EasyChair Preprint

№ 1291

Privacy-Preserving Data Sharing on Multi-Layer Blockchain: Case Study on Healthcare

Wei-Chen Lin

July 15, 2019

Privacy-Preserving Data Sharing on Multi-Layer Blockchain: Case Study on Healthcare

Wei-Chen Lin, Department of Electrical Engineering National Taiwan University

***Abstract*—This thesis designs a data exchange system architecture and applies it to healthcare cases, and will solve the following two problems. First, children forgets whether the type vaccines that has been vaccinated or not. Then, he/she has to vaccinate again in case. Second, we implement the medical record exchange between hospitals. The same concept can also be applied to the following situations. If someone sees a doctor in different hospitals, the medicines from different hospital may have drug interactions, which may cause side effects after taking them at the same time. Doctors can avoid this situation with the information through the blockchain. Recently, the applications of the medical blockchain are mostly implemented by Ethereum. This thesis, unlike other Ethereum applications, also proposed a new multi-layer blockchain architecture and replaced the consensus mechanism in order to speed up and make the system more reasonable in use.**

Keywords—Blockchain, Tendermint, Ethereum, Multi-Layers, Smart Contract, Data Exchange

I. INTRODUCTION

With the immutability and consistency of blockchain, any records of data exchange in our system will upload to the blockchain as evidence. Afterwards, if someone in the system deceives that he/she has not exchanged any information, the dispute can be audited and trailed through the blockchain. The blockchain represents a fair third party with all records that can be reviewed, and these records cannot be tampered or be deleted. That is to say, it has absolute credibility.

This thesis designs a data exchange system architecture and applies it to healthcare cases to deal with the following two cases. First, children's vaccination was recorded on a small yellow card. If someone was vaccinated at school, without creating a record file in hospital or CDC (Centers for Disease Control), and the small yellow card was gone unfortunately, he/she will forget whether the type vaccines that has been vaccinated after few years. Then, he/she has to vaccinate again in case. Second, if we need to transfer or apply for insurance, which medical records are required, it is necessary to go through the relevant procedures in person or by the agent (with consent). And then, we have to pay the cost of printing the medical records. The medical records transferring between hospitals should not be so complex and time-wasting. Some patient will also concern that why they should pay again for the medical records.

For the convenience of smart contract, the medical applications on blockchain are mostly implemented by Ethereum. You can deploy smart contracts to the blockchain and execute contract changing state or querying data from blockchain. With the smart contract, blockchain go more easily using and more potential. The most important is that all the action will be recorded and will not be eliminated. In despite of that Ethereum has its credibility, the speed of sending transaction on Ethereum is not efficient enough. In real case, it might not be scalable and reasonable. Therefore, this thesis also proposed a new multi-layer blockchain architecture remaining the credibility of public chain, and improving the efficiency.

II. RELATED WORK

A. Evmlite with Tendermint

Evmlite [1] is an open source program. It separates the Ethereum Virtual Machine (EVM) from the Ethereum, and the system can be executed. The client can call several APIs of the EVM, including Get controlled accounts, Get any account. Send transactions from controlled accounts, Get Transaction receipt, Send raw signed transactions, Call contract without state change. Using these APIs lets service to receive message first, and then send them to the consensus engine for execution. Once the node reaching a consensus, it will return the results to the EVM and execute contract or API content to change the state of the database. Using Evmlite allows solidity's contract to be executed without Ethereum, and can choose a consensus mechanism you need. At present, the implementation of Evmlite has three kinds of consensus: solo, babble, and RAFT.

B. Related Work and Design Consideration

There are many prototypes for medical applications on Blockchain. After some paper surveys, and discussions with HTC DeepQ, we have designed a system architecture for data exchange application. The design concepts of this system architecture on blockchain are as follows.

First of all, we must return the authority to exchange medical records to the patient and the owner of the medical record. William J.Gordon [2] compared three different healthcare interoperability, one of which is blockchain-enabled patient-driven interoperability. The patient can retrieve data directly from hospitals, and patient can also authorize sharing of medical records to other hospitals that don’t have a formal relationship between hospitals and hospitals.

