2018 21st Euromicro Conference on Digital System Design

Identity and Access Management with Blockchain in Electronic Healthcare Records

Tomas Mikula and Rune Hylsberg Jacobsen
DIGIT, Department of Engineering, Aarhus University, Denmark
Emails: mikula.work@gmail.com and rhj@eng.au.dk

***Abstract*—Blockchain has proved itself to be tamper resistant and secure. It is increasingly getting attention from companies changing from centralized to decentralized systems. This paper proposes a system for identity and access management using blockchain technology to support authentication and authorization of entities in a digital system. A prototype demonstrates the application of blockchain in identity and access management using the Hyperledger Fabric framework. It provides a proof of concept based on a use case concerning Electronic Health Records from the healthcare domain where an immutable and auditable history is desired for data concerning patients. Basic authentication and authorization operations are able to execute in 2-3 seconds with an initial size of blockchain of about 3.8 MB covering physicians in Denmark.**

Keywords*—*Blockchain, authentication, authorization, electronic health records

I. Introduction

Decentralized networked systems are increasingly getting attention due to their resilient properties. The blockchain is an example of a decentralized structure used in cryptocurrencies such as Bitcoin [1]. It has been demonstrated that trusted, distributed, and secure computing is possible in the financial domain. A blockchain is a chain of digitized information where blocks of transactions are cryptographically linked. A blockchain is distributed across the participant's nodes in the network and does not require a centralized trusted authority. It eliminates the problem of maliciously altering the blockchain, because every time a block is added to the chain, the majority of the nodes in the network must validate the block. This concerns the integrity of the blockchain given that the majority of the nodes can be trusted. Valid transactions will lead to an update of the *world state* of the blockchain to provide easy access to the current state of the network.

Identities of resources in a system can be obtained by using public key cryptography [2], [3]. In public key cryptography, such as RSA [4], the public-private key pair is unique for the issuing entity. The public key may be used as an identity for an entity and only the one who owns a matching private key can sign a message, whereas everyone else can validate the authenticity of the message by using the public key. This means that an identity can be created by generating a new public-private key pair without the involvement of a Central Authority (CA). Another scheme that do not require a CA include Identity-Based Cryptography (IBC) where it is possible to calculate public keys representing resources by using a mathematical master secret and the unique IDentifications (IDs) of resources [2], [5].

In a blockchain network, entities can agree on transactions following a set of rules set out in *smart contracts* [6]. A smart contract is a piece of software code that executes on the blockchain and affects entities, which entered the contract. Smart contracts provide a set of functions allowing read and write operations to be made against the blockchain [7]. The main use of smart contracts have so far been in cryptocurrencies [6]. One of which is in a popular blockchain platform called Ethereum [8], [9]. Users can create decentralized applications, which use smart contracts. In this case, the contract code is written in the Solidity programming language [8] and each interaction with the contract costs a small fee, which is paid in the cryptocurrency Ether.

This paper explores the potential of blockchain technology applied to a decentralized system for identity and access management. It exploits a blockchain protocol based on the Hyperledger Fabric [10] to implement basic authentication and authorization operations as registration, login, grant/revoke permissions, and update of the system. An immutable logging of current permission states for resources in the system is obtained. The blockchain is distributed to the participating nodes in the network, which validate transactions. We demonstrate how the system can be deployed as a simple and user-friendly web application and in a case study for the healthcare domain.

A. Case Study: Electronic Health Records

To validate the functionality and assess the performance of the proposed system for identity and access management, we study a use case from the healthcare sector. There are several reasons for using a blockchain for authorization in a healthcare scenario. First, the need of an immutable and auditable history is wanted because permissions to access patient's data change regularly from one actor to another and traceability is of utmost importance in a case of any problems. Second, security risks and cost of trust are reduced by having permissions stored in a decentralized manner rather than using a central entity or distributed database where a single entity is in control. Because access to patient data is less time-critical, as transaction speed is slower in a blockchain network, it is advantageous to have the record of permission changes and trust at the expense of the speed of transactions.

Denmark has a digitalized healthcare system where healthcare personnel work with sensitive data on a daily basis [11]. Today, there are two types of systems that the healthcare industry is using for storing patient's data. One solution is the paper-based system where the data about patient's health histories are written and stored by specific doctors. The other solution is based on the digital counterpart i.e., the Electronic

978-1-5386-7377-5/18/$31.00 ©2018 IEEE
DOI 10.1109/DSD.2018.00008

Health Record (EHR). To offer personalized care, it is essential for a doctor to have access to the medical record of a patient, whether it is in case of emergency or a regular health check. While the EHR software is efficient in processing the patient's information, other problems may arise. The data stored in EHRs are sensitive and must be made available only to the users who are in need of them and it must be protected from any unauthorized access.

