Department of Electrical and computer Engineering

Part IV Research Project

Project Report

Project Number: 105

MedBloc: Blockchain for the New Zealand Healthcare System

Name: Jack Huang

Project partner: Yuanwei Qi

Supervisors: Andrew Meads

Yu-Cheng Tu

08/10/2018

Declaration of Originality

This report is my own unaided work and was not copied from nor written in collaboration with any other person.

[illegible]

Name: Jack Huang

ABSTRACT: In New Zealand, there is currently no shared electronic health record (EHR) system integrated between major healthcare organisations, such as hospitals, GPs and specialists. Due to the core characteristics of blockchain technology which enable untrusted actors to securely share information, the blockchain can be a suitable platform for building a large-scale EHR. In this paper, we introduce MedBloc, a blockchain-based shared EHR system, which enables patients and healthcare providers easy access to secure health records. This system captures a longitudinal view of the patient's health story and enables patients to give or withdraw consent over who may access their records. Health data are also secured on the system through its encryption schema and smart-contract-based access control mechanism. MedBloc not only demonstrates how the blockchain can establish NZ's first shared EHR system but it shows how blockchain can potentially disrupt the entire medical technology domain.

1. Introduction

Currently, in New Zealand, there is no integrated system between hospitals, GPs, healthcare specialists, and other healthcare providers, that enables the sharing of health records [1]. Instead, healthcare providers have their own local data storage system to store the health information of patients who have visited them. As a result, patients often must repeat their health story whenever they visit a new healthcare provider, while healthcare professionals lack the access to accurate health information at the point of care due to how a patient's health information is distributed across various data storage systems.

In recent years, the New Zealand Ministry of Health (MoH) has proposed establishing a shared Electronic Health Record (EHR) system for all New Zealanders [1]. The EHR system is a digital platform which stores patients' health records for use by individuals and healthcare providers. It will be integrated with all District Health Boards (DHBs) and healthcare organisations in New Zealand so that every individual's health records are kept on the system and are readily accessible by all healthcare professionals. The MoH outlined that the EHR system should be people-powered by allowing the patients to give consent over which healthcare provider can access their health records.

Due to the massive scale of an EHR system and the distributed characteristics of health records, we propose using Blockchain technology for the implementation of the shared EHR system.

The blockchain is a distributed ledger technology which enables a network of untrusted participants to make and reach agreement on transactions without relying on a trusted central authority [2]. On the blockchain, transactions are stored in a list of blocks, referred to as a chain. New blocks are appended to the chain as transactions are made. These blocks are immutable meaning they can only be inserted but not deleted or updated. Consensus must be reached amongst the participants in the blockchain network to ensure consistency between ledgers and asymmetric cryptography is used for transaction verification [3].

Along with these core concept, certain blockchain platforms contain embedded smart-properties which enables it to support non-financial transactions. These smart properties are commonly referred to as smart-contract and they are autonomous programs running on the blockchain network which executes business logic on each transaction.

Blockchain networks can also be distinguished by the type of permission management employed. A blockchain can be classified as either permissioned or permissionless. A typical cryptocurrency network can be described as a permissionless network, as anyone can participate on the network and all identities are anonymous. These blockchain networks are susceptible to anonymous attacks, which means that they must implement proof-of-work, or mining, as its main consensus mechanism. In a permissioned blockchain network, entities must be pre-approved before they can join the network and their real-world identities are visible to everyone on the network as well. Because of this, there is an elevated level of trust shared among the participants, so that proof-of-work consensus is not required.

A literature review was conducted to explore the capabilities of blockchain technology, proposed blockchain-based EHR solutions and medical data security frameworks. Our research found that the proposed EHR blockchains do not adequately satisfy the scope which the MoH demands for the shared EHR system. Furthermore, the proposed blockchain systems have issues with performance, scalability and availability, which can be addressed by recent innovations in blockchain technology.

In this paper, we introduce MedBloc, a people-powered shared EHR system leveraging blockchain technology to provide patients and healthcare providers with easy access to secured health records. In MedBloc, patients have primary stewardship over their health data. An immutable log of health records, generated from multiple healthcare providers, are kept on blockchain which patients have direct access to. Patients can give or withdraw consent over who can access their health records. We also encrypt all records on the blockchain and use smart-contracts to restrict access to the blockchain so that patients' data remain secure and confidential.

The remainder of the paper is organized as follows. In Section 2, we'll discuss the requirements which we set for our EHR system. In Section 3, we will discuss the technologies we used and why. In Section 4, we'll discuss the initial blockchain model, before we delve into our final blockchain architecture and model in Section 5. In Section 6, we describe the processes and the available functionality on MedBloc. In Section 7 we describe the validation and testing we performed and in Section 8, we describe the challenges we faced. Then, in Section 9, we discuss how MedBloc meets our initial requirements. Finally, we will discuss our future work in Section 10, and conclude in Section 11.

2. Requirements

Based on the scope outlined by the MoH in their report [1], we created a set of requirements which our EHR solution must satisfy. These are:

- People-powered – our EHR system should empower the patients by enabling them to have a more active role in managing their health and provide them with a convenient platform to engage with the system. The stewardship of health records is handled by the patients so that they no longer need to request access to their records from their healthcare providers. In addition to this, the system should provide patients with the ability to give and withdraw consent over which healthcare providers can access their health records.
- Single longitudinal view of an individual's health journey – data on the EHR system should not describe detailed (episodic) medical encounters. Instead, it should provide an overview of the patient's health condition over time and capture key information which is of value to healthcare professionals at the point of care.

Along with the requirements set by the MoH, we also formed additional requirements based on the based on our literature review of the different blockchain-based health data storage systems, MedRec [4] and MedShare [5]. These requirements primarily address the issues that are present in the two systems.

- Availability – Health records must be continuously available and accessible on our EHR system. In addition to this, the system must have no single point of failure. MedRec had issues with availability because health records are stored on existing off-chain databases managed by the healthcare providers. This means that health records are inaccessible if the connections to those databases are dropped.
- Scalability and performance – Our EHR system must achieve a higher level of scalability and performance than MedRec and MedShare. MedRec uses an Ethereum network, which is permissionless and thus require proof-of-work consensus. This results in poor transaction throughput and scalability. For MedShare, transaction latency increases exponentially as more nodes added onto the network.
- Immutable audit trail – changes to the patients' health records should be recorded immutably. In MedRec and MedShare, records can be altered at the source (database, cloud) without having to log the changes on the blockchain.
- Privacy and Confidentiality – records can be securely shared on the blockchain and identity may remain hidden. MedRec and MedShare do not encrypt health records shared on the blockchain.