On the blockchain layer, it stores authorization rules. We use the concept of patients and smart contract-driven to design the workflow and the logic of contract. As the European Union proposed the General Data Protection Regulation (GDPR), all personal data mentioned in GDPR should be properly protected. People wouldn’t want a company to hold onto their information because it is likely that it can be leaked or hacked. This is why they can request the company to forget their personal information, in other words, delete it.

In order to follow GDPR, Dubovitskaya [3] and Gropper [4] focused on this topic and proposed their architectures to avoid personal data problem. Dubovitskaya used messages on the blockchain, created by patient IDs to identify individuals. The message is encrypted. However, his medical records claim to be placed in a secure cloud access control system, and the access control is managed by the smart contract in Blockchain.

Gropper allows the patients to possess digital wallets that allow them to not be identified on the blockchain. Instead of

the patient's ID, Gropper's system would record only the blockchain-based ID for securing and managing access control. Both of the above can reach GDPR standard.

Nevertheless, Dubovitskaya puts the medical record in the cloud, even if he claims it is a safe cloud, it is still considered very dangerous to put personal data on a cloud system. Gropper's system may encounter the risk of user using the wallet at will and being multiple registered by the same user.

Andrew Lippman [5] proposed medrec architecture. There are two points that inspired me, the concept of smart contract to manage workflow and keeping data in the local database. Nevertheless, medrec still has some aspects to improve. They didn't consider the GDPR and put patient personal data directly on the blockchain. In addition, they didn't divide patient and provider into two categories, so the patient and the provider are actually in the same role in this system. A patient has the same accessibility as a provider, so if I were a patient, I can pass my electronic medical records to others at will. And the provider can create or share medical records to other providers if they want.

In summary, our system design takes the advantages of the above systems and revised their disadvantages. We designed a role-based contract to manage access control and divide different roles to achieve their demand respectively. This design will be discussed in the next Chapter.

III. System Design Overview

A. Scenario

Our data exchange system (DeepLinQ) is mainly designed to solve the following two problems. The first scenario is that children will be recorded on the small yellow card when they are vaccinated at school, without record in the hospital or CDC. Even if there is a record in hospitals or CDCs, people can't directly access their database. And, as long as the small yellow card is gone, we lose the information on small yellow card. In case, you will vaccinate again, if you are not sure whether you have been vaccinated. In order to make the vaccine record easy to obtain, we apply the blockchain technology to make the vaccine information more accessible.

The second scenario is that when we need transferring from hospital A to hospital B or insurance claiming (third parties), we need the proof of medical records. At present, we must go to the hospital in person or ask the agent with the consent to do it. Moreover, you have to pay the cost of the work and paper. However, if the patient has to prove his/her medical record to other hospitals or to claim insurance, it should not need the hospital's consent.

As William J. Gordon said, the owner of the data should be a patient, so if the patient wants to get his/her record, the process should not go through hospital's agreement. Namely, if the patient agrees that his/her medical record sharing to other legal unit. The blockchain is a good environment for patients to easily obtain the metadata of their medical records. The record on blockchain also let hospitals or the insurance industry to believe the information on the blockchain is indeed the information written by the original hospital. With such credibility, the information on blockchain does not need to be questioned.

What we need to implement is a data exchange system that uses the immutability and consistency of the blockchain to ensure the credibility of the data. No one can tamper with the records on the blockchain. The decentralized system can also guarantee the correctness of those data. We can program on smart contract to manage workflows, access control to data on the blockchain, and the relationship between patients and hospitals through the easy-to-develop property of smart contracts.

When the patient sees a doctor in hospital A, hospital A will store a medical record of him medical record to the hospital local database, and save some metadata to the blockchain and the patient will know that he has been added a medical record. The medical record can be agreed, by patient, to share the selected medical record content to other hospital of third party facility through the patient's device. For example, after the patient agreeing to share the medical record with hospital B through the smart contract. Hospital B will send a data query to Hospital A. Then, hospital A would check permission on the blockchains, and eventually, the medical record will be transmitted to the hospital B database.

