A RESTful Privacy-aware and Mutable Decentralized Ledger

Sidra Aslam[1,2][0000−0001−7020−1762] and Michael Mrissa[1,2][0000−0002−2330−1004]

[1] University of Primorska, Faculty of Mathematics, Natural Sciences and Information Technology, Glagoljaška ulica 8, 6000 Koper, Slovenia
[2] InnoRenew CoE, Livade 6, 6310 Izola, Slovenia
{sidra.aslam}@innorenew.eu
{michael.mrissa}@innorenew.eu

Abstract During the last decade, blockchain technology has gained massive attention due to its decentralized, transparent, and verifiable features. However, data stored on the blockchain is publicly available, immutable, and may link to the data owner, thus making privacy management and data modification major challenges. In this paper, we present a RESTful decentralized storage framework that provides data privacy and mutability. To do so, it combines blockchain with distributed hash table, role-based access control, ring signature, and multiple encryption mechanisms. We designed a protocol that exploits metadata and pointers stored on the blockchain, while corresponding encrypted data are stored off-chain, so that data owners are able to control their data. Each peer in our framework offers RESTful APIs to operate, thus ensuring interoperability over the Web. In this paper, we present the operation of our framework and its components that enable data protection at run-time. We also evaluate its performance with time measurements from our proof-of-concept implementation.

Keywords: Decentralized framework · Blockchain · Security · Privacy.

1 Introduction

For several decades, people have been depending on centralized solutions that act as Trusted Third Parties (TTPs) to exchange information and to transfer assets through the Internet. These TTPs are responsible for securing data exchanges and they collect massive amounts of privacy-sensitive information from their users. However, a TTP becomes a single point of failure (SPOF) and is more vulnerable to security breaches and attacks [18]. As a solution to overcome this issue, blockchain [12] has gained massive attention due to its decentralized, transparent and immutable features. Indeed, blockchain allows participants to exchange information and store transactions without any TTP. Concretely, a blockchain is a chain of blocks that contain transactions, and each block is linked to the previous one with a cryptographic signature generated using a hash function.

Adding a block to the chain relies on a consensus algorithm [4], which ensures that the same copy of the transactions in the block are validated by enough (in general, the majority) participants. For the validation to happen, different consensus algorithms (e.g. proof of work, proof of stake, etc.) are available nowadays, with different characteristics (computational cost, complexity, etc.).

However, the availability of the recorded data to everyone in the blockchain network raises issues when it comes to privacy-sensitive data [11]. Besides this, the immutability property of blockchain guarantees that the data records stored in transactions are tamper-proof, i.e. they can neither be deleted nor be mutated, which can be seen as a limiting factor.

In this paper, we aim at addressing these challenges with a single framework that integrates the following contributions:

- a solution for decentralized data storage that combines blockchain and Distributed Hash Table (DHT) to allow for data updates,
- a Role-Based Access Control (RBAC) solution to manage access to privacy-sensitive data,
- a flexible encryption design that allows to choose between multiple types of encryption while storing and querying data on the blockchain,
- a proof-of-concept implementation with performance evaluation that demonstrate the feasibility of our solution.

The rest of the paper is organized as follows. In Section 3, we discuss existing work and their limitations before highlighting the originality of our contribution. Section 4 presents our framework and its components and explains how its provides privacy-preserving, secure, and decentralized data management. Section 5 describes the experimental results that confirm the feasibility of our proposed solution. Finally, Section 6 concludes this paper and lays ground for future work.

2 Motivating Scenario and Research Problem

In this section, we illustrate the need for our work with a wood supply chain scenario. It provides us with the requirements to define our research problem and to design the proposed solution. The wood supply chain includes all activities from the extraction of raw wood from a forest, its transformation, until its sale to the end customer. Traceability in the supply chain allows its stakeholders to understand and guarantee wood origin, transport, processing, and manufacturing. In Fig. 1, we identified the following 6 actors to be typically involved in the wood supply chain:

- **The wood cutting company** identifies specific trees that are useful to make furniture and cuts them into logs.
- **The transport company** picks raw wood (e.g logs) from the forest and transports them into a storage warehouse for further processing.
- **The storage warehouse company** sorts, processes, and stores logs temporarily.
- **The furniture assembly company** cuts logs into pieces to assemble the furniture.
- **The furniture shop company** exposes assembled furniture and conducts sales with the final customer.
- **The customer** buys wooden furniture and verifies product origin.