The paper is organized as follows. Sec. II describes work related to our system for identity and access management. Sec. III introduces blockchain frameworks and smart contracts. In Sec. IV, we present the system design and Sec. V details the most important aspects of our implementation. Sec. VI validates the design in a case study concerning EHRs from the healthcare domain. Finally, Sec. VII discusses our findings and Sec. VIII concludes the paper.

II. Related Works

Relatively few proposals for applications using blockchain for identity and access management exists. Zyskind et al. [12] studied the decentralization of privacy using blockchain to protect personal data. The focus of this decentralized personal data management system was to give users control over what data to share and to allow users to revoke access to that data. The solution combined a public blockchain network with an off-blockchain storage (key-value pair storage with encrypted raw data) that was present in each node. In their work, the blockchain consisted of only hash pointers to that data. The system proposed in this paper uses a similar principle with an off-blockchain storage, where the data of the user is stored. However, in our case, it is not a dedicated storage specially created for the identity and access management system. Rather it is an existing database that is already used. It can easily be connected to the identity and access management system in the blockchain network. Regarding authorization, instead of storing permissions in the off-blockchain storage, these are stored in the blockchain in our system. This eliminates the need to replicate the off-blockchain storage on other nodes as the permissions are registered in the blockchain and can be accessed and validated by members of the consortium blockchain network.

Cresitello-Dittmar [13] proposed an application of a blockchain for authentication and validation of identity. The basic idea was to allow a 3rd party service to validate identities from blockchain IDs, known as addresses. When an address would be added to the blockchain, an identification issuing service would bind a public key by default and transfer the ownership of the corresponding private key to the entity. This way, an entity is able to sign a message, which can be validated with a public key stored in the blockchain. Hence, it would serve as a single sign-on portal where the application would need to only request a digital signature and the ID from the entity that is requesting access to a certain service. The proposed system in this paper is inspired by an idea proposed by Cresitello-Dittmar of storing certain ID information of an entity on the blockchain. While it is a good idea to store non-sensitive information on the blockchain, it should be understood that stored data must be limited in size, because it is replicated in each network node and will impact the overall performance of the blockchain authentication and authorization protocol.

The use of blockchains for medical data access was demonstrated in a system called MedRec [9]. This decentralized system handles EHRs of patients and offers a possibility to retrieve their medical information from different service providers. The system is built on the Ethereum platform using two different incentives for medical stakeholders to participate in a blockchain network. The first incentive is the cryptocurrency Ether itself, which is needed to execute transactions in an Ethereum network. Ether coins need to be either purchased from the cryptocurrency market by the patients or service providers. The second incentive comes from the aggregated anonymized medical data. These medical data are essential for research in the medical industry. The system works on a principle of smart contracts, which contain metadata about the ownership of records, data integrity, and permissions. In contrast to MedRec, our work uses a consortium blockchain technology based on the HyperLedger Fabric platform. This circumvents the need for providing incentives from e.g., mining, which is a consequence of the Proof of Work (POW) algorithm in Ethereum. The incentive in the network is not physical or virtual. Rather, it is more like a "moral obligation". An institution that wants to increase privacy and security of transactions can do so by joining the network. This may likely result in increased credibility in the eyes of end-users. In contrast to MedRec, our system is not limited by a potentially high price of a cryptocurrency i.e., Ether.

Scalability is a main concern of blockchain technology [14]. Li et al. [14] proposed a new blockchain architecture using a set of interconnected but independent subchains of a single block-chain system to improve the scaling properties of their system. Their design is based on the Hyperledger Fabric permission concept [15]. It demands that all prospective blockchain members register and acquire an identity before connecting to the network and submitting transactions. This concept is overall appealing and can likely be used for scaling our proposed system. Nevertheless, the requirements for scalability of our system is far less stringent compared to a blockchain system supporting a cryptocurrency and we clarify how our system can scale to the handling of EHRs in the Danish healthcare system in Sec. VI.

III. Blockchain frameworks

The blockchain intrinsically relies on distributed consensus and trust. However, obtaining distributed consensus is a difficult computer science problem itself [16] and has been a subject for research in blockchain technology [1]. To achieve distributed consensus, nodes individually store the blockchain, referred to as the *ledger*, containing the transactions that have been approved in the network by other nodes. Consensus is achieved on a block-level to avoid inefficiencies by rigorously flooding a network with broadcast transactions. Each time a node wants to append a block of data to the blockchain, consensus must be reached among a majority of nodes.