B. System Architecture

In this part, I want to introduce our system architecture. This multi-layer architecture can be divided into three layers as shown in Fig. 1:

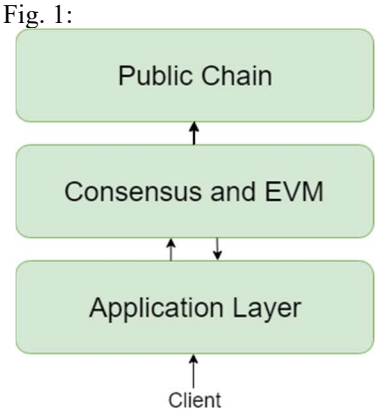

Fig 1. Multi-layers System Structure Architecture

1) *First Layer: Application Layer:* The first layer is the application layer, which mainly designs the client interface and figures out how to interact with the smart contract on the blockchain. It includes how to implement the situation in workflow with web pages, how to design the smart contract content to define the relationship between patient and doctor, and design what functions they should implement in the system related to their character to manage access control. The client webpages in the application layer are sort into patient registration webpage, hospital registration webpage, hospital registration approving webpage (CDC), doctor user interface webpage, and patient user interface with sharing medical record webpage. The functions of these webpages will interact with the smart contract to complete the workflow. The workflow includes medical examination and exchanging medical records and recorded them on the blockchain.

2) *Second Layer: Consensus and EVM Layer:* The second layer is the consensus and the EVM layer. Why do we have this layer? Because if you use Ethereum directly in this system, you will encounter the problem of scalability. Ethereum generates anew

block with an average speed 15 seconds. This is not acceptable for practical applications.

Nevertheless, at the beginning, our system is implemented by using the API of web3.js which let the system connecting directly to the private chain of Ethereum, it is found that if a medical record is sent out and the state is changed by calling a complicated smart contract, it takes about 30 seconds for the EVM to execute bytecode and return messages to the application layer. If the system is used in a real hospital, there are dozens of computers in different division sending medical records to the blockchain at the same time. There will be a lot of transactions which is in a pending state, or even the transactions will fail.

To solve this problem, we need to use a faster consensus algorithm, in which Tendermint is a consensus algorithm that is both fast and can solve the problem of Byzantine fault tolerance. Tendermint is also easy to develop because of the part of ABCI. Developers can use any programming language to process transactions, but there is no way to use these smart contracts like Ethereum with Tendermint. We found an open source program called evmlite, which separates the EVM from Ethereum. In their words, the EVM can reach a consensus with different consensus modules with retaining the smart contract function for application layer use. By changing consensus, it also increases the speed generating new blocks and reach consensus.

3) *Third Layer: Public Chain Layer:* Some people may question the credibility of the second layer, private chain. Therefore we propose the third layer architecture. The system can regularly collect the second layer, Tendermint, block data. In the way of Merkle Tree, it accumulate a certain amount and then get the hash value upload to the public chain. The Merkle Tree method can save the cost of the public chain, and also allows the most credible public chain to help record the information of the private blockchain.

C. *Contract Design Overview*

Smart contracts are the programs that exist on the blockchain network [6]. Once a contract has been deployed, the blockchain will assign an address to record the encoded contract data, binary code, and this record cannot be revised in the future. An authorized user in this system can invoke contracts through the website interface with gas for consuming the computing power and storage of the network. By interacting with smart contracts in our system, patients can share the authentic data of their own Electronic Health Records (EHRs) to an authorized hospital. And, the contracts manage patient's relationship with hospitals or third parties.