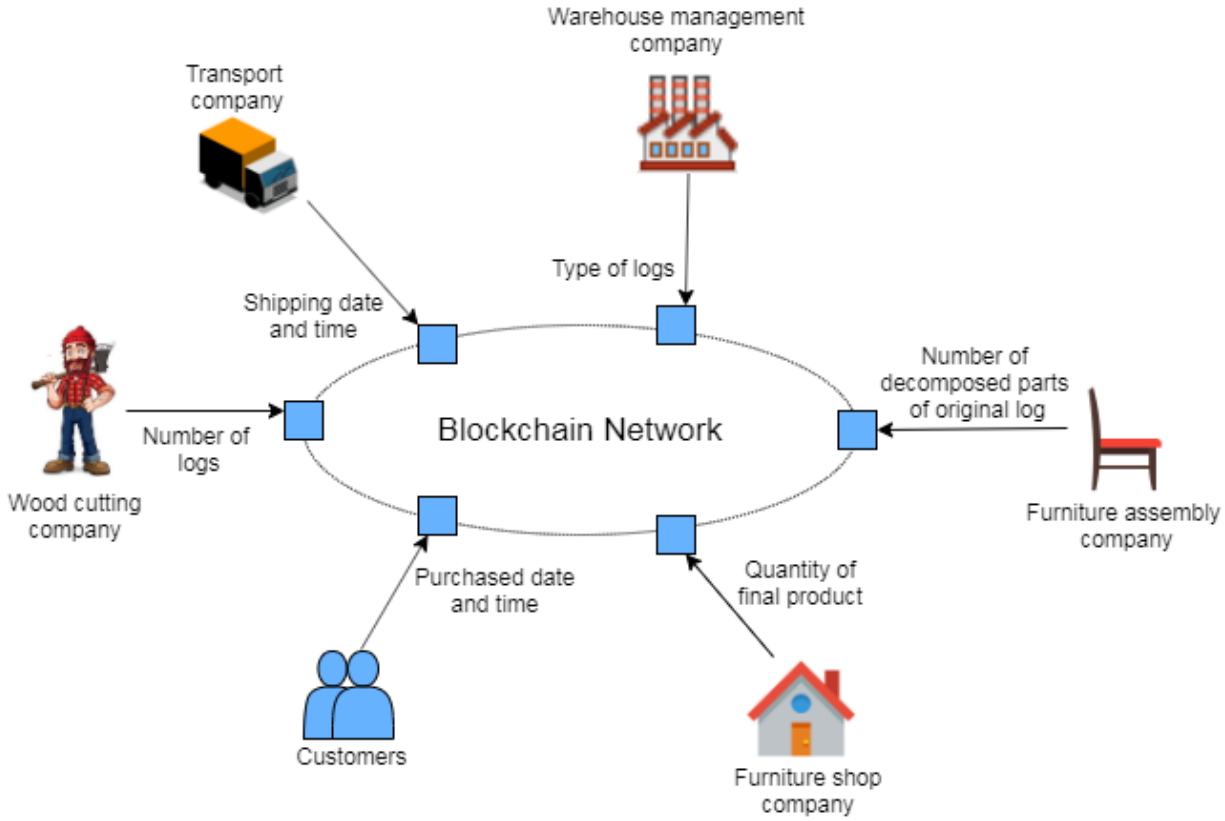


Fig. 1. Overview of wood supply chain process.

Our wood supply chain scenario highlights the need for trust and traceability in the supply chain process. Typically, traceability is very important in the supply chain, allowing all stakeholders to be able to trace products at each point [17]. Existing solutions rely on Radio Frequency Identification (RFID) technology to enable electronic traceability of wood in the supply chain. Generally, this traceability framework needs a third-party centralized database framework to collect and store RFID data, which leads to a Single Point Of Failure (SPOF). Decentralized storage solutions can solve the problem of a single point of failure. In particular, blockchain is a decentralized and distributed ledger technology that stores records of users in such a way that makes them accessible to all participants without the

risk of SPOF. A blockchain consists of a linear sequence of blocks. The contents of each block[3] contains a hash of the contents of the previous one to prevent the modification of stored transactions [14]. If the previous block is modified, the new hash one could generate from the content upon verification would not match the one stored in the next block. This design provides the blockchain with its immutability feature [9]: once data has been stored, no one can modify it.