3. Technology

3.1. Hyperledger Fabric

Hyperledger Fabric [6] is a permissioned smart-contract-based blockchain platform designed for enterprise use. It is highly modular and flexible, as it supports pluggable consensus mechanisms, multiple data storage formats and third-party certificate issuers. It also provides an SDK which allows for smart contracts to be written in general-purpose programming languages such as Java, Go and JavaScript.

Since Fabric provides a permissioned blockchain network, proof-of-work consensus is not employed. Instead, it relies on an ordering service to facilitate consensus. Aside from the ordering service, a Fabric network also has peer nodes, which are traditional blockchain nodes that process transactions and carries the ledger. It also relies on a Certificate Authority to issue identities on the network. How these components work together will be explained later in Section 5.

We specifically chose Hyperledger Fabric, due to its main consensus mechanism and its support for non-financial transactions. The consensus mechanism offered by Hyperledger Fabric will enable greater scalability and performance than the proof-of-work algorithm used by permissioned blockchain platforms such as Ethereum. Additionally, Hyperledger Fabric's architecture has extensive support for Blockchain development. The SDK's and development tools, such as Hyperledger Composer [7] provided by Hyperledger simplifies smart contract development, access control management and network deployment.

Hyperledger Fabric also provides scripts and documentation to help automatically deploy a Fabric network locally on a single machine. When these scripts are executed, the runtime library for each Fabric blockchain node (orderers, peer nodes, certificate authority) is downloaded from the Hyperledger servers. The runtime library of each node is put into a Docker container - an operating-system-level virtual machine –

so they separated from a network point-of-view, despite being hosted on the same physical machine.

3.2. Hyperledger Composer

Hyperledger Composer [7] is an open-source development toolset and framework for developing blockchain applications that run on top of the Hyperledger Fabric infrastructure. The toolset speeds up the blockchain development, by enabling us to quickly model our business network and deploy smart-contracts.

Developers can use Composer to create a model for their blockchain network by forming a business network definition (BND). The BND describes the key blockchain artefacts which will be managed by the blockchain network including Assets, Participants, Transactions and Access Control Rules:

- Asset – A class which represents a commodity that can be traded (change ownership). An asset can contain a number of arbitrary attributes, but it must specify a unique identification key. An instance of an asset is maintained on the ledger.
- Participant – A class which represents a real-world person or organisation who can own and trade assets. A participant can contain several arbitrary attributes, but it must specify a unique identification key. An instance of a participant is maintained on the ledger. Every participant can be issued an identity (certificate and signing keys) once they are added to the ledger.
- Transaction – a class which contains participants and/or assets, as well as arbitrary attributes. It serves as input for the smart-contract. No unique identifier needs to be specified for a transaction instance.
- Concept – an abstract class which is not an asset, a participant or a transaction. A concept must be contained by an asset, participant or transaction as an attribute. Within a concept, it has its own set of attributes, but a unique identifier is not required.
- Transaction Logic – code written in JavaScript which is used to process Transaction instances.
- Access Control Rules – defines rules which restrict the types of transactions that can be invoked and the ledger data that can be seen, depending on the participant who initiated the action

The BND is packaged into a business network archive (.bna) file which can be deployed on a Hyperledger Fabric network. When it is deployed, smart-contracts for create, read, update and delete (CRUD) operations will be generated for each asset, participants and transaction (transaction won't have update or delete operations); while transaction logic and access control rules will be compiled into executable smart-contracts. This collection of smart-contracts will be imported onto every peer node in the network. Additionally, Composer provides a client application which will be using the BNA file to generate APIs that is used to invoke the corresponding smart-contracts on the blockchain.

3.3. AngularJS

AngularJS [8] is a front-side JavaScript web application framework based on the model-view-controller architectural pattern. It provides the ability to manipulate the Document Object Model (DOM) without having to reload an entire webpage from a server. This is achieved by maintaining the controller and the state of a webpage on the client-side, instead of the server.

AngularJS was used to develop the web application for end users to interact with the client application. It was chosen as it had a relatively simple syntax, and it used a common scripting language, JavaScript, which we were familiar with. This enabled us to quickly build a web application prototype that works on any modern web browser to test our interactions with the client application.

Additionally, the use of JavaScript meant that we can import any JavaScript library which we wished to use. The most notable JavaScript library we used was CryptoJS, an encryption library which can perform tasks such as generating symmetric and asymmetric encryption keys and encrypting or decrypting files on the client-side without any server interaction. Despite its reported security concerns, we went forth with JavaScript because of its easy-to-setup libraries and because MedBloc is currently a proof-of-concept which demonstrates how privacy and confidentiality are enforced on the blockchain.

4. Initial Implementation

Our initial implementation of MedBloc was developed on Hyperledger Composer and deployed on a test network provided by Hyperledger Fabric. The test network had a single orderer node and a single peer node with Certificate Authority provided by Fabric.

Our Business Network Definition had Patients as assets, individual records as concepts and Healthcare Provider's and "Viewers" as participants.

The Patient asset is an object which contains all the details of a patient with attributes such as their name, address and gender. Their records are stored as an array of record objects in an attribute called 'records'. When digital records are created for the patient, it is inserted into the 'records' array. There are 6 types of records: Allergy, Procedure, Observation, Medication, Immunization and Condition.

The 'HealthProvider' (HP) participant class represents the real-world healthcare provider (as an individual or organization) who will be interacting with the blockchain. It contains public information about the HP such as clinic name, phone number and address.

The 'Viewer' participant is an entity managed by the healthcare provider to serve as a proxy between the real-world patient and the blockchain. An individual Viewer participant is bound to a single client application. The Viewer application is used to execute queries on the ledger. It is unable to modify information on the ledger, so patients are prevented from adding or updating records for themselves. A single client application can serve multiple patients, meaning that this blockchain model can be highly scalable.

To ensure privacy and confidentiality, all records and patient details on the blockchain are encrypted. This is done using authentication, a password stretcher and symmetric key encryption. Although the security framework was never completed, we initially planned to implement login functionality, so that a patient is authenticated before they gain access the Viewer application. A symmetric patient key, Pk, is generated when the Patient asset is created. Pk is used to encrypt all records and patient information sent to the blockchain. In order for the patient to readily access their encryption key, Pk will be sent to the blockchain. Before doing so, Pk will be encrypted using the user's password. Their password string is stretched using a Key Derivation Function (KDF) to generate an encryption key, Ppass. Ppass is used to encrypt Pk, denoted as {Pk}Ppass, and is added to the blockchain as an attribute of the Patient object. It is the responsibility of the patient's HP to create the patient asset on the blockchain. Therefore, in order for the HP to view the encrypted records, Pk will also be encrypted with the HP's password key, denoted as {Pk}HPpass, and this will be sent along with the patient details and {Pk}Ppass to the blockchain. To decrypt records, the encrypted patient keys ({Pk}HPpass, {Pk}Ppass), retrieved along with the encrypted records from blockchain, is decrypted using the user's password and is subsequently used to decrypt the decrods themselves.

