Electronics and Computer Science
Faculty of Physical Sciences and Engineering
University of Southampton

Augustas Mirinas
April 17, 2024

Trustless System for Personal Data Sharing

Project supervisor: Dr George Konstantinidis
g.konstantinidis@soton.ac.uk

Second examiner: Dr Mark Vousden
m.vousden@soton.ac.uk

A project report submitted for the award of
BSc Computer Science

Abstract

Contents

1 Design

1.1 Data privacy

In this project, data privacy is achieved using an algorithm to compute "consent-abiding query answers" from the publication of the supervisor of this project [1]. In this publication, to ensure that parts of private data, not consented to share, will stay private, data queries are rewritten in compliance with data subject constraints. These constraints consist of rules specifying combinations of data fields that are prohibited from sharing. Each data subject has their own set of constraints, which is used as neccessary when rewriting data queries. After rewriting, resulting queries would only return data that is consented for sharing. To adapt this method for a trustless system - in which all network participants are able to reach an agreement about the past and current state of shared private information - constraints are used to filter shared data instead of rewriting data queries, as done in [1]. This makes the logic of consent-abiding data sharing independent from any private information publisher, as the data during publishing is filtered by the system, not by the publisher.

In order to publish private information, it has to be linked to existing user (data subject) constraints, which are identified by a unique username. If a user of the information being published does not exist, the information will not be saved to the system. To create a user identity, data subject must come up with a username that does not exist in the system yet and submit it through an independent party. This independent party could be a government body or other data protection authorities, but it is not in the scope of this project. At the same time user identity is created, user consent constraints are defined using an index of the information as a reference, as the data is published in the form of a table.

Both constraints and the index of data conforms to the table format. A single constraint is expressed as a combination of columns from the index - an example can be seen in Figure 1, which is taken from [1]. The data being published contains records, or rows, with a username column to specify which user this record belongs to. The index, listing all possible columns of the record, is also uploaded by the user at the time of creating a constraint. Only then can the data about this user, collected by some organization, be published to the system allowing authorised members of the network to access

it, without compromising data subject's consent.

User Consent	Negative Constraint
Patient 1312 does not want to share her phone number (5^{th} attribute of the `Patients` relation)	$N_1(x_5) \leftarrow \texttt{Patients}(1312, x_2, x_3, x_4, x_5, x_6, x_7)$
Patient 1312 does not want to share her any combination of her attributes (N_2). N_2 actually subsumes N_3 and N_4 which dissalow Name and Disease respectively	$N_2() \leftarrow \texttt{Patients}(1312, x_2, x_3, x_4, x_5, x_6, x_7)$ $N_3(x_2) \leftarrow \texttt{Patients}(1312, x_2, x_3, x_4, x_5, x_6, x_7)$ $N_4(x_7) \leftarrow \texttt{Patients}(1312, x_2, x_3, x_4, x_5, x_6, x_7)$
Patients 4872 and 2321 do not want to share the association of their Birthdate together with their Disease	$N_5(x_4, x_7) \leftarrow \texttt{Patients}(4872, x_2, x_3, x_4, x_5, x_6, x_7)$ $N_6(x_4, x_7) \leftarrow \texttt{Patients}(2321, x_2, x_3, x_4, x_5, x_6, x_7)$
Do not share patient IDs when their disease is being cross checked against the `Insurance` table	$N_7(x_1) \leftarrow \texttt{Patients}(x_1, x_2, x_3, x_4, x_5, x_6, x_7) \wedge \texttt{Insurance}(x_7, x_8)$

Figure 1: A capture of the table showing some examples of user constraint semantics

1.2 Blockchain

1.2.1 Motivation

The design chosen for this project relies on blockchain technology - utilising decentralisation and immutability to create a system, where shared information cannot be manipulated and is maintained by every permissioned member of the network. Traditionally, sensitive user information for sharing is pooled into a single place, then accessed by different parties as needed. Add a reference to a centralised HIE implementation. This presents a challenge to match different pieces of data from different sources to form correct user profiles. User identification, data formats and privacy concerns are considered when applying advanced algorithms to solve this problem. A decentralised blockchain network addresses these problems more efficiently and with better reliability - network members have access and can participate only when they are following policies set and agreed by the rest of the network.

