Right to be Forgotten in Permissioned Blockchains

Project-I (CS47005) report submitted to

Indian Institute of Technology Kharagpur

in partial fulfilment for the award of the degree of

Bachelor of Technology

in

Computer Science and Engineering

by

Anand Manojkumar Parikh

(20CS10007)

Under the supervision of

Professor Shamik Sural

Department of Computer Science and Engineering

Indian Institute of Technology Kharagpur

Autumn Semester, 2023-24

November 3, 2023

DECLARATION

I certify that

(a) The work contained in this report has been done by me under the guidance of my supervisor.

(b) The work has not been submitted to any other Institute for any degree or diploma.

(c) I have conformed to the norms and guidelines given in the Ethical Code of Conduct of the Institute.

(d) Whenever I have used materials (data, theoretical analysis, figures, and text) from other sources, I have given due credit to them by citing them in the text of the thesis and giving their details in the references. Further, I have taken permission from the copyright owners of the sources, whenever necessary.

Date: November 3, 2023 (Anand Manojkumar Parikh)
Place: Kharagpur (20CS10007)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

KHARAGPUR - 721302, INDIA

CERTIFICATE

This is to certify that the project report entitled "**Right to be Forgotten in Permissioned Blockchains**" submitted by **Anand Manojkumar Parikh** (Roll No. 20CS10007) to Indian Institute of Technology Kharagpur towards partial fulfilment of requirements for the award of degree of Bachelor of Technology in Computer Science and Engineering is a record of bona fide work carried out by him under my supervision and guidance during Autumn Semester, 2023-24.

Shamik Sural

Professor Shamik Sural
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur
Kharagpur - 721302, India

Date: November 3, 2023

Place: Kharagpur

Abstract

Name of the student: **Anand Manojkumar Parikh** Roll No: **20CS10007**

Degree for which submitted: **Bachelor of Technology**

Department: **Department of Computer Science and Engineering**

Thesis title: **Right to be Forgotten in Permissioned Blockchains**

Thesis supervisor: **Professor Shamik Sural**

Month and year of thesis submission: **November 3, 2023**

Blockchains, famous for their immutability and reliability, have successfully powered cryptocurrencies like Bitcoin and Ethereum. However, extending their usage to enterprise applications in a permissioned setting has raised many privacy and security issues, for e.g. the Right to be Forgotten in the European Union's General Data Protection Regulation, which requires that every user (subject) has the authority to delete all data pertaining to them, and hence be forgotten from the network. Such a redactability requirement clashes with the immutable nature of blockchains, as information once committed can never be changed, let alone erased. This work aims to provide an architecture which uses blockchains to enforce the RTBF for all data users, hence maintaining the qualities of immutability, reliability, security and privacy in a trustless decentralized environment.

Acknowledgements

I sincerely thank my project guide, Professor Shamik Sural, for his invaluable insights and constructive feedback which made this project a successful endeavour. I thank Mr. Tirthankar Sengupta, Research Scholar, for his continuous guidance throughout the project. Furthermore, I thank the Department of Computer Science and Engineering for granting me an opportunity and resources for pursuing this project.
I finally extend my gratitude to my family and friends for their support.

Contents

Abbreviations

HLF	**H**yper**L**edger **F**abric
ABAC	**A**ttribute **B**ased **A**ccess **C**ontrol
ACL	**A**ccess **C**ontrol **L**ist
GDPR	**G**eneral **D**ata **P**rotection **R**egulation
RTBF	**R**ight **T**o **B**e **F**orgotten

Chapter 1

Introduction

1.1 Permissioned Blockchains

Permissioned Blockchains are fundamentally different from their more widely-used permissionless counterparts. Permissionless blockchains, like (Bitcoin(5), Ethereum(4) and other cryptocurrencies) are named so, because they do not require any pre-authorization to join. They are distributed across the globe with millions of peers connected at any given point of time without any of way giving a precise count/identity of connected peers. Since we do not know all peers, we cannot use classical distributed algorithms for achieving consensus and must use Proof-Of-Work (and other probabilistic algorithms which prefer liveness over safety). This leads to poor throughput of the network for ordering and committing transactions. To successfully use blockchains in an enterprise setting for agreeing to and executing customized business logic, such permissionless blockchains are not suitable. So, we move towards permissioned blockchains where each identity (admin, peers, clients, etc.) is pre-registered and known to all when connecting to the network. Now, we can use message-passing based classical distributed algorithms (14)(15)(16) for achieving consensus and building sophisticated systems on top of this architecture.