This network model had significant security issues, centred around the Viewer participant, which made it challenging to fulfil our privacy and security requirements. The details of this will be discussed in more detail in Section 8. Due to these issues we faced, we decided to change the blockchain model to what it is currently.

5. Architecture

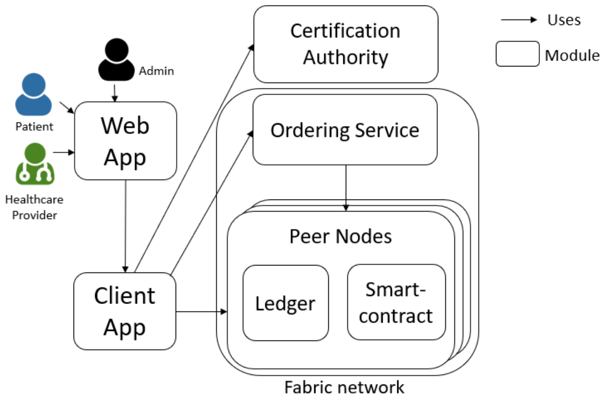


Figure 1. Architecture of MedBloc

MedBloc's final architecture is composed these the following modules, as depicted by figure 1:

Certificate Authority) Issues digital certificates for participants on the blockchain via Public Key Infrastructure (PKI). Certificates are digitally signed by the CA and binded to the participant's public key. In PKI, private keys are used to sign messages or transactions. Recipients of the transaction can verify the sender's identity and the integrity of the message by using the sender's public key to check if the signature is valid. To verify that the public key is bound to the specific identity, it validates the CA's signature on the certificate associated with the sender's public key. In MedBloc's network, a CA is provided by the Hyperledger Fabric.

Ordering Service) The ordering service is used to facilitate consensus. It is responsible for ordering transaction into chronological sequence, packaging batches of transactions into a block and distributing the block to all peers into the blockchain network. The ordering service is made of one or more orderer nodes. When more orderer nodes are added to the network, they logically behave as a single node, but with added transaction processing capacity. In MedBloc's network, only 1 orderer node is deployed.

Peer Nodes) These are blockchain nodes which are responsible for storing ledger data, executing smart contracts, endorsing transactions, validating transactions and committing blocks onto the ledger. There are two types of peer nodes, committing peers and endorsing peers.

- *Committing peers* validates the transactions given by the ordering service before it adds it to the ledger. They do not have to carry smart-contract code.
- *Endorsing peers* executes smart-contracts and signs transactions before they are sent to the respective client application. Since they also hold the ledger, they can do everything a committing peer can do.

In MedBloc's, there are 4 endorsing peers on the network.

Ledger) The ledger records all transactions which occurred in the blockchain network immutably. Every peer node has a ledger. There are two distinct part to a ledger, the world state and the blockchain

- The world state represents the latest values for each state on the ledger. The world state is stored in a database and is expressed using key-value pairs. By having the world state, the blockchain does not have to be traversed through in order to find the current value for a specific ledger state.
- The blockchain is a linked list of blocks, wherein each block it contains a sequence of transactions. Each transaction represents changes or queries to the world state. The world state can be generated by traversing every transaction on the blockchain. Invalid transactions are also stored on the blockchain, however, is not applied to the world state.

Smart-contract) Application running on every peer node, which executes business logic on every transaction. Smart-contracts are designed to provide controlled accesses to the ledger so that the ledger cannot be updated without reaching consensus. Smart-contract code must be deterministic, otherwise, consensus may never to reached between peers. In MedBloc, the smart-contracts are generated from the BNA file modelled using Hyperledger Composer.

Client Application) The client application (or client app) broadcasts proposed transactions to peer nodes and collects endorsements in the form of transaction signatures from each peer node. It sends the signed transaction to the ordering service after all signatures are collected. It also exposes a set of APIs which enable users of the app to generate transactions.

The client app is developed using Hyperledger Composer. Composer provides a set of APIs and components which can be used to invoke transactions to create, read, update, delete (CRUD) or transfer assets, from the client side.

In MedBloc, the client app takes the identity of the participant who uses it. This is for transaction signing when it proposes a transaction. Multi-user support on the client app has not been implemented yet, thus, a user must generate a client app if they wish to participate on the network.

Web application) Made using AngularJS, the web application (or web app or web portal) is the front-end client offered to end users, so they can interact with the client app and thus the blockchain. Users can view appropriate information on the ledger and input data in an intuitive fashion. There are three web apps maintained in MedBloc: the admin, patient and healthcare provider web apps. Each of the web apps offers functionalities that are appropriate for the participant who uses it. Furthermore, each web app connects to the respective client app. i.e. the admin web app must connect to the client app which has an administrator's identity. It calls the exposed REST APIs on the client app to interact with it.

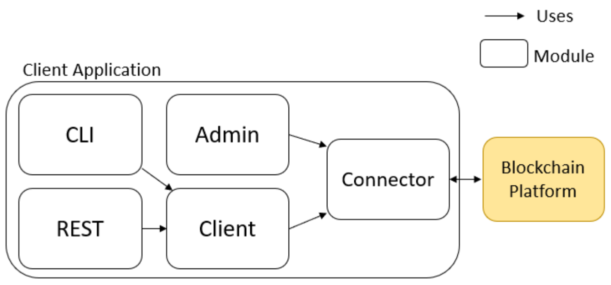


Figure 2. The architecture of the client application

5.1. Client Architecture

There are 6 primary sub-modules which make up the client application (depicted by figure 2):

Client) Exposes a set of APIs for invoking CRUD operations and transaction proposals. It also responsible for handling endorsed transaction proposals and broadcasting them to the ordering service. It interacts with the blockchain via the Connector API.

Admin) Exposes a set of APIs for deploying smart-contracts to the blockchain, updating business network definitions and updating smart-contract code. All interactions are handled by the Connector module.

Connector) Contains the connection profile type (URL, port number) for each blockchain network node (orderers and peers). It routes transactions and other messages to and from the blockchain network.

CLI) The Command Line Interface (CLI) exposes the functionality in Client and Admin modules so that developers can interact with the client using a scripting language. Developers can invoke any transactions that are available on the Client and Admin through the CLI.

REST API) This serves as a REST endpoint which fully exposes the functionality offered by the Client and Admin modules.