The functions in the workflows will be completed by executing the following three types of smart contracts on blockchain, entity contracts (EC), relationship contracts (RLC), and data contracts (DC). The concept of these smart contracts is motivated by MedRec [5], but our system (DeepLinQ) makes these smart contracts more general and more systematic. What's more, we consider more situations that the system may encounter. For example, the personal data that GDPR concerns, access control problems that someone may share medical records at will to his/her friends, or hospital secretly create a fake records to another hospital or third party. Our design is more complete concern, and that entity, relationship, and data contract can be used beyond just supporting the healthcare domain.

In contract overview Fig. 2, we have designed a role-based contract architecture, and you can see how the contact is composed in the figure. At the very beginning, the administrator creates two Entity Contracts (ECs). EC contains the patient and hospital basic data and the address of Relation Contract (RLC). RLC contains the address of Medical Data Contract (MDC). And, MDC contains metadata of electronic health records (EHR).

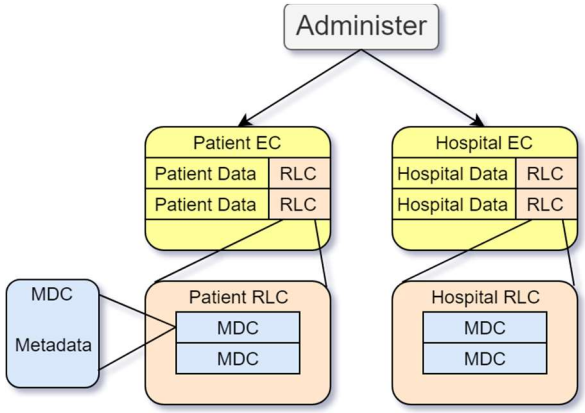

Fig 2. Contract Design Overview

1) *Entity Contract (EC):* Entity Contract (EC). In Fig.3, we apply role-based contract design to divide EC contracts into two types, which are patient EC and hospital EC. Because that patient personal data should be well-protected, we should deal with the data to be anonymous. And, the hospital, a public and transparent institution, should keep data public on blockchain.

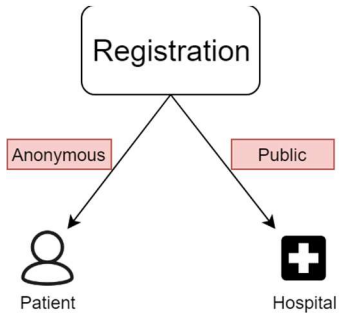

Fig 3. Role-based Entity Contract Design Consideration

In Fig. 4, a patient EC records two data of a registered patient. The first one is Hash ID. To avoid GDPR's concern, the Hash ID is a pre-processed ID encrypted by a one way function from patient's identity which is composed of the patient's citizen ID number and name. Hospital can use the both information to find the other two kinds of data by mapping. The second one is the relationship contract address. It records the relevant Relationship Contract address. A hospital EC records the hospital's name, its Relationship Contract address, and the relevant ID that is given by the government institution for identifying that hospital.

If the user concern other contact user's identity, for instance, patients can check a hospital's identity by calling an EC's smart contract methods, mapping, and vice versa. In order to guard against malicious users, all participants are certificated by the system administrator (CDC).

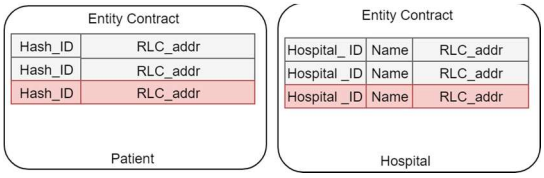

Fig 4. Entity Contract (EC)

2) *Relationship Contract (RLC):* Relationship Contract (RLC). RLC is created by EC when user registration. In Fig. 5. we design role-based contract to solve the problem of access control. Preventing sending MDC at will, there are two types of RLC, patient and hospital RLC. Hospital RLC can send an MDC only to a patient RLC; likewise, a patient can share their medical records only with authorized hospitals and send it to their RLCs. In Fig. 6. no matter what type an RLC is, it also records the owner (patient or hospital account address) of the Relationship Contract to manage the permission executing this contract. It also records medical data contract (MDC) addresses that the owner owns.