1.2 Hyperledger Fabric

Hyperledger Fabric ([1]), developed and distributed as an open source software by the Linux Foundation, is one such architecture that allows building custom distributed applications with permissioned blockchains at the backend in a highly modular and scalable fashion. It allows plugging various consensus protocols and advanced security features such as Attribute-Based Access Control strategies with different levels of permission for different users in a totally customized way.

1.3 General Data Protection Regulation and The Right to be Forgotten

Blockchains are made for data sharing and complete visibility. In fact, this is exactly the property of Blockchains that ensures their immutability and reliability. However, this raises many serious security and privacy concerns because data once written to a blockchain can never be mutated or redacted. The European Union's GDPR ([2]) is set of laws and guidelines for collecting personal information from individuals. Amongst many requirements, is the Right to be Forgotten, which essentially states that every individual whose information is collected in a repository, must have the absolute authority to remove that data from further access by all/any other individuals/organizations and hence be "forgotten" from the network. Indeed, it is a reasonable requirement as permanently recorded information is the most susceptible to privacy attacks. This limits the usage of blockchains for enterprises where sensitive personal data is stored, such as hospital databases.

1.4 Proposal Description

The following problem setting is proposed:
Consider 2 hospitals - Hospital1 and Hospital2. Ellie is a patient of Hospital1 on whose private database her medical records are stored. Someday, she is admitted to the ER of Hospital2. In such an emergency situation, her medical records are shared with Hospital2. After recovery, Ellie wishes to exercise her right to be forgotten from Hospital2's database. Our task here is to build an architecture which allows:

- "Sharing" this information in a secure and immutable manner
- "Redacting" this information whenever and from wherever Ellie chooses
- "Proving" (through the consensus of the network) to her that it has indeed been deleted

In this project, we only provide guarantees about the safety and privacy of personal data used within the application. No comments are made about what happens if the data is carried outside the scope of the project. For instance, if Hospital2 decides to make a bit-copy of Ellie's medical records during the time it has access control and stores that bit-copy in some form of local storage invisible to the application, we obviously cannot do anything about it. This is definitely not a problem we aim to solve, that of making "unreplicable" files.

Chapter 2

Related Work

First, we discuss different issues causing tension between blockchains and the RTBF and what is considered "safe" and what is not. ([6]) and ([13]) have summarized many different viewpoints and interpretations of the GDPR's RTBF requirement. It is abundantly clear that plain-text personal data can never be committed on a blockchain. Since we still need verifiable and immutable data, we need to commit either of 2 things:

- The hash of the private data using some powerful cryptographic hashing algorithms ([19]), possibly SHA256 or Double-SHA 256
- The encrypted form of private data using either symmetric or asymmetric algorithms, for instance AES ([17]) and RSA ([18]) respectively.

The GDPR ([2]) states two vaguely similar terms: pseudonymization and anonymization of data. Psuedonymization means that the private data has been obfuscated sufficiently such that it cannot directly be linked to the user to whom it pertains, without additional information which is stored separately, for eg., encrypted data that cannot be decrypted without its key. Anonymization has been defined as a process that irreversibly transforms private data in a way so it cannot be linked to the user anymore. We consider hashing techniques to fall somewhere between both these categories:

- The only information that can help us link a hash to the pre-image is the pre-image itself. And if we already have the pre-image itself, we do not need the hash anymore. Leakage of data itself is needed for this to happen, and if this is possibility, no cryptographic technique can ever be secure at all.
- We can possibly recover the pre-image from the hash if we know about the domain of pre-images and invest sufficient computational power to do a brute force attack. But again, if this is considered feasible, no cryptographic technique is safe. Since performing such attacks is very expensive, we can ignore this threat.