5.2. Business Network Definition

Since smart-contract were developed using Composer, the blockchain network is modelled by the Business Network Definition (BND) and it is structured as follows.

5.2.1. Assets

The assets maintained on the network are records and encrypted keys. The 'Record' asset is the abstract class which is the parent of all record types. The non-abstract record classes are the same as our previous implementation: Allergy, Procedure, Observation, Medication, Immunization and Condition. Additionally, there is an asset named PatientKey. This asset contains an encrypted symmetric "patient" key, along with a reference to the Patient participant, which the key belongs to, and the Health Provider participant, which the key is shared with. How patient keys are used in MedBloc will be explained later in Section 6. Unlike the initial implementation, there are no Concept classes in our BND.

5.2.2. *Participants*

There are two participants described in the BND, patients and healthcare providers. The Patient participant represents real-world patients whose records will be stored on the blockchain. It maintains attributes which describe the patient such as name, address and gender. It also contains an attribute which holds an array called "consentedHPs". The consentedHPs array holds references to HP participants who the patient has given their consent to access their records. The consentedHPs array will be accessed by smart-contracts to enforce access control. This will be described in further detail in Section 6.
The HealthProvider (HP) participant class remains unchanged from the initial implementation, except for the addition of the public key attribute. This will be used for secure key sharing.

5.2.3. *Transactions and Transaction Logic*

There are three transaction classes: ShareKey, RequestRecordSharing (RRS) and Revoke Medical Records Sharing (RMRS). Each transaction has a transaction logic function to process it. RRS is invoked by the HP to request the patient key from the specific patient. ShareKey is invoked by the patient to share to patient key with a specific HP. ShareKey must contain an encrypted copy of the patient's patient key (as a string) and a reference to the patient itself and the HP. When the transaction is processed, a PatientKey asset will be generated using the properties specified in the transaction, and a reference to the HP will be added to the consentedHP array. Finally, the RMRS transaction is invoked by the patient to revoke the access rights of a particular HP to their private key on the blockchain. When the transaction is processed, the HP is removed from the consentedHPs array.

After the transactions are processed by the relevant smart contracts, a notification event is emitted. Client apps must subscribe to these events to receive it. Patient's client apps are subscribed to RRS notifications, while the HP's client apps are subscribed to ShareKey and RMRS events.

5.2.4. *Access Control Rules*

26 access control rules were described in the BND. The key rules were that HPs cannot perform CRUD operations on records of patients who did not list them in their consent array. Patients are not allowed to view details, including records of other patients, and they are prevented from editing their own records. Patients and HPs are only allowed to update their own participant object. Finally, the network administrator can invoke any smart-contract available on the blockchain network.

5.3. Transaction Flow

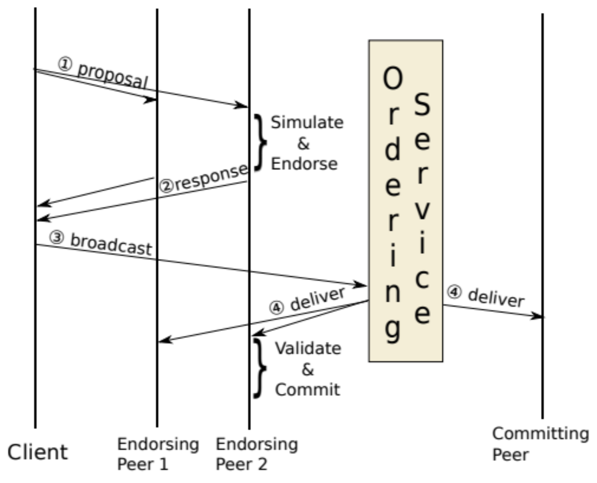


Figure 3. Sequence diagram of the transaction flow [9]

Transactions represent query or update requests to the ledger. Figure 3 depicts how a transaction is processed on the Fabric network (and thus, MedBloc's network). To initiate a transaction, the client application sends the proposed transaction to any to a set of endorsing peer nodes within the blockchain network. The transactions must include the smart-contract name and version number. The number of nodes it sends the transaction to depends on the endorsement policy associated with the smart-contract that will be invoked.

After a peer node receives the transaction proposal, it will verify the signature of the client associated with the transaction, via PKI. Each peer node will then independently execute the specified smart-contract against the given transaction. The results of the simulation represent a set of ledger state changes that would occur if the transaction goes through; referred as the read-write set. The peer node will endorse the transaction, by signing the transaction with its digital certificate, if and only if the output of the simulation satisfies the smart-contract's endorsement policy. The signed transaction proposal and its associated read-write set are returned to the client application.

If the transaction proposal is a ledger query request, the client application does not have to do anything further. If it is an update request, the client application must wait until "enough" endorsing messages are returned from the peer nodes. The exact number which is needed depends on the endorsement policy defined for the smart-contract. If the client app does not receive the required number of endorsements, the client app will retry later or abandon the transaction proposal altogether.

The read-write set returned from the peer nodes should be the identical, as peer node executes the same smart-contract code. In some circumstances, the read-write sets are different between responses. This may be because the read-write set was generated at separate times on peers that had different ledger states. In this case, the transaction proposal must be re-broadcast to the peer nodes. Another reason is that the smart-contract code is non-deterministic, which is likely due to dependencies with random number generation or time values. The client application cannot continue further with this transaction, without contacting the network administrator to change the smart-contract code.

Once enough endorsements are received and the read-write set from the responses are found to be consistent, the client app will broadcast the transaction proposal along with the endorsing messages to the ordering service. The ordering service is responsible for sorting each transaction relative to other transactions and packaging the batches of transactions into blocks for distribution. The order at which the transactions are sorted within the block is not dependent on when the transaction arrived at the ordering service. It is usually based on when the transaction is proposed. The sequence of transactions in a block is final and cannot be changed by any peer node (immutable). The orderer does not host smart-contracts or the ledger; thus, it does not validate the transactions it's given. Its sole purpose is to package transactions in a strict order, without dropping or deprioritizing any that it has been given.

The same copy of the block will be distributed to all peer nodes in the channel. Upon receiving the block, each peer node will independently process the transaction in the sequence which it appears in the block. Each peer must also verify that the transaction has been endorsed by the required number of signers, as specified in the smart-contract endorsement policy, and that the transactions read-write set is consistent between endorsers.

After a peer has successfully validated each transaction in the block, the transactions are committed on the ledger. Transactions which failed validation are not applied to the ledger (world state does not change), however, they remain on the blockchain for auditing purposes. Each transaction will have a tag associated with it indicating if it's a successful or failed transaction.