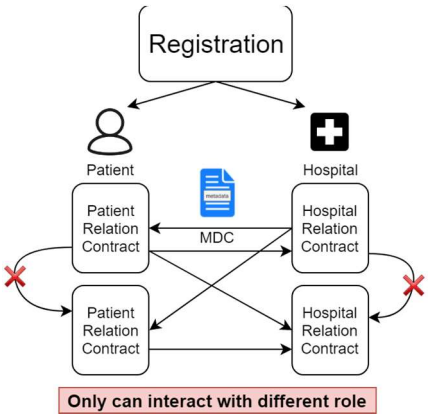

Fig 5. Role-based Relation Contract Design Consideration

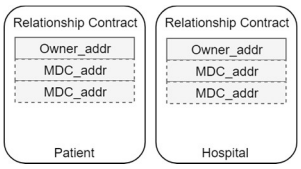

Fig 6. Relationship Contract (RLC)

3) *Data Contract (DC):* Data Contract (DC). In Fig. 7, a DC, or MDC in the healthcare domain, represents data stored in off-ledger storage. The owner recorded in MDC is the patient account address. Without containing sensitive personal data, an MDC records metadata including hospital ID, division, doctor name, time stamp, database name (the data location key access). Authorized users of an MDC can access EHRs by the data location written in the MDC.

Fig 7. Data Contract (DC)

IV. System Flow and Implementation

In this chapter, I will introduce the implementation of this system. First, I will introduce the deployment of smart contract. Then, I will introduce the System Flow of the webpage and the smart contract in the three scenarios of our system application. The second section will introduce the registration. The third section will introduce the scenario of seeing a doctor. The fourth section will introduce the scenario of sharing Electronic Health Records (EHRs). The last section will introduce how we change the system from the full-functional web3.js API (Ethereum) to the evmlite with only basic API functions.

A. Contract Deployment

The process Contract deployment, first, needs to compile our pre-written solidity smart contract using the solidity compiler (version 0.5.9). After compiling, we can get the bytecode and Application Binary Interface (ABI) of these smart contracts. Once the Tendermint commit your transaction, the message will be sent back to EVM and let EVM to execute the function of the smart contract or return the value of the parameter. When you need to call functions and parameter, you can know which public function or public parameter is in the smart contract from the ABI.

The initial smart contract, which is called administrator contract, we only allow the CDC to deploy and execute, the CDC is the largest administrator in the contact system. When preparing the administrator's smart contract, you need to input the administrator's account address, and this address will be recorded in the admin variables in the smart contract. After the administrator's smart contract is successfully deployed, the hospital EC and the patient EC will be automatically generated in the constructor. Then, the hospital EC address and the patient EC address can be called through the API. The EC address parameter, which will become an important role in our registration scenario, will be used recording patient and hospital registration content.

B. Registration Scenario

After the administrator (CDC) has deployed the first smart contract, we can obtain the patient EC address and hospital EC address recorded in the smart contract. These addresses will be set in the back end of the registration page. After the registered person fills in the information, the information will be sent to the EC by a transaction. Then, the function in the smart contract will be executed to register system members, and a corresponding RLC will be generated for that user.

1) *Patient Registration Webpage:* As shown in Fig. 8, the patients need to enter citizen ID number and name, and sets a password. After registration, CDC (default consent) will help patients hash their ID and name (for GDPR concern) and set the hash value in the transaction and execute the function in EC. It will also generate A UTC key file which can be downloaded to the user's device. The UTC contains the account address of the user and the private key encrypted with

the password set by patient. Afterwards, the patient can share the medical record by unlocking the UTC file to prove the patient's identity.