Finally, we conclude that storing the hash of private data on a blockchain can be considered safe if the domain is randomized and large enough to make attacks impractical.
Next, we briefly discuss some approaches to enforcing RTBF in a blockchain.

- ([7]) discusses ways to bypass and ways to remove the immutability of blockchains that allow a trade-off between reliability and the RTBF.
- ([10]) has developed an off-chain data storage approach for allowing confidentiality and enforcing the RTBF for an individual's health data.
- ([9]) has used cryptographic techniques to hash different subsets of private data paired with an off-chain map-based approch for selective visibility and ensuring the RTBF.
- ([8]) has presented an authentication scheme for storing and transmitting data protected by attribute-based encryption based on non-interactive zero-knowledge proofs to minimize the risk of identity theft, though it does not talk about RTBF directly.

Chapter 3

Architecture

This section describes the schematic design, high level architecture, work flow and features provided by the RTBF application. The lower-level implementation details can be found in the chapter 4.
This RTBF application is designed specifically for hospital data in accordance with the Proposal Description in chapter 1.

3.1 Users

The application is intended for 2 kinds of users:

1. Admin: Exactly 1 admin per hospital. The admin is required for registering the users into the system.
2. Patient: Any number of patients per hospital. The application allows each patient to exercise their RTBF.

3.2 Data

The medical records pertaining to different patients' tests is the personal data stored in each hospital's private repository (not made public to the network). Each medical record contains the following personal and non-personal information of a patient:

- Hospital: Name of the Hospital that owns this medical record
- Patient: Name of the patient whose medical record it is (for simplicity, this is considered as the unique identifier for each patient within a hospital)
- Test: Name of the medical test for which this record exists (eg. COVID)
- Result: Result of the test (positive / negative)
- Allergies: List of allergies that are relevant for the this test
- Blood: Blood Group of the patient (extra information stored for convenience)

3.3 Access Control Lists

Each data record (as defined above) has exactly one corresponding access control list. This information is made public, so anyone can verify if some hospital has access to another hospital's private data. Each ACL is actually composed of 2 lists:

1. Read Access List: List of hospitals that are allowed to read a medical record
2. Write Access List: List of hospitals that are allowed to write a medical record

3.4 Functionalities

The following functionalities are provided to the users of the RTBF application. The format is: Function Name (parameters) [types of users allowed to execute this function]:

- Write Private Data (hospital, patient, test, record) [admin]: Admin are allowed to write all medical records to their own hospital's database and only those medical records of other hospitals over which they have write access control. Patients are not allowed to write any medical records.

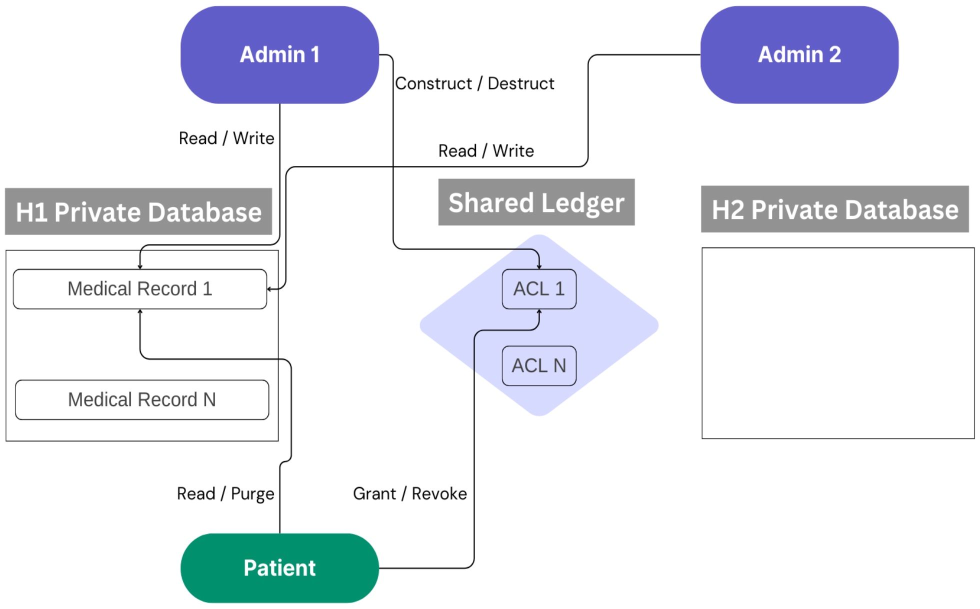
- Read Private Data (hospital, patient, test) [admin, patient]: Admin are allowed to read all medical records from their own hospital's database and only those medical records of other hospitals over which they have read access control. Patients are allowed to read only their own medical records.
- Construct Access Control List (patient, test) [admin]: Before accessing a medical record is allowed (both by oneself or other hospitals), admin of the owner hospital must construct an access control list for the same. Both read and write lists are initially empty, so only the admin and patient can access the data.
- Destruct Access Control List (patient, test) [admin]: Admin of the owner hospital has the authority to destruct the access control list for any medical record of their private database. When the access control list is destroyed, no one can see the record (not even owner hospital's admin). This is good for revoking all access control from others and recreating the empty list without deleting the private data.
- Grant Access Control (test, hospital , manner) [patient]: Patients are allowed to grant read / write / read - write access control of any of their existing