Peers may emit an "event" after the validation process is complete. An event contains information about the transactions that were processed, such as whether they are validated or not. Client apps can subscribe to certain events, so they are notified when they occur.

6. Application Processes

There are five key processes which MedBloc can handle. These processes correspond to actions which are expected on a shared EHR system. When discussing the processes, we assume the network is running in the same configuration as what was described in Section 5. We also assume that a Composer client is running for every participant who was created on the system, and an appropriate web portal (corresponding to the respective participant identity imported to the client app) is connected to the client app.

6.1. Patient Creation

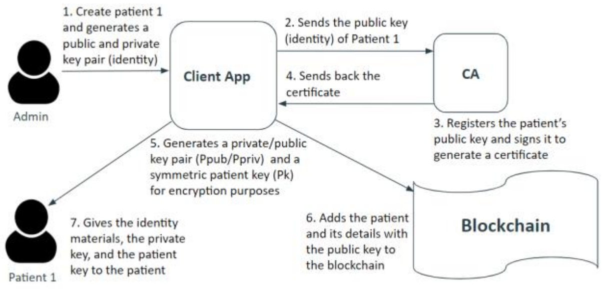

1. Create patient 1 and generates a public and private key pair (identity)
Admin
Client App
2. Sends the public key (identity) of Patient 1
CA
4. Sends back the certificate
3. Registers the patient's public key and signs it to generate a certificate
5. Generates a private/public key pair (Ppub/Ppriv) and a symmetric patient key (Pk) for encryption purposes
6. Adds the patient and its details with the public key to the blockchain
Blockchain
7. Gives the identity materials, the private key, and the patient key to the patient
Patient 1

Figure 4. Patient Creation process flow

To create a patient (refer to figure 4), as a participant on MedBloc, the network administrator must enter the patient details on the Admin web portal. The patient details are wrapped in a JSON file and sent to the client application via the REST API. The client application will form an identity for the patient by issuing them a digital certificate via the Certificate Authority. It then generates a public-private key pair (Ppub, Ppriv) and a symmetric patient key (Pk) which is used for record encryption and secured sharing. Finally, the patient details and its public key is added to the blockchain by invoking the appropriate smart-contract. The cryptography materials are sent to the web app after the process completes and is available to be downloaded by the administrator from a pop-up modal. The cryptography materials cannot be retrieved again after the download modal is closed. In a real-world scenario, the cryptography materials will be distributed to the patient it belongs to.

6.2. Key Upload

The keys generated by the admin (pk, Ppub, Ppriv) can be "uploaded" onto either the Patient portal or the HP portal. When the user opens either portal, they are presented with a "login" screen. This screen does not perform authentication, instead, it provides a button for users to browse their file directory and select the required cryptography materials to upload. By "upload" it means that the cryptography materials are stored as a state on the web browser, rather than its traditional meaning of sending files over the internet. On the login screen, the user can input their participant ID. The web client only redirects the user to the main activity if the user has supplied the correct file types and an existing participant ID (by querying the ledger). It will not verify if the keys belong to the participant nor will it authenticate the participant associated with the ID. The purpose of this screen is for users to supply information which the processes require.

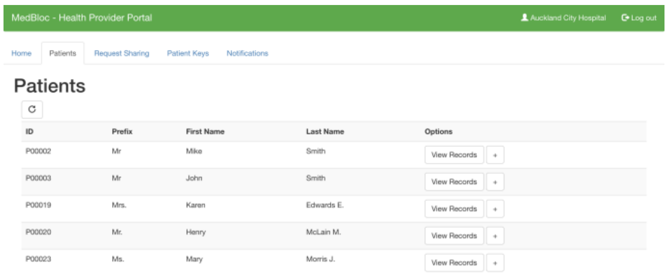
6.3. Record Sharing

In order for a healthcare provider (HP) to view and add records to a particular patient, they must have access to the patient's patient key (Pk). In MedBloc, the Pk is used to encrypt all records for a patient. The HP can request for the patient's Pk on the HP's web portal. When the action is invoked, the HP's client application will direct the request to the blockchain network's peer nodes as a transaction. When the transaction is executed, the peer nodes will emit a notification event to client applications. The corresponding patient's client application which subscribes to "request key" event picks up the message and relays it to the patient's web app. The request will show up as a notification on the patient's portal.

If the patient wishes to grant the requesting healthcare provider consent to access their records, they must share their patient key Pk through the Patient web portal, either by typing HPs ID in the My Shared Keys page or their clicking on "Share" in the notifications panel. The Pk, which is initially uploaded on the web app, is sent to the client app. The client app will retrieve the requesting healthcare provider's public key, HPpub, from the blockchain. HPpub will be used to encrypt Pk, denoted as {Pk}HPpub. {Pk}HPpub will be sent as a ShareKey transaction to the blockchain network. As part of the smart-contract logic, {Pk}HPpub will be stored on the blockchain ledger and a reference to the corresponding healthcare provider is added to the consent array, consentedHPs, belonging to the patient.

After {Pk}HPpub is committed onto the ledger and the patient's consent array has successfully updated, the peer node will emit a ShareKey event. The HP's client app will pick up on the event and then show a notification on the HP's web portal to indicate that a patient has shared the key with them. The HP will now know that they can access the patient's record.

Patients can view all a list of all the HPs they shared their key in the "Shared Keys" (figure 6) tab, while healthcare providers can view a list of all patients who shared their keys with them in the Patient tab (figure 5). When these tabs are opened, a query request is sent to the blockchain, to return the subset of patients or HPs.

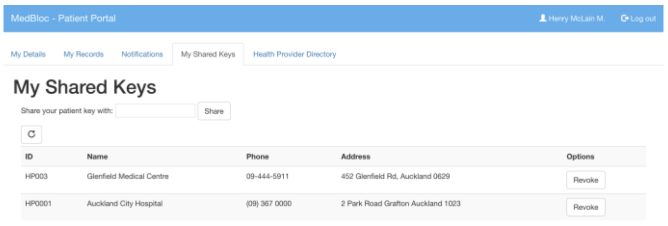

MedBloc - Health Provider Portal   Auckland City Hospital   Log out

Home | Patients | Request Sharing | Patient Keys | Notifications

Patients

ID	Prefix	First Name	Last Name	Options
P00002	Mr	Mike	Smith	View Records +
P00003	Mr	John	Smith	View Records +
P00019	Mrs.	Karen	Edwards E.	View Records +
P00020	Mr.	Henry	McLain M.	View Records +
P00023	Ms.	Mary	Morris J.	View Records +

Figure 5. Patient Tab displaying the patient table


MedBloc - Patient Portal   Henry McLain M.   Log out

My Details | My Records | Notifications | My Shared Keys | Health Provider Directory

