Authorized Updates to Distributed Medical Data

Chunmiao Li[1], Yang Cao[2], Zhenjiang Hu[3], Masatoshi Yoshikawa[4]

[1,3] National Institute of Informatics, Japan [2,4] Kyoto University, Japan

[1,3] SOKENDAI (The Graduate University for Advanced Studies), Japan [3] University of Tokyo, Japan

Email: [1] chunmiaoli1993@nii.ac.jp, [2] yang@i.kyoto-u.ac.jp, [3] hu@nii.ac.jp, [4] yoshikawa@i.kyoto-u.ac.jp

***Abstract*—**

***Index Terms*—medical data, authorization, update**

I. Introduction

"Through the HIPAA Privacy Rule, providers can take up to 60 days to respond (not necessarily to comply) to a request for updating or removing a record that was erroneously added" [1].

"Interoperability challenges between different provider and hospital systems pose additional barriers to effective data sharing. This lack of coordinated data management and exchange means health records are fragmented, rather than cohesive" [2]

"Patients benefit from a holistic, transparent picture of their medical history. In the age of online banking and social media, patients are increasingly willing, able and desirous of managing their data on the web and on the go [3]." [2]

"The ONC's report emphasizes that biomedical and public health researchers "require the ability to analyze information from many sources in order to identify public health risks, develop new treatments and cures, and enable precision medicine" [3].

"We build on the work of Zyskind et al. [4] to assemble references to data and encode these as hashed pointers onto a blockchain ledger. We then organize these references to explicitly create an accessible bread crumb trail for medical history, without storing raw medical data on the blockchain. " Our solution encodes the sharing peers in blockchain ledger and don't store the hash pointer to the raw data on ledger so that won't expose the privacy.

bidirectional transformation (BX hereafter)

We aims to solve problem update shared data between different peers. Since the shared data just reside in each peer's local database, read the shared data is easily promised.

But how about add, delete shared data?

We use two techniques to solve a problem where blockchain is used to prevent malicious party to operate the

II. Preliminary

A. Blockchain

Proposed with Bitcoin [5] in 2008, blockchain technology has been widely used in many fields. Blockchain provides a solution for data storage, data transfer and consensus protocol in a distributed and decentralized environment. Generally speaking, blockchain is a shared ledger and replicated by all nodes on a distributed network, which records the historical valid transactions in a chronologically chained blocks. The nodes who generate new blocks via a proof-of-work are called miners.

Not only can support the platform of cryptocurrency, blockchain can also be applied to other scenes. Ethereum [6] extend blockchain with additions such as a built-in Turing-complete programming language so that one can use this scripts (i.e., Ethereum Virtual Machine (EVM) byte codes) to write programs (i.e., smart contracts [1]) on blockchain. We can just write Solidity [2] programs and then it can be compiled to EVM code. Besides the user accounts controlled by private keys like in Bitcoin, the accounts for smart contracts are allowed in Ethereum. Anyone can build decentralized applications which consists of a collection of smart contracts. Once a transaction involving smart contract creation gets confirmed, an address is generated for the contract and later anyone can send transactions to this address for executing the logic on it. A smart contract transaction is enforced when a miner includes it in a new produced block. Other nodes will validate it and re-run contracts if it is valid.

B. Bidirectional transformation

The theory of BXs was proposed to synchronize two parts of related information (i.e., source and view). A BX consists of a pair of forward and backward transformations. A forward transformation (denoted as *get*) extracts elements from the source to build an abstract view, and the backward transformation (denoted as *put*[3]) embeds information of the view back into the source and produces an updated source. This pair of transformations should satisfy the *round-tripping* laws (i.e., *well-behaveness*) called *PutGet* and *GetPut* properties. Specifically, *GetPut* refers that no extra update should be performed back on the source when there is no change on the view, while *PutGet* hints that *put* should apply all changes on the view to the source so that the view can be regenerated from the updated source by *get*. The most distinguished point of BX is that view can contain only a few part elements of source. Actually, BX stems from the the view update problem in database [7], which studies how to translate the updates on a view table to the updates on the relating base (source) table. By designate some consistency between source and view, BX

[1]Hyperledger and others still provide platforms to write smart contracts.