To enforce the network policies on the participants, the system produces a record of every network member interaction. These records, containing information about the interaction and ordered based on the interaction timestamp, form a public ledger. Member interactions with the system are also called transactions. Only transactions, approved by required members are valid and recorded into a ledger. The ledger is replicated across multiple network members to prevent any unauthorised changes to the ledger - if a malicious member would attempt to change the record of system transactions, other network members would still be able to agree on the correct

ledger version and deny the unauthorised change, based on their ledger replicas. This results in a system that consists of multiple independent members following the same rules and policies. This feature of the blockchain network is used for implementation of the data privacy algorithm presented in subsection 1.1.

One major advantage of a blockchain network over a traditional centralised information exchange approach is data storage. The proposed blockchain network does not store data on the ledger, nor does it communicate private information through transactions - meaning actual data intended for sharing is never recorded on the public ledger. Instead, when publishing new data to the system only the hash of data is made public. Actual private data is stored only by the data publisher and any members that have access to it. This model records evidence of published data and makes it public to everyone without revealing the contents of published data. By inspecting hashes of data in the ledger, it is possible to confirm private data was not tampered with after publishing. This and more scenarios will be analysed in section 3. Another benefit of fragmented data storage is damage mitigation in case of a data breach or a malicious access. A member of the network will only store information that is either used or published by themselves - in most cases this will be a small subset of overall private data published to the system, which means a single instance of a data breach would compromise less private information than in the traditional centralised data exchange, where all private information is pooled to a single place.

1.2.2 Network design

The blockchain network implementing this design is permissioned, meaning every member of the network is known and has defined access to resources and permissions to commit certain transactions. This is different from public blockchain systems, such as Bitcoin[2] or Ethereum[3], where unknown members can be part of the network and their transactions are validated via "proof of stake" or other consensus mechanisms. The task of transaction validation of a permissioned blockchain network is designated to a trusted provider of this service - a member of the network with the role of deciding which transactions are valid and should be recorded into the ledger. The same network node is also responsible for issuing and identifying members of the system. This method of transaction validation and member identi-

fication is more scalable and provides additional features to the network, compared with "proof of stake" and similar mechanisms (provide reference for this statement?)

The second component to be implemented after the network validator is a channel. This network component defines which network members are able to participate in the channel, how transactions of the channel should be approved and who can make and approve changes to the channel configurations. These configurations may vary depending on the use case of this system - some examples will be given in section 3. For each channel, a set of functions are defined in smart contracts of the blockchain network. These functions implement private data filtering using user constraints and enable data publishing, reading and constraint registering. Furthermore, smart contracts define collections of data and which members can read or write to those collections. These network components allow highly customizable segmentation of information access. Different areas of information could be separated into channels - each one would have an individual ledger for better scalability and collections would be used to control access for different members of the network.

2 Implementation

To implement this blockchain system for secure and consent-abiding private data sharing Hyperledger Fabric will be used - an "open source enterprise-grade permissioned distributed ledger technology platform"[4]. This technology was established under the Linux Foundation, which has a reputable history of open source projects. Hyperledger Fabric is a permissioned blockchain system, meaning every member of the network knows each other. This is well suited for a proposed design, as all members - either data subjects, data publishers or data users - should be approved anyway. Every member of the system has the same goal - provide useful information or use it for a good cause. Any action is recorded on a ledger, deterring any malicious members to take action and face consequences.

2.1 Environment and tools

The deployment of this implementation is not production grade - to ease development process, explanation of the system and to allow easy replica-

tion for anyone interested, all network components are deployed on a single machine. Docker Compose is used for this purpose - by defining a `docker-compose.yaml` file, multiple containers are brought up using officialy supported Hyperledger Fabric commands. These commands use environment variables and generated network certification files to correctly set up nodes of the network. In production environment, these files and variables should be carefully considered for each component of the network, but in this project the Docker Compose configuration and other required files are going to be generated by Fablo - "a simple tool to generate the Hyperledger Fabric blockchain network"[5]. Fablo takes one configuration file as an input and generates the network with specified organizations and other neccessary components. Then `fablo` commands can be used to manage the generated network. This tool is part of Hyperledger Labs - a space to start and contribute to projects related to Hyperledger Fabric. Another tool from the same origin is Blockchain Explorer[6], useful for observing transactions made on the network using graphical user interface. Java with Maven is used for developing smart contracts and Git for version control.

