue requires the use of a flexible 'access rights' system, where different actors in MedRec are afforded different viewing permissions of a person's data. For example, a patient should maintain the right to withhold information about their methadone prescription, unless they are in a situation where that is vital knowledge to the actor administering care.

A patient can specify on the level of granularity of a single viewer in a particular relationship what medical data that viewer is entitled to access. This permission can be specified to start on a particular day and last for a specified number of days. This information is stored publicly accessible on the blockchain inside Relationship contracts. As described in an earlier section, a patient has one Relationship contract per provider, with multiple viewers listed inside. This contract also contains information about which permissions each viewer possesses. This information is public so there is minimal cost to read it. When a viewer makes a request to read information from a provider the provider can easily compare the permissions on the data that the viewer is requesting to the list of permissions afforded on the blockchain.

Scalably managing custom-defined permissions may be an issue on the blockchain. Providers may potentially be using an unbounded set of permissions. One of the assumptions of MedRec is that there is no there is not single standard for how medical data is stored across providers.

- General Practitioner — May view all of your prescribed medication, referrals and medical history.

  Emergency Room — May view vital information that could affect emergency treatment: blood type, allergies, currently prescribed medication. This

  Pharmacist — Before prescribing over-the-counter drugs, access to allergy-type information is invaluable in a pharmacy setting. In addition, there may be additional flags for medicines already prescribed that would clash with possible prescriptions. This could be optional, but might help a great deal in cases where a patient would otherwise be given wrong medication, or possibly prescribed the same thing twice.

13 Message Authentication

The same signing mechanism used for transactions in MedRec can similarly be utilized for signing arbitrary messages. In MedRec all messages to providers must be signed. When a patient or viewer is requesting access to data the provider can check that the public address associated with the signature is authorized to view the documents they are requesting.

14 Delegate Accounts

To preserve the privacy of identities in the system I propose and implement a layer of abstraction called delegate accounts for relationships. This idea is an extension of the property of blockchain technologies that normally makes them pseudonymous. In MedRec providers and third parties automatically generate a new ethereum account for use with each patient's relationship contract. This guarantees that even though relationship contracts are public the system inherently reveals no information about their relationships. Similarly if two users both have the same provider the system does not reveal this to either user.

The patient only has to use one private/public keypair, while providers and third parties must use a separate keypair for each patient they manage data for. Another advantage of using a unique keypair for each patient is providers can sign messages sent to their patients just as patients sign each message sent to their provider. A requirement of the system is to both allow for limitless key generation, while still allowing the user to recover all their keypairs from a single recovery seed. I do this using Hierarchical Deterministic (HD) Wallets, a wallet standard that does exactly this.

Implementing delegate accounts does not require implementing delegate Agent contracts. These accounts do not initiate any transactions on the blockchain. A patient includes provider and third party delegate accounts into Relationship contracts, but those delegate accounts do not create or otherwise interact with smart contracts. They are only used for signing transactions between nodes on the network off-chain. For example, when a third party exercises their right to view a patient's medical data, as defined in some Relationship contract, they sign a message to a provider using the same delegate account as written inside the Relationship contract. Using delegate accounts still leaves open another form of attack, which is discussed in the next section.

If the provider account inside a Relationship contract is only a random string of characters then patients need another way of sharing the actual identity of the provider with viewers. I would like to do this in a way that does not require both parties to be online at the same time, the use of a centralized coordination server, or querying every provider in the network to see if they match the identity in the relationship. My solution is to encrypt the true identity of the provider and store it inside the relationship contract. This requires encrypting the main provider account once for each viewer and storing that encrypted string in the Relationship contract.

15 Processing Transactions

Another aspect of using an Ethereum based system is that every operation on smart contracts requires some ether to run. To be able to modify their contracts patients need to first obtain ether from a provider. Participants do this by making a call to their provider asking for enough ether to do an operation.