[2]https://solidity.readthedocs.io/en/v0.5.2/

[3]*put* is not a simple inverse of *get*. Instead, it accepts the view and the original source as input and produces an updated source as output.

programs can synchronize the source and view and promise that they satisfy the consistency.

One recent big progress towards practical use of BX is the observation that the essence of bidirectional programming is nothing but writing a well-behaved backward transformation (*put*) [8]–[10], because the corresponding forward transformation can be derived uniquely and freely. A core language, BiGUL [11], has been developed for putback-based bidirectional programming.

III. System architecture

In this section, we will illustrate our system architecture. Part A will describe the data distribution between sharing peers. Part B will present our system design and later explain it using an update scenario.

A. Data distribution

Shared data can exist in different peers, as shown in Fig. 1. Suppose there are three entities, doctor, researcher and patient. Doctor and Researcher share some data which are stored in D12 on Doctor side and D21 on researcher side respectively. D12 and D21 should contain the same data, which means if doctor or researcher update this shared data on their own side, then this modification should be propagated to the shared part on the other side. Accordingly, D13 and D31 have the same content and so do in D23 and D32.

Each node will store its all data on a bigger database, such as D1, D2 and D3. Each shared data can be seen as a view which is produced by this bigger database named as source. For example, D12 can be made from D1 by using *get*. If D12 is modified, then the D1 need to be reproduced from original D1 and D12 by using *put*. We just write BX in researcher and patient side to express the pair of *get* and *put* functions.

Note in our data distribution, shared data between any two parties will not be exposed to the third party, which can keep privacy between two parties in some degree. For example, any update on D12 and D21 can only be known by patient and researcher. However, after the change on D21 are reflected to D2, since D2 has been modified, D23 might need to be updated to a new version by using *put* on D2.

Fig. 2 presents an example to show data distribution between patient Alice, doctor Bob and researcher Charlie. Each party have their own local base table, named as D1, D2, D3 respectively. For each table, there are some attributes such as medication name and address on D1. Table D13 (also D31) are shared by Alice and Bob and can be produced from either D1 or D3 by *get*. Table D23 (also D32) are shared by Charlie and Bob and can be generated from D2 or D3 by *get*. Shared tables all stay in sharing peer's local databases. Note here the metadata collection table resides in the smart contract on blockchain. Each entry of metadata collection table corresponds to a shared table. For example, the entry for D13 or D31 declare that it is shared by Alice and Bob and Bob can update all attributes value but Alice can only change the clinical data. The "Latest Update Time" shows when the metadata was changed most recently. Moreover, "Authority to Change Permission allow Bob to change other peers (here just Alice)'s authority so that Alice can update dosage by change the value for "Dosage" attribute to "Bob, Alice".