A simple blockchain data structure is presented in Fig. 1. The structure consists of hash pointers $H(x_i)$ used to reference other blocks while also providing means for integrity checking. By cryptographically linking blocks B_i, an immutable infor-

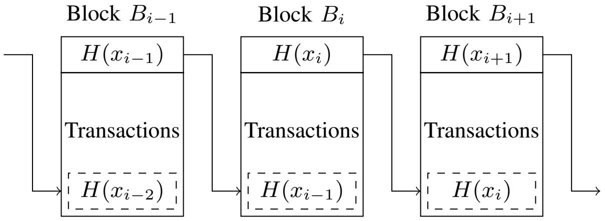


Fig. 1: Section of a simple blockchain structure.

mation record can be created. Each node in the blockchain network not only stores the chain of blocks, that the network agreed upon, but it also needs a block of outstanding transactions, which the node heard about. When consensus is reached and a given block is chosen, it will be inserted in the blockchain together with the transactions, which the node has validated. This step may require successfully solving the POW depending on the type of blockchain in use. For a network to work properly, there must be nodes continuously validating and appending blocks to the ledger. This can be achieved by providing an incentive for the participating nodes such as the potential gain of cryptocurrency.

A. Blockchain Types

While the objective of most blockchain networks is to be open, transparent, decentralized, and without a central authority, there are cases where openness is less desirable. These cases include, for example, financial institutions, logistics supply chains, and the healthcare industry where only a certain group of entities needs to interact. To address these diverse requirements, different types of blockchains exist. These differ by the degree of control and restrictions of members.

The *public blockchain* is open for anyone to join the network. It is transparent, the consensus process is open, and all transactions are visible. Public blockchains are secured by a combination of incentives and cryptographic validation, i.e., POW. Transactions are validated and synchronized within the network before they are appended to the blockchain. Each node contains a copy of the ledger. This redundancy makes it highly secured, but it also increases the cost of transactions and decreases the speed of processing. In the *private blockchain*, the permission to write to the blockchain is controlled by a central authority. Read permissions can be public or restricted to a certain extent allowing a higher degree of privacy compared to public blockchains. The centralization of write permissions increases the efficiency and speed of processing transactions. The *consortium blockchain* is partly private and transactions can optionally be public. The consensus protocol is controlled by pre-selected members such as healthcare institutions. This results in a partially decentralized blockchain.

The consortium blockchain offers a hybrid between the relative low trust provided by the public blockchain and the centralized trusted entity model of the private blockchain. The private blockchain more precisely describes a traditional centralized system with a high degree of cryptographic auditability [17]. The analogy of council of trusted entities can be used for the consortium blockchain. The public blockchain is primarily designed to eliminate an intermediary, maintain the anonymity of users, and having transactions publicly visible. This is less suited for healthcare institutions. Rather, private or consortium blockchains should be used where the possibility of controlling the data privacy is possible. Our prototype uses the consortium blockchain framework Hyperledger Fabric, which fits our use case requirements. The Hyperledger Fabric provides a business-oriented private permissioned blockchain with smart contract support. The Hyperledger Fabric is different than other blockchain frameworks because it strictly orders transactions within blocks. In other blockchains, the same transaction may occur in multiple different blocks. Blocks generated in Hyperledger Fabric are said to be final because once a transaction has been written into a block, its position in the ledger is assured to be immutable. This means situations such as forking chains cannot occur.

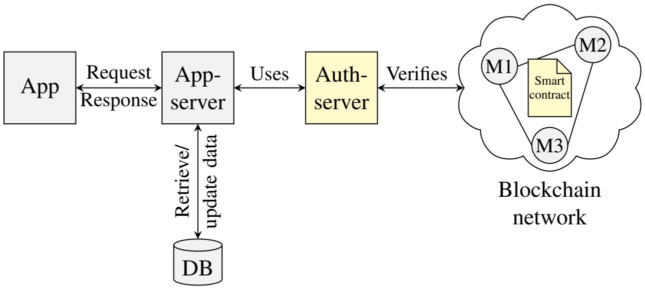


Fig. 2: Architecture of the identity and access management system using a blockchain with smart contracts.

B. Smart Contracts

Smart contracts are user-defined rules governing transactions [7]. Smart contracts can be thought of as digitized variants of agreements that enforces certain rules for transactions by a network of peers based on defined policies [18], [19]. A smart contract is a set of agreed promises including the protocols specifying how parties deliver on these promises. A smart contract is converted to a computer code, stored, and replicated in the blockchain. It can be made fully or partially self-enforcing and self-executing.

When the first smart contract is created, i.e., the top-level contract, all members who participate agree to follow these rules. The top-level contract contains methods, which can be executed by members of the network. In our case, the smart contract contains rules for authentication and authorization. For instance, when a user tries to execute certain functionality, and has the permission to do so, the validation process queries the blockchain world state for permission states and subsequently performs the needed operations. While subcontracts might be created, all future contracts are following the rules of a top-level contract.