However, our wood supply chain scenario highlights that actors need to insert, retrieve and delete data about their business activities, and at the same time, they need to be able to modify data, while keeping the proof that data was inserted. There is a need to develop a solution that overcomes the immutability characteristic of blockchain to allow update and delete operations on stored data. At the same time, the developed solution must fine-grained access control, as data access permissions vary depending on the data requester identity (data owner, business partners, client).

2.1 Research Problem

According to wood supply chain scenario discussed above, using blockchain technology in supply chain requires taking into account the following issues:

- **Data modification management:** In our case study, actors may need to modify data in blockchain (e.g number of logs and product type). However, blockchain does not allow data modification, once it has been added to the chain due to its immutability nature. The challenge consists in overcoming this limitation while keeping the properties that make the blockchain interesting.
- **Data security and access control:** Blockchain stores data publicly and allows anyone to access it. In our context, a decentralized solution should ensure data privacy and protect privacy-sensitive data from unauthorized access.

Therefore, according to wood supply chain scenario and limitations discussed above, the proposed solution must satisfy the following requirements:

- **Data protection:** Users' data (e.g name, identity) and product data (e.g location) records on the blockchain must be protected from unauthorized access. Such data may be sensitive and user does not want to expose their data publicly on the blockchain.
- **Data security:** Our framework must ensure the following data security properties: 1) confidentiality to make sure that data must not available to unauthorized user, 2) integrity to verify that data contents are original and unauthorized user can not alter data, 3) availability to make sure that data must be available to the authorized user, and 4) non repudiation, so that once data is added to the chain no one can deny its existence (such as proof that a log has been transported or processed).
- **Anonymity:** In some cases, the framework must make sure that other actors on the network may not be able to link data with their owner. Typically, the data owner may want to protect the link between their identity and the data they store on the blockchain.

In summary, due to its public nature, blockchain data is available to everyone and may link to its users which raise privacy and security issues. Besides this, blockchain data cannot be deleted and modified thus making data modification major concerns. In particular, data mutability, confidentiality, and identity privacy are vital challenges for blockchain implementations. In the following, we discuss the limitations of decentralized solutions supporting privacy-aware data access and data update.

3 Related Work

In this section, we present the related work and its limitations. First, we discuss existing work to store data on blockchain. Second, we provide papers related to data updates on blockchain.

3.1 Blockchain-based data storage

In [10], a blockchain-based software connector framework is presented. The proposed framework is used to share information between companies and stakeholders. Hash sum of the data is stored on blockchain whereas original data is stored on a MySQL database. However, MySQL databases are centralized and subject to the SPOF problem, in

[3] Except the first block called genesis block.

addition they are not as scalable as DHTs to store large amounts of data [8]. In our work, we rely on DHT to store data, which does not form a SPOF and better handles large amount of data.

In [3], the authors present a blockchain-based framework called u-share for data sharing. The proposed framework enables users to control and trace the data they share with their family, friends, and others. A software client is used to manage the sharing of the private key with its circle members. It maintains the record of shared keys and ensures that the shared data is encrypted with the circle's public key. However, private key sharing is subject to security issues. Additionally, the this framework uses one type of encryption. In contrast, our framework enables actors to stay in control of their keys and does not allow the sharing of private key as a recommended good practice. Our solution also allows actors to use multiple types of encryption methods while storing and querying data on blockchain.

In [19], the authors propose a decentralized supply chain framework to maintain the traceability of goods and recipe ingredients. A smart contract is used to manage the exchange of goods on distributed ledger. However, product data is accessible publicly and immutable, which leads to privacy and data updates issues. In contrast to this solution, we store encrypted data on off-chain storage to maintains data privacy. Our framework enables actors to update data at each point of the chain.

In [7], the authors present a blockchain-based supply chain framework to maintain food traceability using a smart contract. The proposed framework stores data hash on a blockchain whereas actual data are stored on IPFS (point-to-point distributed hypermedia distribution protocol) off-chain storage. However, a manufacture node server is used to manage the entire framework modules, which leads to a single point of failure. In contrast, we propose a fully decentralized framework without any central server. Our solution relies on a registry server that enables nodes to connect with each other, however, a decentralized discovery protocol can also be used instead of this registry server [6].