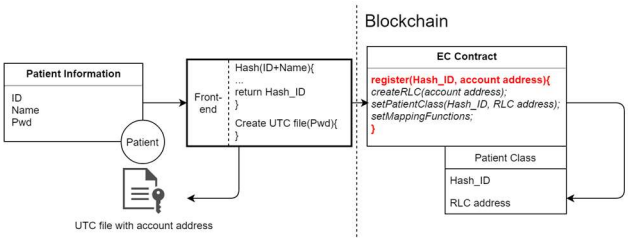

Fig 8. Patient Registration Flow

2) *Hospital Registration Webpage:* As shown in Fig. 9, the hospital should inputs the relevant ID that is given by the government institution and the name of the hospital, and set a password. After registration, a UTC file (used to verify the hospital identity) will be generated for the hospital to download to the node. Identity verification is needed when executing the function of sending MDC, and passes the information to the CDC webpage for audit. If the CDC confirms, it will help hospital to send the data to blockchain by a transaction and execute the smart contract.

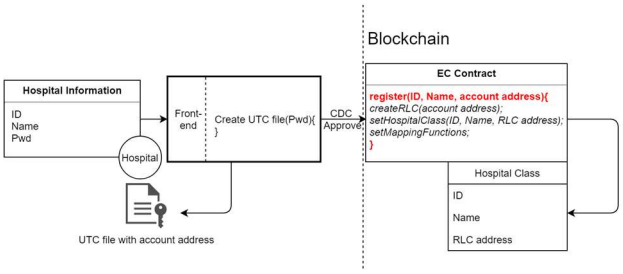

Fig 9. Hospital Registration Flow

C. *See a doctor Scenario*

As both the patient and the hospital have been registered, we can continue to see the doctor's scenario. As a patient sees a doctor, the doctor will input the patient's information and diagnosis results as an electronic medical record and store it in the hospital's local database. After the server (node of hospital) listens to the database and finds a new electronic medical record, it generates an MDC containing the metadata according to the content of the electronic medical record and sends it out as a transaction to the patient's RLC.

Doctors can add the patient's electronic medical record information through this website. As shown in Fig. 10, the doctor needs to enter the patient's ID number, the patient's name, and the patient's diagnosis. After the record is completed, the information will be made into an electronic medical record and store in the local database of the hospital. Then, the node of the hospital will listen to the database if there is a new electronic medical record. If yes, the node will prepare an MDC and send the MDC address to the patient's RLC. The hospital needs some information to generate the MDC. First, the hospital will process the patient's identity card name and name through the hash function, call the addrById mapping function to obtain the patient's account address. The account address calls usreByAddr to obtain the patient's RLC address. The hospital's account address to call hospitalByAddr to obtain the hospital's RLC. Using the information obtained above, the node can generate MDC and sends the MDC address to the patient's RLC.

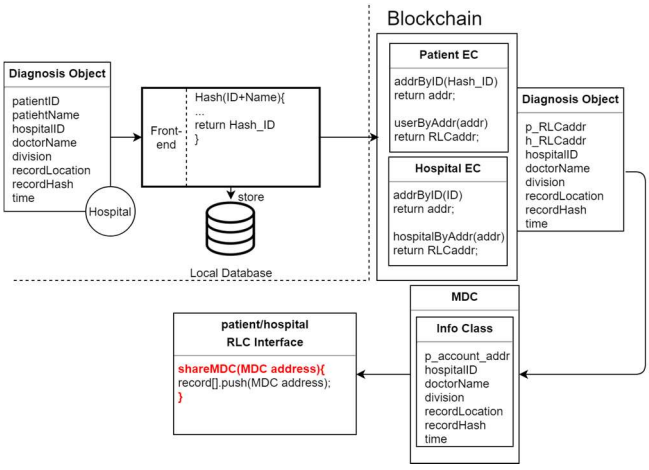

Fig 10. See a Doctor Flow

D. *Share Medical Record Scenario*

This scenario is based on a patient's device (mobile phone or smart watch). When there is MDC in patient's RLC, the patients can see the MDC metadata list and registered hospital on their devices. If the doctor sends the MDC to the patient's RLC, the patient's device will monitor whether there is a new MDC in RLC. If yes, the new MDC will be updated in the list. In this scenario, hospital A had send the MDC to the patient's RLC. Then, the patient go to hospital B, and use the webpage on device to check MDC list and the hospital to share, and send MDC to the hospital RLC. The MDC address is stored in the RLC of the hospital B. The node of the hospital B will monitor whether there is a new MDC in the RLC. If yes, the node of the hospital B will make a request to the patient's device, and the patient's device will ask the hospital A for the electronic medical record. Then, the private key will prove the patient's identity. The electronic medical record will be sent to the patient's device and passed to the hospital B local database.