IV. System design

The conceptual architecture of the proposed identity and access management system is shown in Fig. 2. It consists of a client application (App), an application server (App-server), a Database (DB), and an authentication and authorization server (Auth-server). The system authenticates and authorizes users by validating transactions in the blockchain network. In a simple scenario, the App communicates with the App-server, which authenticates users by credentials stored in the DB. After the user has been authenticated successfully, the App-server performs the desired operation such as retrieving or updating the data in the database. In this case, security may be enhanced by communication with authentication server such as OAuth 2.0 [20].

The user interacts with the system through the App. The App can be a web, a mobile, or a standalone application that communicates with the App-server to perform the desired functions. This operation requires users to be authenticated and authorized. The App-server assists the client App and delivers the requested data. For a server to deliver the data, it needs to communicate with the DB and the Auth-server. The App-server delegates work to the Auth-server to validate the authenticity of the user and to authorize access to the DB. The Auth-server acts as an intermediary between the App-server and the blockchain network. It authenticates and authorizes the user by querying the blockchain network. Additionally, it can invoke methods and update the world state. The Auth-server is the only component able to interact with the blockchain network and it delivers the result to the application server. The participating nodes (i.e., M1...M3 in Fig. 2) keep the network running and maintain the ledger. The nodes follow the rules in the smart contract. The nodes validate and broadcast transactions and run the consensus protocol. The purpose of separating the Auth-server from the App-server is to decouple the logic and enable modularity. It enables having multiple application servers performing different functionality accessing different databases while still accessing the same Auth-server, if any authentication and authorization is needed. An off-blockchain DB is used in the system to limit the amount of data in the blockchain as much as possible since it is replicated to members of the network and additional data is stored in the DB. For an App-server to access the database it requires a permission validation in the blockchain.

V. Implementation

For simplicity, the implemented blockchain consists of a single organization. This organization contains one peer (validating node) and one “solo” ordering node with one public channel available for communication for the registered participants in accordance with the Fabcar example network [15]. It consists of a single peer node configured to use CouchDB as the world state database, a single “solo” ordering node, a CA and a command line interface container for executing commands. The prototype is implemented in Javascript and uses the Hyperledger Fabric v1.1.0-preview that supports Node.js. In the following, we will address the essential parts of our system implementation.

A. Front-end Subsystem

The system front-end consists of two layers: *i*) the presentation layer and *ii*) the presentation logic layer. The presentation layer is what an actor sees and can interact with. In contrast, the presentation logic layer is not visible to the actor. It is responsible for handling events and transforming data but also for communication with the back-end. It makes requests to the server exposed Application Programming Interface (API).

The front-end is built using AngularJS and follows the Model/View/Controller pattern [21]. It is composed of a single web page application, consisting of one view and controller associated with it. The View binds to the data in the controller and displays the information needed. The front-end is responsible for presenting the data necessary to the actor.

Fig. 3 shows examples of two views of the web application. In the first view, a user may grant permission to a healthcare actor e.g., a peer doctor colleague for a particular patient. The responsible medical actor selects a subject ID ① and enters the medical ID of another actor that should be given access rights ②. The type of permission is selected ③ and granted ④ thereby starting interactions with the blockchain. The second view allows an actor to revoke permissions granted to a healthcare actor. An already granted permission is revoked from a single action ⑤ that triggers the interaction with the blockchain.