2.2 Network configuration

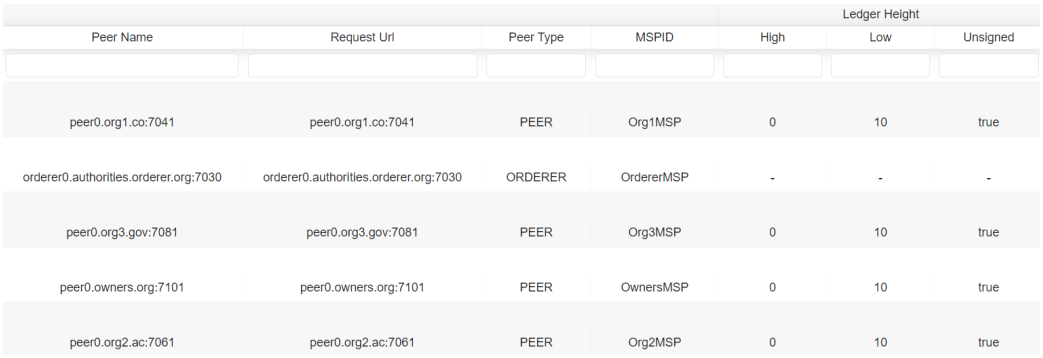
In Hyperledger Fabric, the node validating blockchain network transactions is called an orderer. Here, orderer is hosted by a regulating authority - a trusted third party. Other network components are 3 organizations - separate business entities, publishing and/or using private data collected from users (data subjects). Fourth organization is responsible for consensus of data subjects. Through this organization, users can submit their consent constraints, which will be used to only publish information consented for sharing. Each organization is only a logical unit; to interact with the network, each organization will have one physical network node, called peer in Hyperledger Fabric. As stated in subsection 2.1, network configuration files and required environment variables will be generated with the help of Fablo, therefore in this section Fablo configuration file will be demonstrated, which is more concise and easier to explain compared to many files, required for the Fabric network. Here, only a part of the configuration file will be shown with comments to explain each setting.

```yaml
---
# global network settings
global:
    fabricVersion: "2.5.0"
    # enable TLS for secure communication between network nodes
    tls: true
    tools:
        # start a Node.js server hosting Blockchain Explorer
        explorer: true
orgs:
    # orderer organization, hosting a node which validates transactions
    - organization:
        name: "Orderer"
        domain: "orderer.org"
        orderers:
          - prefix: "orderer"
            groupName: "auth"
            type: "raft"
            # can be more than one orderer node for redundancy
            instances: 1
    # data publisher, user or consenter organization
    - organization:
        name: "Org1"
        mspName: "Org1MSP"
        domain: "org1.co"
        ca:
            prefix: "ca"
        peer:
            prefix: "peer"
            # multiple nodes could be used to have multiple data points in the organization
            instances: 1
            db: "LevelDb"
        tools:
            # enable REST interface to call chaincode and create identities
            fabloRest: true

    # ...
    # three more organizations are defined but omitted with identical settings
    # org2.ac - publisher/user
    # org3.gov - publisher/user
    # owners.org - data subjects (consenters)
```

```
    # ...

channels:
    # one channel consisting of all organizations defined
    - name: "ch1"
        orgs:
          - name: "Org1"
            peers:
              - "peer0"
          - name: "Org2"
            peers:
              - "peer0"
          - name: "Org3"
            peers:
              - "peer0"
          - name: "Owners"
            peers:
              - "peer0"
chaincodes:
    # ...
```

Listing 1: fablo.yaml - configuration file generating the network

After generating the network using given configuration in listing 1, all nodes are created and started. As mentioned, each organization has its own peer, which can interact with the network. Organization `mspName` can be used to control resource access in smart contracts - this is used when granting access to create consent constraints for `owners.org` organization. Another option for organization is database implementation. Every node has an instance of database, where network data, such as consent constraints or published user information, is stored and queried from. Any changes in network data is propagated in databases of each node. Option `db` specifies whether to use LevelDb or CouchDb for peer database implementation. LevelDb is used for this project, utilising its flexibility and speed. Data is stored as ordered key-value pairs, and maps string keys to byte array values, meaning any information can be stored, as long as it can be deterministically serialized into bytes. CouchDb is a NoSQL database storing data as JSON objects, which runs as a separate database process and enables rich queries, however it is left for future considerations. Lastly, a single channel is defined consisting of peers of every organization defined. In figure 2, all spawned Docker containers and Hyperledger Fabric commands used to start them are shown.