The patient can connect to patient user interface webpage through the device. As the patient wants logging in, he/she has to attach the UTC file and enter the password by registration. If the password is correct, the private key can be successfully solved and enter another webpage. After logging in, shown in Fig. 11, it will call userByAddr in the patient EC to obtain the patient's RLC address, call the getMDCNum in the RLC to obtain the total number of MDCs in RLC, and then call getHistory in RLC to obtain the address of each MDC. After obtaining the address of the MDC, it calls the getInfo function in each MDC. The function gets MDC's metadata (hospital name, date of visit, clinic category, doctor's name) and will be displayed in the MDC list of the interface.

This page will listen to the newMDCevent that sent by the sharedMDC function in the RLC. Once there is a new newMDCevent, it will get the new MDC address, call getInfo to get the MDC metadata and update the MDC list in the interface. In addition, the page will call the hospital's hospitalNum in EC to obtain the total number of registered hospitals, call idByNum to get the corresponding hospital IDs in EC, call addrByID to get the account address of each hospital, call hospitalByAddr to get the name and the RLC address of each hospital. Finally, it displays the hospital list

information in the interface. As shown in Fig. 12, the main function of this page is to share the medical record to another hospital. Patient can check the MDC list of the interface, and check the hospital to share with, call the Readable mapping function in the MDC to confirm the return (false) indicating that it does not repeated sharing to the hospital. The transaction will be sent and the private key will be used to prove the patient's identity. Call addRelation in MDC and send MDC address into the RLC of hospital B.

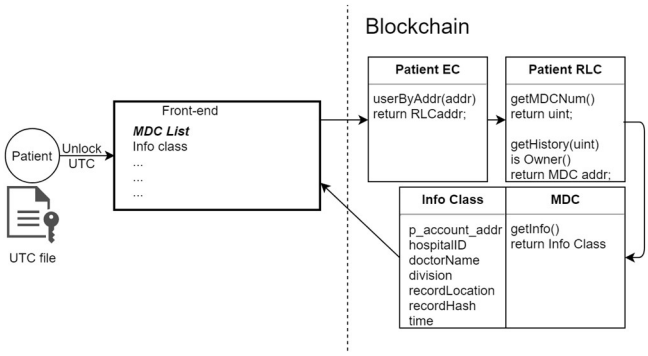

Fig 11. Share Medical Record Flow - Showing MDC List

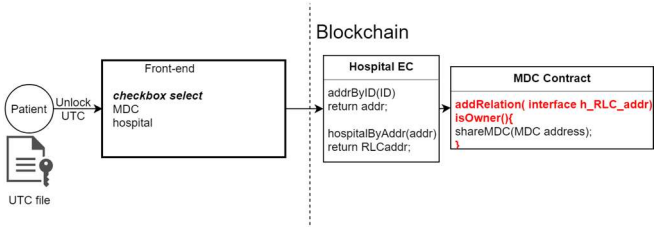

Fig 12. Share Medical Record Flow - Sharing Medical Records

E. System Transfer to Evmlite Tendermint

We choose Tendermint, which has a relatively high speed and can solve the problem of Byzantine general's fault tolerance. However, we need to use smart contract in this system to implement a more complex process and access control management on the blockchain. We choose evmlite with Tendermint consensus instead of the original Ethereum. However, evmlite did not have the consensus of the Tendermint. There are only solo (no consensus), Babble, and RAFT. All of them cannot solve the problem of Byzantine general's fault tolerance, we need to first embed the consensus Tendermint into the consensus module of evmlite. My lab classmate completes the ABCI of Tendermint and successfully connected the Tendermint consensus module to the evmlite. With evmlite with Tendermint, there is still a problem. Compared to web3.js, evmlite's API function is not complete enough. Evmlite only has basic APIs such as querying accounts to send transactions and query transaction receipt. In addition to the transaction function needs to be rewritten into the form of evmlite, if we want to replace the function programmed by web3.js to evmlite with Tendermint, we change the web3.js API to the basic evmlite API and implement quite functions.