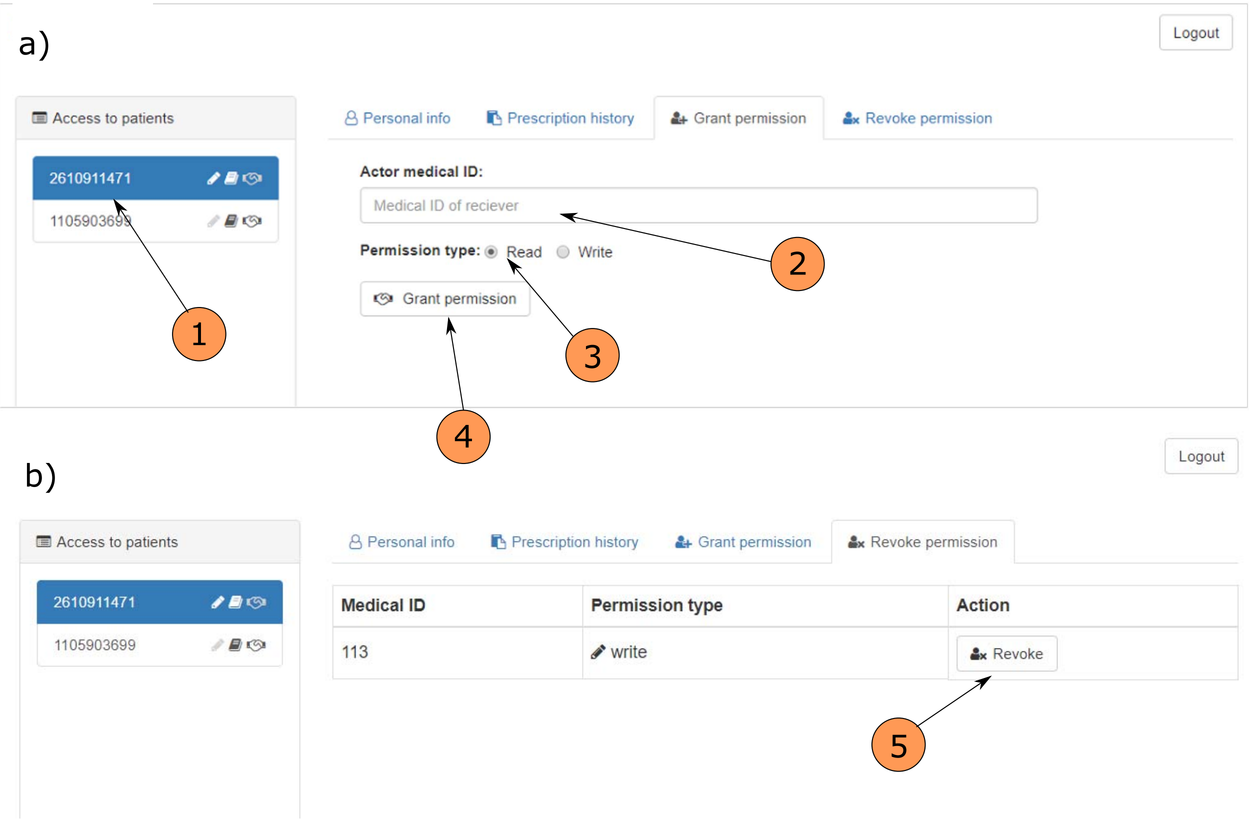
B. Back-end Subsystem

The back-end consists of a single server, which has configured routes and these routes are exposed and can be called by clients. Each route has associated a corresponding method with it. These methods use the Hyperledger Fabric NodeSDK to communicate with the blockchain network. The back-end is built using NodeJS and NodeSDK. It is responsible for handling client requests and returning the response from the blockchain network.

To access the database through the system, there is a need for an authorization to access the blockchain network. This is accomplished by using JavaScript Object Notation (JSON) web token created during the login process. The database used in our prototype is PostgreSQL. This part of the system is based on an architecture from some of our previous work [22]. For simplicity, it contains only sample data, used in the system, separated into three different tables: *i*) ‘actors’ containing a data about healthcare personnel which interact with the system, *ii*) ‘patients’ containing personal data about the patients, and *iii*) ‘prescriptions’ describing the history of prescriptions for patients prescribed by doctors.

The structure of the network is based on an example from Hyperledger Fabric’s simple test network i.e., the Fabcar network. It builds only on the necessary components to run the application. This network uses a single peer node with world state database, single ordering node, and a CA. When the network is started, it creates a channel connecting a peer node to install a smart contract (a custom one) on a peer’s file system and executes the initialization function on that contract.

The App-server interacts with the blockchain network using APIs to invoke functions of the smart contract. The APIs are accessible within the Hyperledger Fabric NodeSDK and APIs include communication with the smart contract by running queries and receiving updates from the ledger. For a server to interact with a blockchain through the API, it needs to enroll an admin identity [15]. The admin identity is acting like a registrar, issuing registration and enrollment calls for new users to be registered within the blockchain network. Whenever a user (i.e., an identity) interacts with the App, the App-server creates an identity on the blockchain from which subsequent calls will be made to invoke functions of the smart contract.



Medical ID	Permission type	Action
113	write	Revoke

Fig. 3: Example of functional tabs of the web application. a) shows a user interface of the grant permission function and b) shows the user interface of the revoke function.

Listing 1: Example of object/entry structure in a smart contract.

```
"Key": "hash of public key",
"Record": {
   "actor": "actor medical Id",
   "subjects": [
      {"id":"subject1Id", "write": true,
          "granted": true},
      {"id":"subject2Id", "write": false,
          "granted": false}
   ]},
   "granted": [
      {"receiver":"actor1Id",
          "subjectId":"subject1Id",
          "permissionType":"read"},
      {"receiver":"actor2Id",
          "subjectId":"subject1Id",
          "permissionType":"write"}
   ]}
```

C. Key Functions and Objects

Transactions on the blockchain are stored as key-value pairs in an object/entry structure shown in Listing 1. The `Key` is a unique identifier of an entry and contains a hash of a public key of an actor. A `Record` is associated with a key and contains an actor and a list of subjects and granted permissions. The actor has an associated unique number assigned e.g., a medical ID from the medical institution, which is defined in the blockchain during the registration process. `Subjects` contain lists of subject IDs and permissions to which an associated actor has access to. The subject ID is a unique identifier of a patient. This may be represented by a Personal Identification Number (PIN) number. The write permission specifies whether an actor can change the data of a subject. The `granted` list keeps track of permissions granted and the ability to revoke when needed. Each entry in a list contains a receiver (medical identity of an actor, i.e., the actor ID), subject ID (a PIN of a patient), and permission type, i.e., either read or write.