In [16], the authors propose a blockchain-based agri-food supply chain traceability framework. RFID (Radio-Frequency IDentification) is used to manage and identify products through radio-frequency signal. However, blockchain has a limited size of a block to store transactions. In our solution, we overcome this limitation by storing only metadata and pointer on blockchain, while original data is stored on off-chain storage.

3.2 DHT-based data storage

In [20], the authors propose a blockchain-based personal data management framework that combines blockchain with DHT. The proposed framework stores encrypted data and shared key on DHT while the corresponding pointer is stored on blockchain. It allows service and user to query the data using the pointer. However, this framework uses one type of encryption (shared symmetric key) to encrypt/decrypt data. Additionally, the paper does not explain how they protect the symmetric key from unauthorized access. Their work relies on fine-grained access control to access blockchain. However, the authors did not clearly explain who has permission to read, write, and update data. Permissions are stored on blockchain and lead to immutability issues. In contrast, our framework provides multiple types of data encryption methods depending on the actor's requirements. We encrypt a symmetric key with the owner's public key that allows only owner to access it using their private key. Our solution is flexible and simple because it allows the owner to define and modify their access control policies.

In [15], the authors present a distributed access control and data management framework. The proposed framework combines blockchain with DHT for secure IoT data sharing. Blockchain is used to manage and store access control permissions, which is publicly visible and leads to privacy issues. However, access control permissions are unable to modify due to blockchain immutability feature. In contrast to this solution, our proposed solution is flexible to update access control permissions. We also ensures data owner anonymity while sharing data.

The authors in [1], present a blockchain-based data storage for PingER (Ping End-to-End Reporting). The proposed solution use permissioned blockchain to store metadata of PingER files whereas corresponding data are stored on DHT without any encryption mechanisms. Additionally, this framework stores monitoring agent name and upload locations of the file on the blockchain, which raises data security and privacy issues. Typically, blockchain users want to protect their sensitive information such as username and data location from unauthorized access. They may not want to disclose their sensitive information on the blockchain. Inspired by the PingER metadata structure, our framework extends metadata structure and enables privacy and security management that ensures authorized access control and privacy protection.

In [5], the authors propose a permissionless blockchain-based LightChain framework that replicates each block and transaction within the peers of DHT. The proposed solution allows every peer to access blocks and transactions using a skip graph. However, blocks and transactions are publicly accessible which leads to data security issues. As

compared to this framework, we use RBAC to control unauthorized access to blocks and transactions. Our solution stores metadata on blockchain that ensures data traceability.

As a summary, we have identified the most relevant work related to blockchain and DHT data storage. To the best of our knowledge, this is the first paper that provides decentralized data storage, data mutability, manages access to privacy-sensitive data, multiple types of encryption, and message sender anonymity at the same time in a single solution. In the following, we discuss the steps of our proposed framework in detail.

4 Proposed Framework

In this paper, we propose a secure privacy-aware decentralized framework that supports role-based access control, data mutability and actor's anonymity. Each actor, as a peer of the framework, runs the same code that is structured into a set of components. The following subsections describe each of these components in detail as depicted in Fig. 2.