				Ledger Height		
Peer Name	Request Url	Peer Type	MSPID	High	Low	Unsigned
peer0.org1.co:7041	peer0.org1.co:7041	PEER	Org1MSP	0	10	true
orderer0.authorities.orderer.org:7030	orderer0.authorities.orderer.org:7030	ORDERER	OrdererMSP	-	-	-
peer0.org3.gov:7081	peer0.org3.gov:7081	PEER	Org3MSP	0	10	true
peer0.owners.org:7101	peer0.owners.org:7101	PEER	OwnersMSP	0	10	true
peer0.org2.ac:7061	peer0.org2.ac:7061	PEER	Org2MSP	0	10	true

Figure 2: A Blockchain Explorer table listing all network components

2.3 Chaincode endorsement

In figure 2, ledger height can also be seen. The largest number corresponds to 10 blocks appended to the ledger. These blocks contain endorsement transactions, which are produced with `peer lifecycle chaincode` commands. These commands install chaincode to the Fabric network. Chaincode defines smart contracts, which have a set of functions, that describe what actions can be executed in the network. These transactions are used to setup the network, and more transactions can be submitted by updating or installing new chaincode or invoking the chaincode.

Every chaincode has an endorsement policy, indicating which organizations have to approve transactions produced by chaincode invokes. To facilitate endorsement each required organization executes invoked function on their own peer node and sends the output of the function to the ordering service. If all required peers agree on the same output, the orderer writes function output to the ledger and all channel peers update their instance of the ledger to the newest version. Endorsement policies are agreed at the time of chaincode installation and they are applied for every chaincode invoke. In Fablo, network chaincode with endorsement policies is also defined in `fablo.yaml` - the configuration file from listing 1. In listing 2, a continuation of the file, containing chaincode configuration, is shown.

```yaml
---
global:
    # ...
orgs:
    # ...
channels:
    # ...
chaincodes:
  - name: "private-data"
    version: "1.0"
    # language that the chaincode is written in
    lang: "java"
    # channel that will have this chaincode installed
    channel: "ch1"
    # physical directory of the chaincode files
    directory: "private-data-chaincode"
    endorsement: "AND('OwnersMSP.member', OutOf(2, 'Org1MSP.
       member', 'Org2MSP.member', 'Org3MSP.member'))"
    # private data collections shared with different
       organizations to controll private information access
    privateData:
      - name: "CollectionA"
        orgNames:
            - "Org1"
            - "Org3"
      - name: "CollectionB"
        orgNames:
            - "Org1"
            - "Org2"
            - "Org3"
      - name: "CollectionC"
        orgNames:
            - "Org2"
            - "Org3"
```

Listing 2: fablo.yaml - configuration file defining network chaincode

In this implementation, option `endorsement` defines a policy - transactions have to be endorsed by `owners.org` and the majority of data publishing/using organizations. This is done to give approval power to the organization responsible for data subject constraints. The chaincode in Hyperledger Fabric is installed to one channel only, meaning other channels in the system will have their own version of chaincode, with different collections and different

endorsement policies. Endorsement policies make the network transparent - a chaincode invoke is impossible without the knowledge of other endorsing members and the result of the invoke is agreed on by all endorsing organizations. Separate channels, however, give the ability to select a specific set of functions, collections and an endorsement policy based on the sensitivity and usage of shared data, as well as relations between channel organizations.