medical records (for a given test) to any other hospital they wish. Only the other hospital's admin has this access, not the patients.

- Revoke Access Control (test, hospital, manner) [patient]: Patients are allowed to revoke read / write / read-write access control of any of their existing medical records (for a given test) from any other hospital whom they had previously granted such manner of access control.
- Purge Private Data (test) [patient]: Patients are allowed to exercise their Right to be Forgotten from the network completely. If they choose to purge the information of any of their tests, that medical record and all historical information pertaining to it is forever deleted from the owner hospital's private repository. No one (other hospital admin / owner hospital admin / even the patients themselves) can now access the data. It is simply erased.

3.5 Work Flow

The following diagram summarizes the architecture:



Chapter 4

Implementation

This section contains the precise implementation details of the RTBF application including the explanations of guarantees given in chapter 3

4.1 Database and Authorization

For simplicity, the current version of the application uses a set of hardcoded admin and patient identities for each hospital listed below. The users need to enter their username and password along with the respective hospital they are logging into on the login page. This first stage enforces simple authorization for further access.

[H1 user]	,	[password]
admin		admin1
Ellie		Ellie1
Dina		Dina1
[H2 user]	,	[password]
admin		admin2
Joel		Joel2
Tommy		Tommy2

Table 4.1: Database of users and passwords

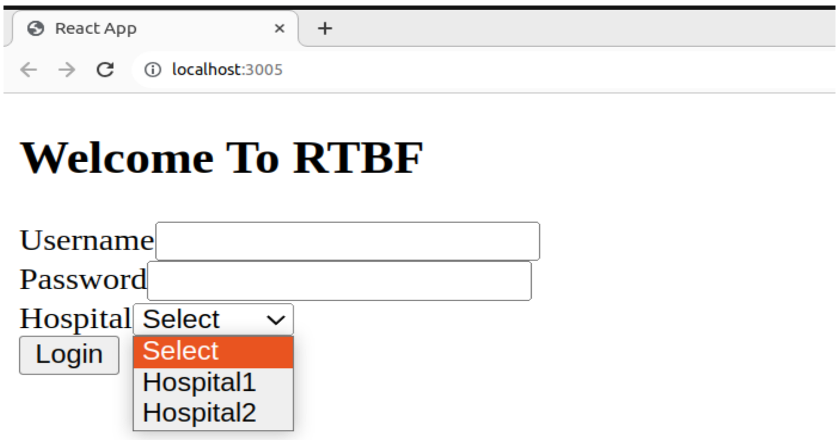
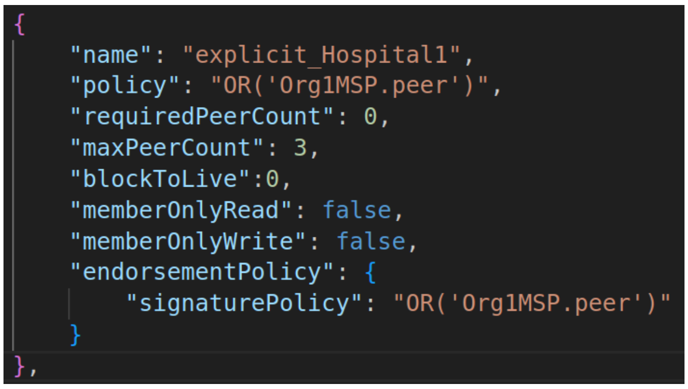