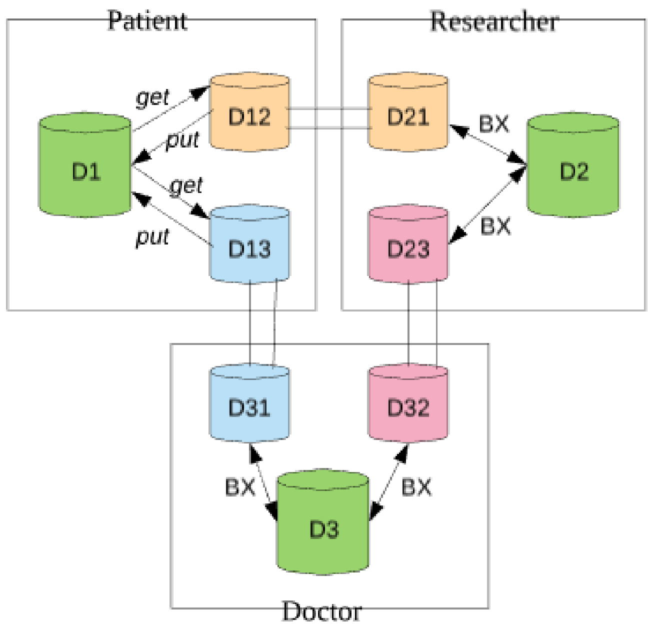
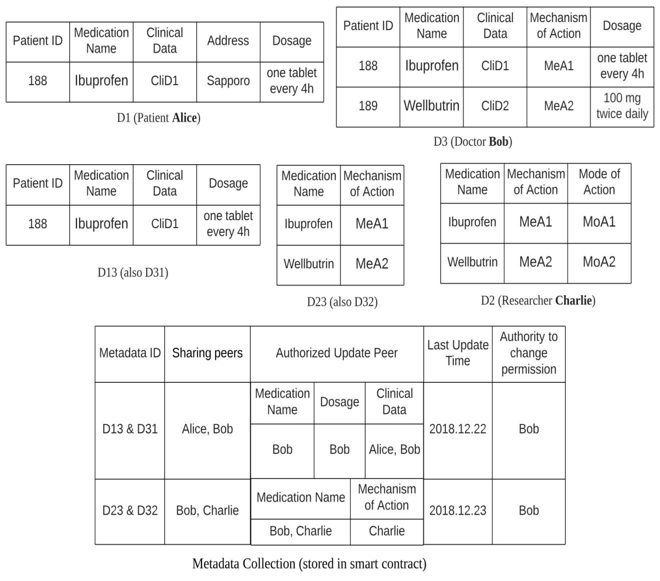


Fig. 1. DB architecture



Patient ID	Medication Name	Clinical Data	Address	Dosage
188	Ibuprofen	CliD1	Sapporo	one tablet every 4h

D1 (Patient **Alice**)

Patient ID	Medication Name	Clinical Data	Mechanism of Action	Dosage
188	Ibuprofen	CliD1	MeA1	one tablet every 4h
189	Wellbutrin	CliD2	MeA2	100 mg twice daily

D3 (Doctor **Bob**)

Patient ID	Medication Name	Clinical Data	Dosage
188	Ibuprofen	CliD1	one tablet every 4h

D13 (also D31)

Medication Name	Mechanism of Action
Ibuprofen	MeA1
Wellbutrin	MeA2

D23 (also D32)

Medication Name	Mechanism of Action	Mode of Action
Ibuprofen	MeA1	MoA1
Wellbutrin	MeA2	MoA2

D2 (Researcher **Charlie**)

Metadata ID	Sharing peers	Authorized Update Peer			Last Update Time	Authority to change permission
D13 & D31	Alice, Bob	Medication Name	Dosage	Clinical Data	2018.12.22	Bob
		Bob	Bob	Alice, Bob		
D23 & D32	Bob, Charlie	Medication Name	Mechanism of Action		2018.12.23	Bob
		Bob, Charlie	Charlie			

Metadata Collection (stored in smart contract)

Fig. 2. Data distribution

B. Permission on update data

The metadata are created immediately once some peers wants to share data with each other. Suppose doctor Bob initiates the data sharing with Alice. He will deploy a smart contract on blockchain which stipulates the metadata about the shared data, such as sharing peers (i.e., Alice and Bob) and so on. Table Metadata Collection on Fig. 2 are built by Bob.

Since smart contract can not be altered after it was deployed on blockchain. There are two ways to update the permission for update to shared data:

1) deploy a new contract and notify all nodes on blockchain that this new one should be used to verify authority later;
2) update the state of variables in contract.

We choose the latter way. As shown in Fig. 2, the authority to modify the update permission are encoded in the field named as "authority to change permission". For example, for D13 and D31, Bob can change the permission to update "Dosage" to "Bob, Alice" so that Patient Alice can also update the "Dosage".