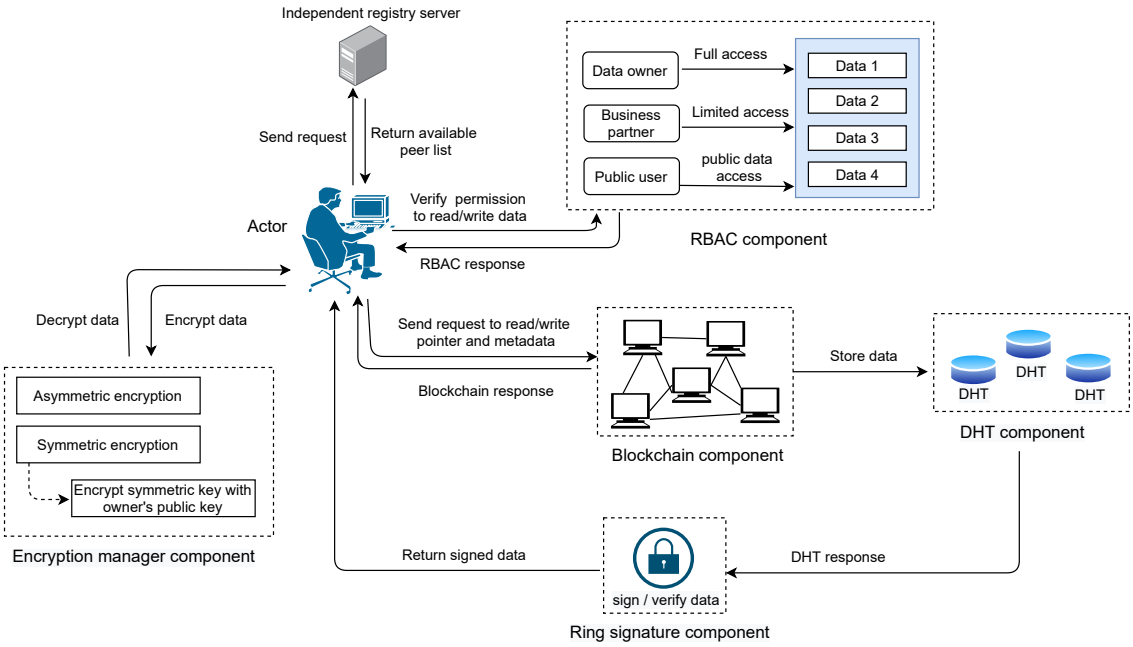


Fig. 2. Overview of a peer architecture.

4.1 General Overview

Our proposed framework allows wood supply chain actors to store data, read data upon request, and communicate with other actors through HTTP calls. Fig. 2 shows the layout of our framework and its components organized around a `main` program. In our framework different actors are running the `main` program and they connect to each other using their APIs, after an initial call to the `registry_server` to get the list of available peers.

Let us consider the following example: an actor such as a wood cutter logs in our framework to store the number of logs cut on this day. When the program starts, the wood cutter will send a `'POST'` request to the `/peers` resource of the `registry_server` to add its public key and URL (Uniform Resource Locator) to the list of connected peers[4]. Then, it will call the `/peers` resource with the `'GET'` method and retrieve the list of connected peers. It will then connect with other available peers to download a copy of the blockchain (`/chain` resource, method `'GET'`, available on each peer).

[4] Please note that the registry server can easily be replaced by a decentralized discovery protocol like Chord4S [6].

Upon request, the `main` component will call the `rbac_manager` component to verify the current actor's permission such as wood cutter to perform read and write operations. Indeed, each actor's roles, resources, and permission are defined in this `rbac_manager` component.

Our framework allows the authorized actor to choose different types of encryption methods while storing data and generates a public key, private key, or symmetric key accordingly. Before storing the data, the `main` component will call the `encrypt_manager` component to encrypt the entered data with the current actor's public key or symmetric key depends on the selected encryption method. For each actor, the `encrypt_manager` component is responsible for generating public and private keys. This encrypted data sent to off-chain (key-value) storage called `DHT_manager` component, while corresponding pointer and metadata are stored on a `blockchain_manager` component (pointer is the hash of the data).

An authorized actor allows to create, update, delete, and read data using the pointer stored on the public ledger. A request (`/chain/<block_no>`, method `'GET'`) to the `main` component might call the `ring_signature` component to sign data anonymously, only in the case where the data is privacy-sensitive and the role of the requester requires anonymization. Accordingly, a request to (`/chain/<block_no>`, method `'POST'`), will create a block, or update it if it already exists, and process the contents sent in the request message.

The following subsections describe each of these components in detail.

4.2 Framework components

In this section, we describe in detail the components of the proposed framework including decentralized data storage, authorized data access, ensure data traceability, and maintains the actor's anonymity.

RBAC manager component: we use a Role-Based Access Control (RBAC) model to manage access to privacy-sensitive data. The RBAC model is based on the following four parameters: user, role, resource, and permission. In RBAC, users are actors related to the application. Roles are the application's functions that allow to access resources based on the given permissions. A permission is an authorization to access one or more resources within the application [2].

In our work, we define the following users, roles, resources, and permissions that assigns permissions to the user based on their role in our wood supply chain scenario.