The two most essential functions to modify the object/entry structures of the blockchain are the *queryPermission()* and the *grantPermission()* functions. The *queryPermission()* function returns a 'permission object' for specified ID shown in Listing 2 and Listing 3, respectively. The code first checks for a correct number of arguments, i.e., a hash of a public key (Listing 2, lines 2-3). Then the code checks the world state for that permission with the particular ID (lines 7-10). If the ID does not exist, an error is thrown (line 11). Otherwise, the code returns the 'permission object' for the requested ID to the server (line 13).

The *grantPermission()* function updates the granted list of the fromUser as well as the subject list of the toUser. The code checks for the number of arguments (Listing 3, lines 2-

Listing 2: Query permission function.

```
async queryPermission(stub, args) {
   if (args.length != 1) {
      throw new Error('Incorrect number of
          arguments. Expecting permissionID');
   }
   let permissionId = args[0];

   let permissionInBytes =
         await stub.getState(permissionId);
   if (!permissionInBytes ||
         permissionInBytes.toString().length <=
             0) {
      throw new Error(permissionId + ' does not
          exist: ');
   }
   return permissionInBytes;
}
```

Listing 3: Grant permission function.

```
1  async grantPermission(stub, args) {
2     if (args.length != 4) {
3        throw new Error('Grant permission:
             Incorrect number of arguments.');
4     }
5     let fromUserId = args[0];
6     let toUserId = args[1];
7     let patientId = args[2];
8     let canWrite = args[3];
9
10    let fromUser = JSON.parse(await
          stub.getState(fromUserId));
11    let toUser = JSON.parse(await
          stub.getState(toUserId));
12
13    if(!fromUser || !toUser){
14       throw new Error('One of the user were not
             found!');
15    }
16    for(let i = 0; i < fromUser.subject.length;
17       i++){
18          if(fromUser.subject[i].id
19             === patientId &&
20             fromUser.subject[i].granted
21             == 'true'){
22             throw new Error('Permission cannot
23                be granted further');
24          }
25    }
26    for(let i = 0; i < toUser.subject.length;
27       i++){
28          if(toUser.subject[i].id === patientId)
29             throw new Error('Patient already
30                assigned to ' + toUserId);
31          }
32    fromUser.granted.push({receiver:
          toUser.actor, patientId: patientId,
          permissionType: (canWrite === 'true' ?
          'write' : 'read')});
33    toUser.subject.push({id: patientId, write:
          canWrite, granted: 'true'});
34
35    await stub.putState(fromUserId,
          Buffer.from(JSON.stringify(fromUser)));
36    await stub.putState(toUserId,
          Buffer.from(JSON.stringify(toUser)));
37 }
```

4). The expected arguments are: *i*) the user giving permission (fromUser), *ii*) the user receiving permission (toUser), *iii*) the PIN of a patient, and *iv*) the type of permission (read or write). The code then retrieves the permission objects from the world state from both fromUser and toUser (lines 10-11). If neither of the users exists an error is thrown (line 14). Hereafter, the code checks to see if fromUser is authorized to grant permission further. The code continues and ensures that the toUser does not already have that patient assigned (lines 16-25). If not, then the blockchain will be appended with a new state, meaning the fromUser will update its granted list and toUser will update its subjects list (lines 32-36).

The entire code of the prototype is available as open source software under the Apache 2.0 license. It is available from the URL: https://github.com/Sharky1231/blockchainPrototype.

VI. Results and Validation

To validate the functionality and assess the performance of our prototype, a number of experiments have been made. These experiments furthermore improve the understanding of the usability of the blockchain in real world applications.

Experiments are performed on a server hosted locally on a machine with an Intel Core i7 processor running with a 2.4 GHz clock speed, 8 GB memory, and 1 TB SSD storage. Experiments are performed on blockchain sizes of $N \in \{1, 100, 500, 2000, 4000\}$ entries and 50 measurements are acquired for each test. Testing was performed in the Chrome web browser using 10 simultaneous requests. The results are listed in Table I and the average values are plotted in Fig. 4. The initial blockchain size ($N = 1$) was observed to have the size 19 KB. As the number of entries in the blockchain increases, the overall size of the blockchain increases as well (cf. Table I). The average query time with 1 entry in the blockchain takes 54 ms and the invoke time with 1 entry takes on average 2196 ms. The scaling of the experiments was limited by the local computer's memory resources. While the initial blockchain size is small, the limiting factor was the amount of RAM needed to simultaneously run Docker, PostgresSQL, and the actual blockchain network.