The system as defined so far leaves open an avenue for'potential corroboration between transactions included by providers. If a provider only includes transactions from or relating to, their own patients in blocks, then it's possible for anyone to query the blockchain for a particular address and identify which providers the address is associated with. My solution to this, is twofold. Providers do not create their own value transfer transactions. Instead they verify the requester is a patient and then randomly select a provider (which may be themself) to fulfill the ether request. Transactions from patients are distributed to all providers for any one of them to include in a block. This is inherent in the way Ethereum processes transactions. In this way both patients and providers are not able to be traced simply by reading the blockchain.

16 Notifications

By using a open architecture for smart contracts, MedRec provides a mechanism for feature extension. One example is notifications. All providers are publicly listed with a unique ip address. A future application could use this list to allow Reasearchers to communicate directly with hospitals to inform them of new drugs and viral outbreaks. Hospitals can then push this information to all their patients. Patients would be able to opt in to receive real time information about outbreaks from hospitals and research institutions.

17 Acknowledgements

I thank my supervisor, the venerable Dr. Andrew Lippman. I also thank the other people directly working with me on this project in the Media Lab Viral Communications group, Kallirroi Retzepi and Agnes Fury Cameron. I thank my parents Jacqueline and Zacharia for pushing me to complete the Meng program, and my dad for doing the first proofread. I have great appreciation for the enthusiastic intern Harry Minsky who did the final proofread of this paper. I would also like to thank the elusive yet revered inventor(s) of Bitcoin, Satoshi Nakamoto. Without their contribution, who knows where I would be now.

References

[1] Mandl, K. D. (2001). Public standards and patients control: How to keep electronic medical records accessible but private Commentary: Open approaches to electronic patient records Commentary: A patients viewpoint. Bmj, 322(7281), 283-287. doi:10.1136/bmj.322.7281.283

[2] Halamka, J. (2009, July 24). On the Road to RHIOs. Retrieved July 9, 2018, from https://www.healthmgttech.com/on-the-road-to-rhios.php

[3] Young, A., Chaudhry, H. J., Pei, X., Arnhart, K., Dugan, M., & Snyder, G. B. (2017). A Census of Actively Licensed Physicians in the United States, 2016. Journal of Medical Regulation, 103(2), 7-21. doi:10.30770/2572-1852-103.2.7

[4] Nakamoto, S. (2008, November 1). Bitcoin: A Peer-to-Peer Electronic Cash System. Retrieved June 21, 2018, from https://bitcoin.org/bitcoin.pdf

[5] Azaria, A., Ekblaw, A., Vieira, T., & Lippman, A. (2016). MedRec: Using Blockchain for Medical Data Access and Permission Management. 2016 2nd International Conference on Open and Big Data (OBD). doi:10.1109/obd.2016.11

[6] Atzei, N., Bartoletti, M., & Cimoli, T. (2017). A Survey of Attacks on Ethereum Smart Contracts (SoK). Lecture Notes in Computer Science Principles of Security and Trust, 164-186. $doi : 10.1007/978-3-662-54455-6_8$

[7] Byzantine fault tolerance. (2018, June 19). Retrieved June 21, 2018, from $https : //en.wikipedia.org/wiki/Byzantine_fault_tolerance$

[8] Szabo, N. (1997). Formalizing and Securing Relationships on Public Networks. First Monday, 2(9). doi:10.5210/fm.v2i9.548

[9] E. (2018, July 1). Ethereum/wiki. Retrieved July 9, 2018, from https://github.com/ethereum/wiki/wiki/White-Paper

[10] Wood, G. (2014, April). ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION ... Retrieved July 9, 2018, from ethereum.github.io/yellowpaper/paper.pdf

[11] Shin, L. (2017, July 17). The First Government To Secure Land Titles On The Bitcoin Blockchain Expands Project. Retrieved July 10, 2018, from *https* : *//www.forbes.com/sites/laurashin/*2017/02/07/*the − first − government − to − secure − land − titles − on − the − bitcoin − blockchain − expands − project/*#7*fd*9*c*7*d*94*dcd*