My Shared Keys

Share your patient key with: Share

ID	Name	Phone	Address	Options
HP003	Glenfield Medical Centre	09-444-5911	452 Glenfield Rd, Auckland 0629	Revoke
HP0001	Auckland City Hospital	(09) 367 0000	2 Park Road Grafton Auckland 1023	Revoke

Figure 6. My Shared Keys tab displaying the HP table

6.4. Accessing Records

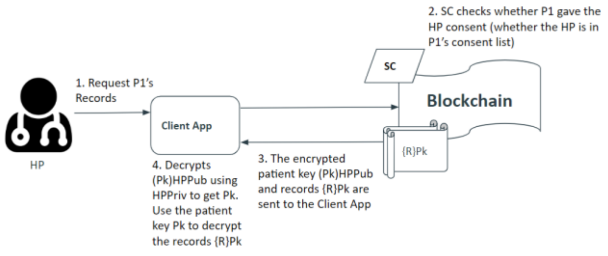

2. SC checks whether P1 gave the HP consent (whether the HP is in P1's consent list)
SC
1. Request P1's Records
Client App
Blockchain
{R}Pk
HP
4. Decrypts (Pk)HPPub using HPPriv to get Pk. Use the patient key Pk to decrypt the records {R}Pk
3. The encrypted patient key (Pk)HPPub and records {R}Pk are sent to the Client App

Figure 7. Accessing records process flow. P1 is Patient 1

Clicking on "View Record" for a given patient, in the Patient table on the HP's web portal, will retrieve all records for that patient. The client application will send a request for the patient's records and their encrypted patient key, {PK}HPpub, to the blockchain nodes as a transaction. The smart-contracts on the blockchain will first verify if the corresponding patient has listed the HP in their consent array before it retrieves the records. After successful verification and smart-contract execution, {Pk}HPpub and the encrypted records, {R}Pk, is sent to the HP's web app, via their client app. The web app will decrypt {Pk}HPpub using HPpriv which has been uploaded earlier onto the web app, to retrieve Pk. Pk will then be used to decrypted {R}Pk. Finally, the decrypted records, R, will be displayed on the web app for viewing. This process is depicted in figure 7.
For the patients to view their own records on the patient portal, they need to navigate to the "My Records" tab. The same underlying process for accessing records is repeated for the patient, except they use the patient key which they uploaded on their web client beforehand, to decrypt the records.

6.5. Adding Records

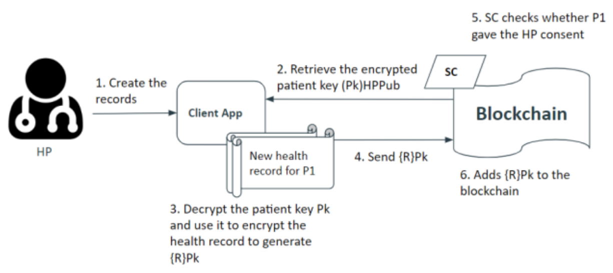

5. SC checks whether P1 gave the HP consent
1. Create the records
2. Retrieve the encrypted patient key (Pk)HPPub
SC
Client App
Blockchain
HP
New health record for P1
4. Send {R}Pk
6. Adds {R}Pk to the blockchain
3. Decrypt the patient key Pk and use it to encrypt the health record to generate {R}Pk

Figure 8. Adding records process flow. P1 is Patient 1

The same process for accessing records is followed for adding records (depicted in figure 8). The HP must add the record information using the form provided by the web app, by clicking on the "+" in the Patient table. It will attempt to retrieve the patient key, Pkm from the ledger after the information is supplied. The smart-contract on the blockchain will verify that the HP is included in the corresponding patient's consent array before it returns {Pk}HPpub to the web app, via the client app. Using the healthcare providers private key, the web app decrypts {Pk}HPpub to obtain Pk so it can encrypt the record information. The encrypted record is sent to the blockchain as a transaction, where it will be eventually committed onto the ledger as a record for the given patient.

6.6. Revoking Sharing

The patient can revoke their consent for an HP to access their records by clicking on the 'Revoke' button on the HP table in the My Shared Keys tab. A Revoke Medical Records Sharing (RMRS) transaction will be formed by the client app and sent to the blockchain. When the transaction is processed, the HP referenced in the RMRS transaction is removed from the patient's consent array and the corresponding PatientKey asset ({Pk}HPpub) is deleted from the ledger. Upon successful transaction execution, an RMRS notification is emitted from the blockchain nodes. The corresponding HPs client app will pick up the event and shows it as a notification on the HPs web app.

Note that the patient's public-private key pairs are not used in any of the processes. These were intended to be used to encrypt the patient key on the blockchain so that users can maintain fewer cryptography materials. We deemed that this was not necessary, however, it could be implemented in the future.

7. Validation

Validation of MedBloc's security architecture was conducted with Dr Rizwan Asghar, a leading expert in medical data storage security. He was satisfied with how records are encrypted and securely shared on the blockchain without compromising any encryption materials. This is because the symmetric patient key and private keys are managed off-chain by their respective owners. He was also pleased our patient key concept. Having a patient key means that only one key is needed to encrypt and decrypt all records of a patient generated from different sources. This makes it convenient for patients and healthcare providers alike to access health records. The patient key is also difficult to compromise, as the key itself is stored off-chain and only an encrypted copy of the key can be stored on the blockchain.

Asghar was also impressed with MedBloc's ability to capture patient's consent. He was pleased with how we used smart-contracts to enforce access control and to manage consent. Having a consent list on the blockchain means that patients can give and withdraw consent whenever they want. By implementing access control over the patient key, it prevents unconsented healthcare providers from viewing for adding records for a specific patient, even if they were granted consent at some point before their permissions were revoked.

Asghar also praised the patient-focused nature of MedBloc. In MedBloc, the responsibility of managing records is given to the patient instead of their healthcare providers. This is because, unlike traditional health data management systems, the encryption materials are kept by the patients. The result of having a patient-focused EHR system means that the patient no longer needs to request and potentially wait up to 60 days to access their own medical information [4], as this can be done instantaneously.

While our security framework provided strong privacy and confidentiality for patients and their health data, Asghar discovered one security vulnerability within the framework. It was found that since patient's cryptography and identity materials (patient key, public-private key pairs) are generated on the admin web app, it can be compromised if the admin's web app stores the materials even after it shares it with the respective patient. To address this issue, the generation of cryptography materials should be handled by the Patient's web app. It should also be the responsibility of the Patient to certify their public key with a CA, as well.