The query time of the blockchain is comparable to the response time of login and update data functions of the

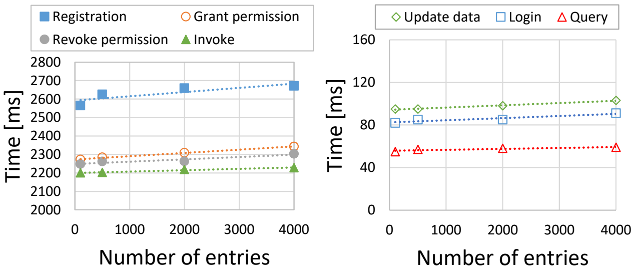


Fig. 4: Performance measurements. Dashed lines show the linear regression trend lines.

TABLE I: Performance measurements for different activities. N specifies the blockchain size in a number of entries.

Blockchain function	Mean values ± one standard deviation [ms]			
	$N = 100$	$N = 500$	$N = 2000$	$N = 4000$
Registration	2565 ± 75	2625 ± 91	2659 ± 75	2671 ± 62
Grant permission	2272 ± 15	2284 ± 16	2309 ± 17	2343 ± 14
Revoke permission	2248 ± 19	2262 ± 20	2263 ± 21	2303 ± 27
Update data	95 ± 4.4	95 ± 4.3	98 ± 4.6	103 ± 5.7
Login	82 ± 5.3	85 ± 5.3	85 ± 5.8	91 ± 6.3
Invoke	2200 ± 16	2202 ± 14	2218 ± 12	2228 ± 23
Query	55 ± 2.1	57 ± 1.7	58 ± 2.0	59 ± 2.3
Blockchain size [kB]	38	112	392	767

system. Registration, grant permission, and revoke permission functions have response time closely related to the invoke time. The reason is that login and update data functions are querying the blockchain. In case of a data update, the blockchain is queried to check whether the actor has permission to access the specific subject. In the case of registration, the data needs to be inserted in the blockchain and the invoke function must be executed. Another thing to notice is that with increasing number of entries in the blockchain, the response time of querying or invoking the blockchain, and functions of the system, such as registration, grant permission or login, is increasing slightly.

VII. Discussion

The problem, well known for blockchains, is scalability [14]. According to Eurostat healthcare personnel statistics made in 2015 [23], there are in total 20,639 physicians in Denmark. Assuming a linear scaling up to this figure, we estimate that the response time of a query of the blockchain would increase with about 46%, which would correspond to a query response time of 86 ms. Furthermore, the invoke time would increase with about 7.5%, which would correspond to a response time of approximately 2.4 seconds. While the initial blockchain size increased by a factor of 40, using this metric, size would be expected to increase by a factor of approximately 200. This would result in an initial blockchain size of about 3.8 MB, holding all physicians in Denmark.

In 1994, MedCom introduced a system with the purpose to establish a communication standard for a messaging in the Danish healthcare industry, including public hospitals, general practitioners as well as private companies in the sector [11]. Messages in the system consist of healthcare related data, which is text-based and include for instance prescriptions or discharge letters. According to the overall traffic monitoring report of MedCom [24] from September 2017, there is an increasing number of messages sent each year and the system is heavily used. Traffic monitoring of hospital messages, the number of hospital messages sent in 2016 is around 3,000,000 per month [24]. This equals to 1.12 messages per second. However, the aim of our prototype is not to replace or reinvent the current solution but rather to advance the existing solution by the introduction of blockchain technology to provide identity and access management.

According to the Hyperledger Fabric documentation [15], the expected performance goal is to handle 100,000 transactions per second in a production environment with 15 validating nodes. This is more than enough to handle the number of messages sent per second needed in an up-scaled system with our design also taking into account that the prototype does not only use a blockchain to grant or revoke permission but also utilizes it for registration, login, and update of patient data. Even if the number of transactions per second would increase by a factor of 100, it is still below the expected performance goal of the Hyperledger Fabric.

VIII. Conclusions

In this paper, we proposed a system for identity and access management using blockchain. A prototype based on an open source community blockchain framework called Hyperledger Fabric was made to demonstrate the feasibility of such a system. It confirms that identity and access management can be achieved in a decentralized, efficient, and secure manner. The proposed concept was applied and discussed in a use case with EHRs from the healthcare sector. We find that it is feasible to deploy a blockchain network capable of supporting all physicians in Denmark with an initial blockchain size of about 3.8 MB and with serving response times for functions to interact with the blockchain on the order of 2-3 seconds. Future work includes the transition of the prototype to our data center and the demonstration of the upscaling of the system by allocation of architectural component to different server instances.