[12] Digital Diploma Pilot Program. (n.d.). Retrieved July 10, 2018, from http://web.mit.edu/registrar/records/certs/digital.html

[13] Nilsson, K. (2017, July 27). Breaking open the MtGox case, part 1. Retrieved from https://blog.wizsec.jp/2017/07/breaking-open-mtgox-1.html

[14] Nchinda, N. (2016, July 06). Exploring Pseudonimity on Ethereum – ConsenSys Media. Retrieved from https://media.consensys.net/exploring-pseudonimity-on-ethereum-dda257019eb4

[15] Wong, J. I. (2017, December 04). CryptoKitties is jamming up the ethereum network. Retrieved from https://qz.com/1145833/cryptokitties-is-causing-ethereum-network-congestion/

[16] J. (2018, July 23). Jpmorganchase/quorum. Retrieved July 26, 2018, from https://github.com/jpmorganchase/quorum

[17] HHS Office of the Secretary,Health Information Privacy Division. (2016, February 25). Individuals' Right under HIPAA to Access their Health Information. Retrieved July 20, 2018, from https://www.hhs.gov/hipaa/for-professionals/privacy/guidance/access/index.html

[18] De Angelis, S., Aniello, F., Baldoni, A., & Sassone, V. (2018) PBFT vs proof-of-authority: applying the CAP theorem to permissioned blockchain. Italian Conference on Cyber Security. 11 pp.

[19] Li, X., Jiang, P., Chen, T., Luo, X., & Wen, Q. (2017). A survey on the security of blockchain systems,. Future Generation Computer Systems, 73. $doi : https : //doi.org/$10.1016$/j.future.$2017.08.020

[20] Sweeney, L. (2002). K-ANONYMITY: A MODEL FOR PROTECTING PRIVACY. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, 10(05), 557-570. doi:10.1142/s0218488502001648

[21] Heilman, E., Kendler, A., Zohar, A., & Goldberg, S. (2015, October 04). Eclipse Attacks on Bitcoin's Peer-to-Peer Network – MIT Security Seminar – Medium. Retrieved July 13, 2018, from $https : //eprint.iacr.org/2015/263.pdf$

[22] Marcus, Y., Heilman, E., & Goldberg, S. (2018, January 9). Low-Resource Eclipse Attacks on Ethereum's Peer-to-Peer ... Retrieved July 13, 2018, from https://eprint.iacr.org/2018/236.pdf

[23] Castellucci, R. (2017, January 23). Https://github.com/brainflayer. Retrieved June 21, 2018, from https://github.com/ryancdotorg/brainflayer

[24] Ducklin, P. (2017, August 07). Serious Security: How to store your users' passwords safely. Retrieved June 21, 2018, from https://nakedsecurity.sophos.com/2013/11/20/serious-security-how-to-store-your-users-passwords-safely/

[25] Palatinus, M., Rusnak, P., Voisine, A., & Bowe, S. (2013, September 13). Bitcoin/bips. Retrieved June 21, 2018, from $https : //github.com/bitcoin/bips/blob/master/bip -$ 0039$.mediawiki\#wordlist$

[26] Z. (2018). What are zk-SNARKs? Retrieved from https://z.cash/technology/zksnarks.html

[27] Zhong, M. (2002). A faster single-term divisible electronic cash: ZCash. Electronic Commerce Research and Applications, 1(3-4), 331-338. doi:10.1016/s1567-4223(02)00024-8

[28] Bowe, S. (2017, December 14). Cultivating Sapling: Faster zk-SNARKs – Zcash Blog. Retrieved June 21, 2018, from https://blog.z.cash/cultivating-sapling-faster-zksnarks/

[29] Narula, N. Vasquez, W., & Virza, M. (2018). zkLedger: Privacy-Preserving Auditing for Distributed Ledgers. USENIX Symposium on Networked Systems Design and Implementation, 80.