Asghar also suggested some improvements we can make to our security framework to further strengthen the system. Firstly, we should implement time limits on all consent given by patients. Currently, in MedBloc, once the patient gives consent to an HP, that HP has unlimited to access to the key until the patient has revoked their consent. This can result in privacy issues, as some patients only intend to share their data with certain HPs for a limited period of time, however, they may forget to revoke their consent after granting it. A solution to this is to implement access tokens that can be used to define the scope and lifetime of the consent. The access token will enable patients to specify how long their consent lasts for and/or how many accesses (to their patient key) the HP is allowed.

In addition to this, we should implement finer grained record requests and sharing. Although HPs are able, in the current system, to send requests for patient key's, they do not have to specify why they are requesting their key and what records they are accessing. An improved request feature, for HPs to specify their reason for accessing the patient records and how long they wish to access it, can be implemented in conjunction with the access tokens. Furthermore, additional notification alerts can be added, so that patients are notified of when their HPs has accessed their records, and which records they are viewing.

7.1. Testing

To test the behaviour the smart-contracts several unit tests were written. These tests validated the correctness of the transaction logic and the access control rules, such that it matches what we expect it to do. For transaction logic testing, we tested for the correctness of the read-write (output sets) and the generation of events. For access-control tests, we focused on testing the consent array; ensuring that it is properly queried when enforcing access control and the appropriate HPs are added to the array.

The unit tests can run without having to set-up a blockchain network or even a peer node beforehand. During the setup phase, the BND is compiled and installed on a NodeJS runtime to simulate the Fabric blockchain environment. Since it runs NodeJS, any JavaScript test framework can be used to test the smart-contract code. Thus, we used Mocha and Chai as our unit test framework. To run the tests, we simply run `npm test` in the directory where our test script is located.

8. Challenges

8.1. Initial Implementation

There were several issues we faced with our initial blockchain model which forced us to change to the current model. Firstly, it is also difficult to enforce access control with the Viewer participant. We cannot restrict patients from viewing other patients' records as the Viewer must retain permissions to access all records on the ledger. This is because Fabric uses role-based access control, where each participant type has different access permissions. As such, to enforce the above access control rule, we must make the patient a participant in the system.

This blockchain model also cannot easily capture the patient's consent or share records. Since all patient interactions are handled by the Viewer application, they are not allowed to invoke transactions that can update the ledger. Thus, they are unable to express who they want their records to be shared with on the blockchain system.

We knew that changing the Patient from an asset to a participant will make the system much less scalable. However, strong security is paramount in any health storage context, thus, we thought that it would be a reasonable tradeoff to make.

8.2. Web App Implementation

A significant portion of our time was spent on implementing the web front-end, despite it not being the main focus of our project. We initially planned to make the web app for testing purposes; so that we can interact with the client app without using the CLI tool or directly calling the REST server. However, the scope of the web client changed as we decided to use the web app for public demonstration. Thus, we had to change our web design to look professional and easy for the general public to use. The improved design of the front-end featured pop-up modals, a login screen, notification handling and "cards" to display record information. To implement these features, we had to learn advanced AngularJS functionalities such as document object model (DOM) manipulations, routing, and parsing states. The learning process took a significant amount of time due to our unfamiliarity of AngularJS.

Another challenge we faced when designing our web-app is that we had three separate code bases to maintain. We decided that for demonstration purposes, we would have separate web portals for the admin, healthcare provider and the patient. These web portals were developed as separate web apps, as they all have different functionalities. The healthcare provider and patient web portal were forked from the admin web portal (initially our test web application) so that each web application share the same framework and libraries (AngularJS, Bootstrap and CryptoJS), as well as some common functionalities.

Developing three web applications proved to be very challenging and tedious. As some web apps shared the same features, changes to one feature in one web portal must propagate to the same feature on another web portal. This became tedious to maintain as we often forgot to propagate our changes across web portals. The reasoning behind having three web apps was because the functionalities were mostly complete by the time we forked, so we thought that we did not need to make too many changes on them. This proved to be mostly untrue, as towards the exhibition day, we made numerous changes across a range of functionalities on the web apps.

It also became difficult to navigate our working directory, as the file names were the same between the web app directories. So, sometimes we may unintendedly make changes to one web client that were supposed to be made for another.

To address this issue, we should have made separate web pages designated for each type of end-user, on a single web app. This will allow the web pages to share the same controller, logic and states, which will make development much simpler.

A challenge with using AngularJS is that all behavioural code must be written in JavaScript. Since JavaScript is an interpreted programming language, which does not provide type-checking, it is very easy to write code that will lead to errors.

To address this problem, we should have used web development framework which used a statically typed language such as React or Angular 2+. Typescript is a programming language which compiles to JavaScript and offers features such as static type checking, class-based object-oriented programming and integrated static code analyser. This would make it easy for us to identify errors as the compiler can detect obvious code errors.

8.3. Security

We initially planned to implement secure authentication for the client application. Implementing authentication will help protect accesses to the client app and can secure the user's cryptography materials, to reduce the risk of them losing their encryption keys or having it stolen. Authentication can be achieved by implementing a typical login page where the user must supply their username and password to the system. The challenge with implementing the login feature is that it must be implemented at the web app side. This means that the client app must be notified of when a user has successfully signed in, through an external module.

The Composer client only supports single sign-on services using an OAuth provider such as Github or Facebook. To set up the authentication framework, we must create our own authorization server. The users will register their identity on the server by providing a username and password. The authorization server must securely store a strongly hashed copy of the password within its database. The login screen on the web app must securely send the login materials to the authorization server in order for the identity to be verified. Upon successful verification, the access token is given sent to the web app. The web app will use the access token whenever it interacts with the Composer client app. The client app will verify the identity associated with the access token with the authorization server before it processes any request sent to it.

While this framework can be set up, it would incredibly time-consuming and difficult for us to implement as we are dealing with unfamiliar technology and security protocols. Further, since the client app cannot support more than one identity, it is not worthwhile implementing secure authentication with the current architecture.

9. Discussion

MedBloc provides a people-powered EHR system, where patients retain primary stewardship over their health data and are able to have their consent captured. By having the patient keep their cryptography materials themselves, it means HPs must request permission from their patients to access health records, while patients are able to retrieve their health information at any time they want. Furthermore, the addition of smart-contract enforced access control provides a reliable and convenient way for patients to specify which HPs they wish to give consent to or to withdraw consent from.

The types of record assets which are available to store on MedBloc (allergies, conditions, medications etc.) restrict healthcare professionals to only add health information that valuable at the point of care. This reduces the number of detailed health records that need to be managed by the blockchain. A longitudinal view of a patient's health story is captured on MedBloc. Medical records are stored immutably on the blockchain, which means that healthcare professionals can see the progression of a patient's health over time since all records generated remain on the ledger permanently and are ordered chronologically.