References

[1] J. A. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol: Analysis and applications." in *EUROCRYPT (2)*, 2015, pp. 281–310.

[2] A. Shamir, "Identity-based cryptosystems and signature schemes," in *Proceedings of CRYPTO 84 on Advances in Cryptology*. New York, NY, USA: Springer-Verlag New York, Inc, 1985, pp. 47–53.

[3] E. C. Ferrer, "The blockchain: a new framework for robotic swarm systems," *arXiv preprint arXiv:1608.00695*, 2016.

[4] R. L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-key Cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[5] R. H. Jacobsen, S. A. Mikkelsen, and N. H. Rasmussen, "Towards the Use of Pairing-Based Cryptography for Resource-Constrained Home Area Networks," in *2015 Euromicro Conference on Digital System Design*, 2015, pp. 233–240.

[6] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts," in *2016 IEEE Symposium on Security and Privacy*, 2016, pp. 839–858.

[7] K. Delmolino, M. Arnett, A. Kosba, A. Miller, and E. Shi, "Step by Step Towards Creating a Safe Smart Contract: Lessons and Insights from a Cryptocurrency Lab," in *Financial Cryptography and Data Security*, J. Clark, S. Meiklejohn, P. Y. A. Ryan, D. Wallach, M. Brenner, and K. Rohloff, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 79–94.

[8] N. Atzei, M. Bartoletti, and T. Cimoli, "A Survey of Attacks on Ethereum Smart Contracts (SoK)," in *Principles of Security and Trust*, M. Maffei and M. Ryan, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2017, pp. 164–186.

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

[10] "Hyperledger," url: https://www.hyperledger.org/projects/fabric. [Online]. Available: https://www.hyperledger.org/projects/fabric

[11] "eHealth in Denmark; eHealth as a part of a coherent Danish health care system," Danish Ministry of Health, Tech. Rep., April 2012 2012. [Online]. Available: https://www.sum.dk/ /media/Filer%20-%20Publikationer_i_pdf/2012/Sundheds-IT/Sundheds_IT_juni_web.ashx

[12] G. Zyskind, O. Nathan, and A. Pentland, "Decentralizing Privacy: Using Blockchain to Protect Personal Data," in *2015 IEEE Security and Privacy Workshops*, May 2015, pp. 180–184.

[13] B. Cresitello-Dittmar, "Application of the Blockchain For Authentication and Verification of Identity," Tech. Rep., 2016, available: http://www.cs.tufts.edu/comp/116/archive/fall2016/bcresitellodittmar.pdf.

[14] W. Li, A. Sforzin, S. Fedorov, and G. O. Karame, "Towards Scalable and Private Industrial Blockchains," in *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts*, ser. BCC '17. New York, NY, USA: ACM, 2017, pp. 9–14.

[15] "HyperLedger Fabric Documentation," 2018, url: https://media.readthedocs.org/pdf/hyperledger-fabric/latest/hyperledger-fabric.pdf.

[16] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *Journal of the ACM (JACM)*, vol. 32, no. 2, pp. 374–382, 1985.

[17] "Vitalik Buterin, On Public and Private Blockchains," August 7th, 2015. 2015, available: https://blog.ethereum.org/2015/08/07/on-public-and-private-blockchains/.

[18] N. Szabo, "Smart Contracts: Building Blocks for Digital Markets," 1996. [Online]. Available: http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.2.html

[19] K. Christidis and M. Devetsikiotis, "Blockchains and Smart Contracts for the Internet of Things," *IEEE Access*, vol. 4, pp. 2292–2303, 2016.

[20] D. Hardt, "The OAuth 2.0 Authorization Framework," oct 2012, IETF RFC 6749. [Online]. Available: https://rfc-editor.org/rfc/rfc6749.txt

[21] A. Leff and J. T. Rayfield, "Web-application development using the Model/View/Controller design pattern," in *Proceedings Fifth IEEE International Enterprise Distributed Object Computing Conference*, 2001, pp. 118–127.

[22] S. A. Mikkelsen, R. H. Jacobsen, and A. F. Terkelsen, "DB&A: An Open Source Web Service for Meter Data Management," in *2016 IEEE Symposium on Service-Oriented System Engineering (SOSE)*, March 2016, pp. 4–13.

[23] "Healthcare Statistics. Eurostat Statistics Explained," 2016, url: http://ec.europa.eu/eurostat/statistics-explained/index.php/Healthcare_statistics.

[24] "Overall traffic monitoring 1994 – 2017; Principal figures for traffic development per month for hospitals, General Practitioners and municipalities," December 2017 2017, url: https://medcom.medware.dk/exports/medcom_monitorering_en.pdf.