FIGURE 4.1: View of the Login Page

4.2 Private Data Collections

Each hospital stores the medical records of its patients off-chain in a private database. Hyperledger fabric has inbuilt ways of doing this using explicit private data collections. Each explicit private data collection consists of a subset of the organizations connected to the channel (blockchain network) who have controlled access to this data (non-members cannot view this data). When some information is written in private data collections, the hash of that data is written into the shared ledger (visible to all outside the collection). So, when a hospital writes a medical record to it's own private data, it's hash is simultaneously registered on the network automatically. Now, later when the patient decides to share it with another hospital, the other hospital's admin can verify that the off-chain record hashes to match the on-chain hash. This way, every hospital can record data privately without revealing the personal details and others can still verify later that it indeed maintains this data and nothing else. All of these techniques are inbuilt in Hyperledger Fabric.
For our purpose, we make 2 explicit private data collections, one simulating each hospital's off-chain database. Example collection for Hospital 1 is recorded below.



```
{
    "name": "explicit_Hospital1",
    "policy": "OR('Org1MSP.peer')",
    "requiredPeerCount": 0,
    "maxPeerCount": 3,
    "blockToLive":0,
    "memberOnlyRead": false,
    "memberOnlyWrite": false,
    "endorsementPolicy": {
        "signaturePolicy": "OR('Org1MSP.peer')"
    }
},
```

FIGURE 4.2: Private Data Collection Configuration for Hospital 1

The "policy" is set to Org1MSP.peer, meaning this data is only visible to peers of Hospital1. Also, memberOnlyRead and memberOnlyWrite are set to False, since we will over-ride read and write access control by our custom ABAC and not use default settings. "signaturePolicy" is elaborated in the following section.

4.3 Endorsement Policies

Different kinds of endorsement policies are added in this application at different levels to achieve our goal. An endorsement policy specifies the members along with their peers that must approve a transaction before it's added to a block and submitted to the ledger. Endorsement guarantees the legitimacy of a transaction. We discuss 3 levels of endorsement policies and how we override the default settings.

4.3.1 Chaincode-Level EP

This is the default policy used for all transactions of a network defined when the chaincode is installed on peer nodes. By default, it is set to a majority of peers in the network. We override and reduce this to the bare minimum: At least one peer of the network must endorse any transaction. We build on top of this chaincode-level EP as needed.

4.3.2 Collection-Level EP

This is the policy that applies to transactions that access state which is part of a private data collection. Since our private data collections include only the owner organization, we set "signaturePolicy" to the respective organization's peers. It means that only endorsements from the owner organization's peers are valid and necessary for endorsing transactions on it's private data. It says that only the owner hospital's peers can be used for querying/updating its medical records. This means that even after other hospitals are granted access, they still query the owner hospital's peers remotely for accessing their data.

4.3.3 State-Based EP

This overrides all other EPs and can be dynamically set via chaincode execution. A state-based EP applies only to the state of a particular key (can be private or public). We need to ensure that only owner organizations can modify access control lists, which are public. Even though we have ABAC to ensure this, we further make this requirement stronger by setting the state-based EP of all ACLs to the owner organization's peers. It says that only the owner Hospital's peers can be used for querying/updating ACLs of it's medical records.

4.4 Object Structures

We store medical records (private) and access control lists (public) in our blockchain. The exact structure and reasons for choosing such design are described below.