C. System design

Our system consists of following components, which can be seen from Fig. 3.

- Front-end user interface: control the interaction between users and other components.
- Database: each user has an overall database and many shared databases with other users. The latter can always be reproduced from the former.
- Blockchain: keep a record of access and update to the shared data. Also, fonrt-end user interface communicate with blockchain network via a blockchain node. The smart contract on blockchain contains the metadata of shared medical data and maintain an update log to store historical modification for each metadata.
- Database manager (server) dispose the synchronization between shared data and local data in terms of consistency logic relations.

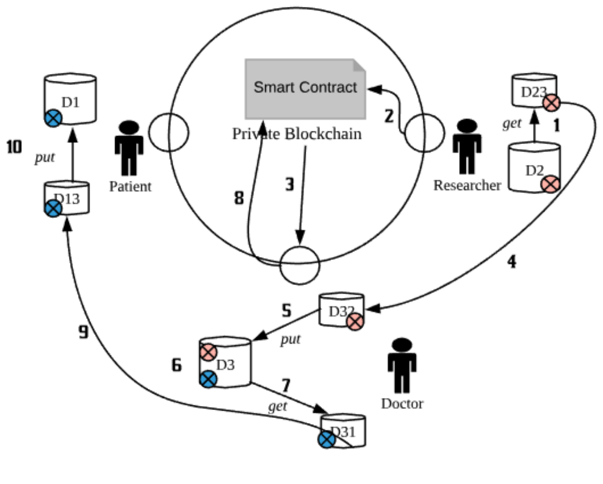


Fig. 3. An work flow for updating shared data

D. Case analysis

Fig. 3 depicts an scenario where researcher initiates the update the shared data. The numbers indicates the corresponding operations sequence.

After the update on D2, the researcher wants to propagate the update to the shared data D23 so that he uses the *put* to regenerate D23 in(step 1). Then he will call a smart contract via a node by sending the repquest for update to the shared data (i.e., D32) (step 2). Note that a smart contract records all permission info about that shared data. Only the researcher's update satisfy the permission can his update be propagated to others. The smart contract will be executed until all nodes form consensus on this update request, which means researcher is authorized to update the shared data with doctor. Each node will conduct the smart contract locally. The metadata of shared data will be updated. Meanwhile, the node relating to Doctor will receive the notification from contract that the shared data with researcher need to modified (step 3). The doctor will request data from researcher by sending a message based on some encrypted communication protocol and get the newest shared data to refresh D32 (step 4). After that, doctor will use *put* to reflect the change on D32 to the total database D3 (step 5). Since doctor shared some data (D31) with patient, he need to check whether D31 need to be reproduced (step 6). If yes, he will use *get* to regenerate D31 (step 7) and request smart contract for permission to update D13 (step 8). Once this permission is allowed, patient will receive the notification about the change on shared data and ask doctor to send the updated D31. After patient get the modified D31 (step 9), he will use this to update D1 via *put* (step 10).

IV. Discussion

A. Pros of our solution

- blockchain's distribute consensus protocol scheme keep the shared data between all sharing peers are consistent.
- Keep sensitive data private
- All modification on shared data can be recorded on blockchain (tamper-resist; permanent)

B. Data source

C. Threat to validity

1) Privacy: threat1 + possible solution privacy: blockchain limitations

2) Throughput: threat2 + possible solution

3) Robustness:

V. Related work

The idea of introducing Blockchain technology to healthcare data system was presented firstly in [12] where they use blockchain for data storage to guarantee medical data can not be modified by anyone. Also, they designed a Healthcare Data Gateway (HDG) to control access of the shared data. However, medical data size can become huge so that become a burden for blockchain nodes' storage since each node have the same copy of blockchain. MedRec [13] choose to store raw medical data on providers' database and patients can download the data from it after authorized by smart contract on blockchain. They aimed to enable patient to engage in their healthcare.