MedBloc provides patients and healthcare providers easy access to health records. MedBloc's decentralized network allows any blockchain node to serve the needs of the patient or healthcare provider. Each blockchain node provides the same functionality (same smart-contract code), shares the same ledger, and can still commit transactions even if not all nodes in the network are running. Thus, by storing health records directly on the blockchain, it means that health records are continuously available on the blockchain. The level of

Requirement	**MedBloc**	**MedRec [4]**	**MedShare [5]**
People-powered	Patient's retain stewardship over their health data. Patient's consent is captured by the system.	Patient-initiated health data sharing, however, consent revocation is not captured	Consent not captured.
Longitudinal health story	Only health information that are of value at the point-of-care are stored on the blockchain.	All medical encounters are available over the blockchain.	All medical information on the cloud is available to access.
Availability	Records replicated across a distributed network with no single point of failure.	Records in off-chain databases, which can be inaccessible if database is brought down.	Records stored on the cloud (unknown cloud architecture)
Performance	Potentially >175 transactions/second [9]	Maximum 15 transactions /second [10]	Not disclosed
Scalability	No proof-of-work consensus, but there is a 1:1 ratio of client apps to active participants.	Proof-of-work consensus – harder to scale.	Latency increases exponentially with the number of users.
Immutable audit trail	All request to the ledger, even if unsuccessful, all recorded.	Data can be altered at the source without it being logged on the blockchain.	Data can be altered at the source without it being logged on the blockchain.
Security/privacy	All records are encrypted. No exposure of cryptography materials.	Data is not required to be encrypted.	Continuous monitoring of data movements. Full data encryption.

Table 1. Summary table of how the requirements are satisfied/unsatisfied by different blockchain-based EHR systems

data redundancy achieved by MedBloc is difficult and costly to replicate on the typical database system, and it outdoes what MedRec and MedShare can provide, due to how records are directly stored on the blockchain.

By using a permissioned blockchain network, MedBloc can achieve provide greater scalability and performance than proposed blockchain-based EHR, MedRec [4] and MedShare [5]. While we are not able to benchmark MedBloc, we are confident that based on experimental values, MedBloc (with 4 endorsers) can achieve a throughput of over 175 transactions per second [9]. This is greater the average transaction throughput of 15 t/s achieved by Ethereum [10], which is the blockchain platform behind MedRec. Furthermore, since MedBloc does not have to continuously monitor user activity, it can scale more effectively than MedShare.

However, the scalability of our system can be further improved upon. Currently, a client application must be generated for every patient or healthcare provider who wishes to interact with the blockchain. This is not scalable as potentially millions of client apps must be maintained. We will address this problem by creating multi-user support for client apps.

While there are major security risks with keeping health data in a distributed network, a high level of privacy and confidentiality is achieved by the security framework implemented on MedBloc. All records are encrypted before they are sent to the blockchain and critical encryption materials are all managed off-chain. As validated by Dr Rizwan Asghar, our security framework has very few points of vulnerability that can leave critical encryption materials or the patient's health data exposed.

Table 1. summarises how MedBloc satisfies all requirement we initially set and how the other blockchain-based EHR's compare with MedBloc in terms of satisfying those requirements.

10. Future Work

For future work, we will implement the suggested security improvements provided by Rizwan Asghar, as detailed in Section 7. This will include changing the generation of the encryption material from the admin web app to the patient and HP web apps, implementing access tokens and allowing finer-grained record sharing.

In addition to this recommended security improvement suggested by Asghar, we will also explore implementing secure authentication at the client app.

We will also reconfigure the client app to support multiple users. This will enable MedBloc to better scale with increasing participants.

Finally, we will consult with healthcare professionals and blockchain experts to find identify further improvements for MedBloc.

11. Conclusion

MedBloc demonstrates how blockchain technology can be leveraged to create a people-powered shared EHR system. In our system, patients are given primary stewardship over their personal health data by allowing them direct access to their health information on the blockchain and by capturing their consent. We employed a security schema which encrypts all records on the blockchain but can be decrypted by the patient it belongs to and the healthcare providers who the patients consented to. Our security framework has been proven to be highly secure, with few points of vulnerability, while being convenient for the healthcare provider's and patients to access health records. Further, our system has a higher performance and scalability potential than traditional blockchain networks. MedBloc has not only shown that it can meet the scope of being New Zealand's future EHR system, but it has proved that blockchain has a place in the medical technology domain and beyond.

Acknowledgements

I like to thank my partner Yuanwei Qi, supervisors Andrew Meads and Yu-Cheng Tu, and security consultant Rizwan Asghar for their help on the project.

References

[1] C. Reid and G. Osborne, "Strategic Assessment: Establishing the Electronic Health Record," 2016.

[2] X. Xu, C. Pautasso, L. Zhu, V. Gramoli, A. Ponomarev and S. Chen, "The Blockchain as a Software Connector," in *2016 13th Working IEEE/IFIP Conference on Software Architecture*, 2017.

[3] Z. Zheng, S. Xie, H. Dai, X. Chen and H. Wang, "An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends," *2017 IEEE 6th International Congress on Big Data*, pp. 557-564, 2017.

[4] Asaph Azaria, A. Ekblaw, T. Vieira and A. Li, "MedRec: Using Blockchain for Medical Data Access and Permission Management," in *2016 2nd International Conference on Open and Big Data*, 2016.

[5] Q. Xia, E. B. Sifah, K. O. Asamoah, J. Gao, X. Du and M. Guizani, "MeDShare: Trust-Less Medical Data Sharing Among Cloud Service Providers via Blockchain," *IEEE Access*, vol. 5, pp. 14757-14767, 2017.

[6] Hyperledger, "Hyperledger Fabric Docs," [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-1.2/index.html. [Accessed 8/10/2018].

[7] Hyperledger, "Hyperledger Composer Docs," [Online]. Available: https://hyperledger.github.io/composer/latest/introduction/introduction.html. [Accessed 8/10/2018].

[8] AngularJS, "AngularJS," [Online]. Available: https://angularjs.org/. [Accessed 8/10/2018].

[9] P. Thakkar, N. Senthil and V. Balaji, "Performance Benchmarking and Optimizing Hyperledger Fabric Blockchain Platform," arXiv preprint arXim:1805.11390, 2018.

[10] Coindesk, "How Will Ethereum Scale?," [Online]. Available: https://www.coindesk.com/information/will-ethereum-scale/. [Accessed 8/10/2018].