4.4.1 Medical Record Structure

The fields are self-explanatory:

[data , type]	
ID	string
Hospital	string
Patient	string
Test	string
Result	string
Allergies	string
Blood	string

Table 4.2: Medical Record Structure

The "ID" field of the medical record is the unique key by which it is identified within the ledger state. Since a tuple (Hospital, Patient, Test) is considered unique, we can use <Hospital>_<Patient>_<Name> as the key. But for uniformity (in next subsection), we use key = MD5Sum(<Hospital>_<Patient>_<Name>).

4.4.2 Access Control List Structure

The fields are explained below:

[data , type]	
ID	string
Hash(Hospital)	[ ] byte
Hash(Patient)	[ ] byte
Read_Access_List	[ ] string
Write_Access_List	[ ] string

Table 4.3: ACL Structure

The "ID" field of the ACL is the unique key by which it is identified within the ledger state. ID = MD5Sum(<Hospital>_<Patient>_<Name>)_ACL, where the MD5 substring identifies the medical record object to which it corresponds. In case we use plain-text <Hospital>_<Patient>_<Name>, each time we construct an ACL for a new medical record, all members of the network can see this ID and infer that this test was conducted for this patient (because the ACL is public). Of course, they cannot see the actual record details, but they can see that such a record exists. This is still personal information of the patient, and to achieve complete privacy, we use the MD5 hash instead of a plain-text key.

4.5 Attribute Based Access Control

HLF allows us to add custom attributes to users (admin and clients) of the network. We can then implement custom access control logic using attributes of our choice inside the chaincode. We manage access control using 2 attributes:

1. OrgMSP [inbuilt within HLF]: The MSP supplied with a transaction defines which Organization the member belongs to. It cannot be faked, since it requires digital signatures.
2. ClientID [custom - added manually]:
   - Admin: "ClientID" attribute does not exist (not set) for admin users
   - Patient: "ClientID" = patient_<name>_<MSP> is set for each patient by the Admin of the Hospital during registration and enrollment to the network

Using these attributes, in addition to the ACL structure, we have the following access control logic:

4.5.1 Chaincode Access Control

We have already stated which functions can be called by which type of users under Functionalities in chapter 1. Enforcing this is simple by checking if the "ClientID" attribute is set or unset to find out type of user and allow respective functions. Each chaincode function first verifies permission to call the requested function so we can limit who can make transitions to the state.

4.5.2 State Access Control

Before allowing access to the requested medical record, we check if the transaction invoker satisfies one of the 3 qualities:

1. ClientID is unset [admin type user] and Hash(Hospital) (from OrgMSP) matches the corresponding ACL value: This means the user is the admin of the Hospital who owns the record and is allowed read/write permissions to that record.
2. ClientID is set [patient type user] and both Hash(Hospital) (from OrgMSP) and Hash(Patient) (from ClientID) match the corresponding ACL values. This means the user is the patient whose record it is and is allowed read permission.
3. ClientID is unset [admin type user]and Hash(Hospital) (from OrgMSP) is found in one of Read_Access_List or Write_Access_List of the ACL. This means the user an admin of another hospital who has some manner of access control based on which lists contain that hospital's name.

Implcit Access Control: The Hash(Hospital) and Hash(Patient) are the MD5 Sum hashes of the Hospital that owns the record and the Patient whose record it is. When attempting to access this record these credentials are matched against the ACL values (as explained above). So, the owner admin and owner patient do not have to rely upon the read / write lists for access control. Moreover, it guarantees that as long as the data is present, both can access the data and no one can revoke their access by modifying the lists.

Explicit Access Control: The Read_Access_List and Write_Access_List are simply lists of Hospitals that are allowed to read and write to the medical record that the ACL corresponds to. Only the patient is allowed to grant and revoke access control in this list hence ensuring complete autonomy of the user.

Chapter 5

Conclusion

We have understood and can now appreciate the power of blockchains and how Permissoined Blockchains can revolutionize distributed systems by achieving provably reliable decentralization. We have also seen the clash between law and technology in the form of a trade-off between privacy and security. Further, we explored multiple proposals for enforcing the Right to be Forgotten in Permissioned Blockchains, developed and implemented an end-to-end solution for this problem and analyzed the intricacies and implications of the same.