V. Conclusion and Future Work

We have successfully implemented a data exchange system. Based on our thorough considerations and reference to experiences of many other medical blockchains, our system is more complete. We have improved the problems encountering in most medical blockchains, scalability in Ethereum. We have succeeded in making the system faster.

Nevertheless, there are still some problems. High speed blockchain will encounter the problem of relatively fast expansion of storage space. There will be similar problems in the blockchains, such as Bitcoin, which has 200g data storage, Ethereum has 2T data storage, EOS has 40T data storage. Maybe partially synchronizing will be a solution. The node from Blockchain synchronizes with less storing information or only synchronize the nearest block, or use the methods of Hyperledger. Hyperledger architecture is designed to synchronize only the blockchain group information you need.

In addition, to solve the above problems in the future, we can also implement the functions to improve access control, such as using time bound to achieve access control management. For example, I can share my electronic medical record for hospital B, but only share it for one year. One year later, hospital B will not be able to view the medical records of mine from hospital A, or the patient can remove the viewers on the device anytime. After removal, the status on the blockchain will be changed. When hospital B wants to view it, it will be blocked by permission management.

The above future work indicates different improvement on the first and second layer. In this way, the system can be more functional and the consensus property be more acceptable. With the improvement in the future, we hope that people use our consensus and EVM layer will gradually increase.

References

[1] Evm-lite, Mosaic Networks https://github.com/mosaicnetworks/evm-lite

[2] Gordon WJ, Catalini C. Blockchain technology for healthcare: Facilitating the transition to patient-driven interoperability. Comput Struct Biotechnol J 2018 Jan

[3] Dubovitskaya A., Xu Z., Ryu S., Schumacher M., Wang F. 2017. Secure and trustable electronic medical records sharing using blockchain. arXiv preprint arXiv:1709.06528.R. C. Gonzalez, R. E. Woods, *Digital Image Processing second edition*, Prentice Hall, 200

[4] Gropper A. 2016. Powering the physician-patient relationship with hie of one blockchain health it.

[5] J. Ray, "A next-generation smart contract and decentralized application platform," *Ethereum Wiki*.

[6] A. Ekblaw, A. Azaria, J. D. Halamka, and A. Lippman, "Medrec prototype for electronic health records and medical research data," *MIT Media Lab and Beth Israel Deaconess Medical Center*, 2016.

[7] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008. [Online]. Available: https:// bitcoin.org/ bitcoin.pdf

[8] Peterson K., Deeduvanu R., Kanjamala P., Boles K. 2016. A blockchain-based approach to health information exchange networks.

[9] T. Tyndall, A. Tyndall, "FHIR healthcare directories: adopting shared interfaces to achieve interoperable medical device data integration", Stud Health Technol Inform, 249 (2018), pp. 181-184

[10] H. Wang, Y. Song, "Secure cloud-based EHR system using attribute-based cryptosystem and blockchain", J. Med. Syst., 42 (8) (2018), Article 152

[11] G. Zyskind, O. Nathan, and A. Pentland, "Decentralizing privacy: Using blockchain to protect personal data," IEEE Security and Privacy Workshops, pp.180–184, 2015.

[12] MOAC, "Multi-layer blockchain architecture," 2018.

[13] C. Cachin and M. Vukolic,´ "Blockchain Consensus Protocols in the Wild," ArXiv e-prints, Jul. 2017.

[14] Tendermint Core 2018. Last accessed 15 September 2018. https://tendermint.com/docs/introduction/introduction.html#consensus-overview.