2.4 Private collections

In listing 2 option `privateData` defines collections and organizations with access to them. In this network configuration, each of the three data using/publishing organizations have access to two or three private collections. `owners.org` does not get any access, because it does not publish, nor read data from the system - only submit consent constraints. Private data published on the system will be written into one of three collections implemented as replicated data stores, separate from the public channel ledger, which are maintained only by organizations with access to that collection - one organization maintains three ledgers: one public ledger and two private collections, except `org3.gov`, which has access to all colections. Because access to collections is limited, not all organizations can endorse transactions. For this reason, endorsement of transactions involving reading or writing to a private collection is done only by organizations with sufficient access - meaning `owners.org` only endorses functions related to constraints. Figure 3 contains all network nodes with a channel ledger and accessible private collections. Arrows indicate communication - thicker arrows are public transactions of the channel, issued by the orderer for every change of the ledger. Remaining arrows represent gossip between peers with access to private collections. These updates are not public and does not use orderer as a verifier, since collection updates do not produce transactions. This way data remains uncompromised, but still agreed on by multiple nodes.

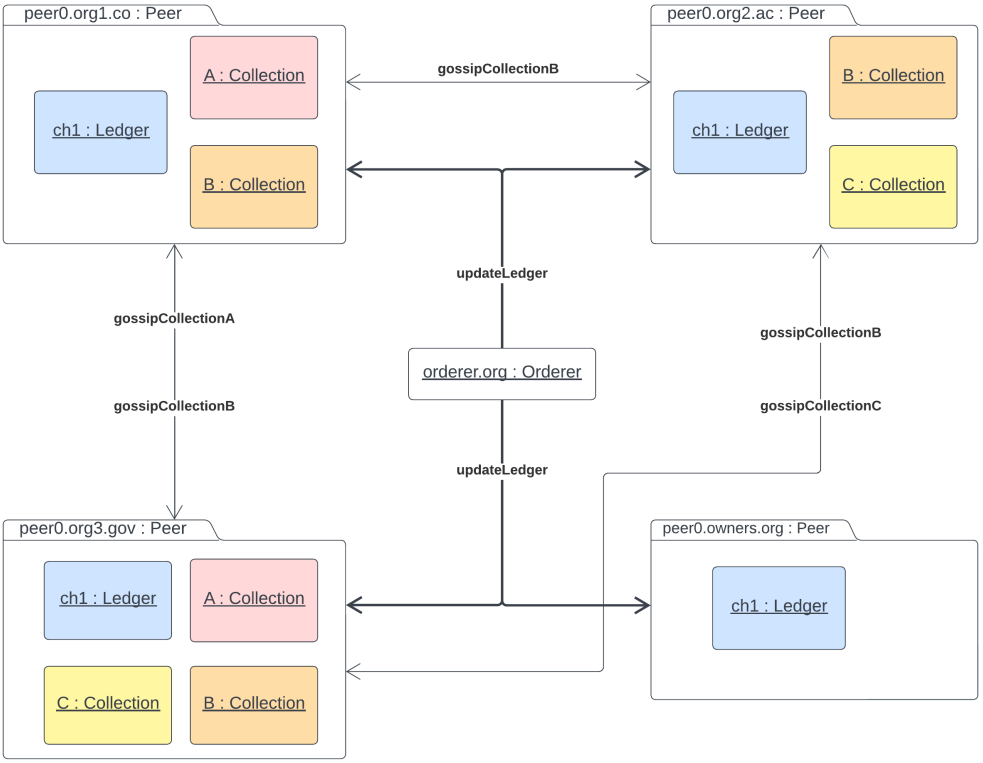


Figure 3: A UML object chart showing a channel ledger and private collections

3 Example scenarios

3.1 Private data tampering

References

[1] G. Konstantinidis, J. Holt, and A. Chapman, "Enabling personal consent in databases," *Proceedings of the VLDB Endowment*, vol. 15, no. 2, p. 375 – 387, 2021.

[2] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Cryptography Mailing list at https://metzdowd.com*, 03 2009.

[3] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[4] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolić, S. W. Cocco, and J. Yellick, "Hyperledger fabric: A distributed operating system for permissioned blockchains," in *Proceedings of the Thirteenth EuroSys Conference*, EuroSys '18, (New York, NY, USA), Association for Computing Machinery, 2018.

[5] J. Dzikowski, "Fablo." `https://github.com/hyperledger-labs/fablo`, 2024.

[6] N. Frunza, "Blockchain explorer." `https://github.com/hyperledger-labs/blockchain-explorer`, 2024.