In the future, we plan to expand the capabilities of the project by scaling to an arbitrary number of users and organizations with complex hierarchical structures. We can explore more avenues such as integrating raw file transfer through IPFS instead of limiting to the on-chain data formats and enforcing RTBF for transactions involving data transfer between a set of organizations, possibly across interoperating blockchains. We can move on towards true trustlessness by plugging multi-orderer PBFT services rather than the single orderer RAFT service currently employed. And finally, we may attempt to provide a more complete and understandable proof of information redaction/obfuscation to the users who exercise their RTBF.

Appendix A

Requirements and Source

A.1 List of Tools, Libraries and Frameworks

The implementation of the RTBF application uses the following:

- Hyperledger Fabric 2.4.9 test-network
- GoLang version go1.17.3 for chaincode
- NodeJS v12.22.9 for middle-end
- ReactJS v18.2.0 for front-end (GUI)
- Docker v24.0.6 for making containers in HLF
- Github for version control

A.2 Project Source Link

In spirit of open source, all code is made publicly available at the github URL: `https://github.com/anandparikh4/RTBF_BTP`

Bibliography

[1] https://www.hyperledger.org/

[2] https://gdpr-info.eu/

[3] https://oag.ca.gov/privacy/ccpa

[4] https://ethereum.org/en/

[5] https://bitcoin.org/en/

[6] Rahime Belen-Saglam, Enes Altuncu, Yang Lu, Shujun Li. A Systematic Literature Review of the Tension between the GDPR and Public Blockchain Systems. Blockchain: Research and Applications, 2022

[7] Eugenia Politou, Fran Casino, Efthimios Alepis, Constantinos Patsakis. Blockchain Mutability: Challenges and Proposed Solutions. IEEE Transactions on Emerging Topics in Computing, 2019

[8] Antonio Emerson Barros Tomaz, Jose Caludio Do Nascimento, Abdelhakim Senhaji Hafid, Jose Neuman De Souza. Preserving Privacy in Mobile Health Systems Using Non-Interactive Zero-Knowledge Proof and Blockchain. IEEE Access, 2020

[9] Eugenio Balistria, Francesco Casellatoa, Carlo Giannellib, Cesare Stefanellia. BlockHealth: Blockchain-based secure and peer-to-peer health information sharing with data protection and right to be forgotten. ICT Express, 2021

[10] Aurelie Bayle, Mirko Koscina, David Manset. When Blockchain Meets the Right to be Forgotten: Technology Versus Law in the Healthcare Industry. IEEE/WIC/ACM International Conference on Web Intelligence (WI), 2018

[11] Xinyu Li, Jing Xu, Lingyuan Yin, Yuan Lu, Qiang Tang, Zhenfeng Zhang. Escaping From Consensus: Instantly Redactable Blockchain Protocols in Permissionless Setting. IEEE Transactions on Dependable and Secure Computing, 2023

[12] Imran Makhdoom, Ian Zhou, Mehran Abolhasan, Justin Lipman, Wei Ni. PrivySharing: A blockchain-based framework for privacy-preserving and secure data sharing in smart cities. Computers & Security, 2019

[13] Aloni Cohen, Adam Smith, Marika Swanberg, Prashant Nalini Vasudevan. Control, Confidentiality, and the Right to be Forgotten. arXiv preprint, 2022.

[14] Leslie Lamport. Paxos made simple. ACM SIGACT, 2001

[15] Miguel Castro, Barbara Liskov. Practical byzantine fault tolerance. OSDI, 1999

[16] Diego Ongaro, John Ousterhout. In Search of an Understandable Consensus Algorithm. USENIX ATC, 2014

[17] AES: The Advanced Encryption Standard. Lecture 8 of Computers and Network Security, Purdue University

[18] Public-Key Cryptography and the RSA Algorithm. Lecture 12 of Computers and Network Security, Purdue University

[19] Hashing for Message Authentication. Lecture 15 of Computers and Network Security, Purdue University