But not all provider data such as physician intellectual property can be exposed to patients [14], [15]. Instead, we allow each node share a piece of medical data not total but still keep consistency between them after the updates to the shared ones. Additionally, any modifications on data shared by two nodes will not be disclosed to the third party which keep the consistency only exists in sharing peers. Moreover, since all shared data with others can be a part of each nodes' local total databases, we can decide whether one shared data have some influence on the other shared pieces and then propagate this change to the third party.

Our solution differs with previous ones. In our system, all parties, such as doctors, patients, and researchers can benefit from sharing data with others.

VI. Conclusion

In the future, we will use the real patient data to do experiment but use some de-identification technology to protect patient data from being exposed in terms of

Acknowledgment

The authors would like to thank all PRL lab members for the helpful advice in the draft version.

References

[1] H. Centers for Medicare & Medicaid Services *et al.*, "Hipaa administrative simplification: standard unique health identifier for health care providers. final rule." *Federal register*, vol. 69, no. 15, p. 3433, 2004.

[2] K. D. Mandl, D. Markwell, R. MacDonald, P. Szolovits, and I. S. Kohane, "Public standards and patients' control: how to keep electronic medical records accessible but privatemedical information: access and privacydoctrines for developing electronic medical recordsdesirable characteristics of electronic medical recordschallenges and limitations for electronic medical recordsconclusionscommentary: Open approaches to electronic patient recordscommentary: A patient's viewpoint," *Bmj*, vol. 322, no. 7281, pp. 283–287, 2001.

[3] O. of the National Coordinator for Health Information Technology, "Report to congress: Report on health information blocking," 2015.

[4] G. Zyskind, O. Nathan *et al.*, "Decentralizing privacy: Using blockchain to protect personal data," in *Security and Privacy Workshops (SPW), 2015 IEEE*. IEEE, 2015, pp. 180–184.

[5] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[6] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger."

[7] F. Bancilhon and N. Spyratos, "Update semantics of relational views," *ACM Transactions on Database Systems (TODS)*, vol. 6, no. 4, pp. 557–575, 1981.

[8] Z. Hu, H. Pacheco, and S. Fischer, "Validity checking of putback transformations in bidirectional programming." in *FM*, 2014, pp. 1–15.

[9] H. Pacheco, Z. Hu, and S. Fischer, "Monadic combinators for putback style bidirectional programming," in *Proceedings of the acm sigplan 2014 workshop on partial evaluation and program manipulation*. ACM, 2014, pp. 39–50.

[10] H. Pacheco, T. Zan, and Z. Hu, "Biflux: A bidirectional functional update language for xml," in *Proceedings of the 16th International Symposium on Principles and Practice of Declarative Programming*. ACM, 2014, pp. 147–158.

[11] H.-S. Ko, T. Zan, and Z. Hu, "BiGUL: A formally verified core language for putback-based bidirectional programming," in *Proceedings of the 2016 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation*, ser. PEPM '16. New York, NY, USA: ACM, 2016, pp. 61–72. [Online]. Available: http://doi.acm.org/10.1145/2847538.2847544

[12] X. Yue, H. Wang, D. Jin, M. Li, and W. Jiang, "Healthcare data gateways: found healthcare intelligence on blockchain with novel privacy risk control," *Journal of medical systems*, vol. 40, no. 10, p. 218, 2016.

[13] A. Azaria, A. Ekblaw, T. Vieira, and A. Lippman, "Medrec: Using blockchain for medical data access and permission management," in *Open and Big Data (OBD), International Conference on*. IEEE, 2016, pp. 25–30.

[14] U. D. of Health, H. Services *et al.*, "Individuals' right under hipaa to access their health information 45 cfr 164.524," 2017.

[15] C. Grossman, W. A. Goolsby, L. Olsen, and J. M. McGinnis, "Clinical data as the basic staple of health learning: creating and protecting a public good," *Washington, DC: Institute of Medicine*, 2011.