- *Users:* In our framework, we need to define RBAC users according to the actors of the wood supply chain. Therefore, we define the following users: wood cutter, transporter, warehouse manager, furniture assembler, furniture seller, and customer.
- *Roles:* According to the different actions our supply chain users can perform on the architecture, we define the following roles:
  - *Data owner:* Any user[5] can be data owner. Data owners can add, read, modify and delete data about their products. For example, a wood cutter would act as "data owner" and insert information such as (date:1.1.2021, trees-cut:20, type:oak, price:20 Euro, margin:20%).
  - *Business partner:* The business partner role allows specific users (chosen by the data owner) to access data that is not available to anyone. For example, a furniture assembler would act as "business partner" and might be allowed to read from the previous example: (trees-cut:20, type:oak, price:20 Euro).
  - *Public reader:* The public reader role gives access to all public data. For example, a customer would act as "public reader" to monitor the origin of a product and might be allowed to read from the previous example: (type:oak).
- *Resources:* In our framework, user can access resources according to defined roles and permissions. In our framework, we define the following resources:
  - *DHT:* User can access DHT resource to add data about their business activities. For example, a wood cutter has a role "data owner" and store information such as (date:1.1.2021, trees-cut:20, type:oak, price:20 Euro, margin:20%).
  - *Blockchain:* User allows to access blockchain resource to read data. For example, a customer has a role "public reader" and might be allowed to read information such as (date:1.1.2021).

[5] Except the end client that has read-only access

- *Permissions:* We define permissions to restrict user's actions to access resources. For example, from previous example. a wood cutter has a role "data owner" and has a "permission" to write, read, update, and delete data such as (trees-cut:30, type:maple, price:50 Euro), whereas transport company would act as a "business partner" and has only "permission" to read information such as (trees-cut:30).
- *Rules and policies:* Our framework defines rules and policies that controls access to the data such as private data, privacy-sensitive data, and public data. Our `rbac_manager` component is responsible to authenticate role of current login actor. It also ensures if current role has permission to access resource or not as denoted by *verify_permission (role, operation, resource)*.
  For example, wood cutter has a role 'business partner' logs into the framework to store data on blockchain. The `main` component calls the method `authenticate(actor, role)` to authenticate that if a 'business partner' role exists in our `rbac_manager` component or not. After role authentication, the `rbac_manager` component verifies the permissions of actions for current login actor's role such as if `(actor_role == 'owner')`, then "owner" has permission to perform read, write, update, and delete all types of data on the blockchain. In case, if `(actor_role == 'business_partner')`, then our framework allows just to read some data such as privacy-sensitive and public data. If `(actor_role == 'public_user')`, then our framework provides access to just read public data.
  Our framework provides filter access based on role such as wood cutter as a 'business partner' has not permission to write, update, and delete data. We maintain data security by limiting unnecessary access to sensitive data based on each actor's role. Please note that although this simple RBAC model answers the requirements of our scenario, more elaborate models could be plugged in without changing anything in the framework design.

Blockchain component: We use `blockchain_manager` component to manage metadata and pointer of encrypted data. Our proposed metadata structure consists of the data entry date, data entry time, and data pointer. The main components of the blockchain include block transaction, consensus algorithms, and metadata extension. Each component is explained as follows.

- *Block transaction:* Each block contains the block header, consensus signature, hash of the previous block, timestamps, verified metadata, and pointer of the actual data. Each block has a unique hash value, which maintains the integrity of the entire blockchain from the first block (genesis block) to the last block in the network [13]. In our framework, actors will connect to the framework and call `initialize(chain)` method to copy the blockchain if there will be any other available actors on the network, otherwise genesis block will be created and added to the blockchain.
  A blockchain is composed of a chain of the blocks where each block is comprised of many transactions [13]. Each transaction is broadcast on the network for verification and miners verifies the transaction through signature. Then, the verified transactions are added to the block of the blockchain. After storing verified metadata and pointer on the blockchain, our framework returns the block number to the data owner. The proposed framework allows data owner to access specific block from the blockchain by using block number. The data owner can read, update, and delete data from this specific block.

- *Consensus mechanism:* It is used in our `blockchain_manager` component to establish the agreement on one state of the data in a distributed network. It ensures that the same copy of the data is replicated to all nodes in the blockchain network. Further, it verifies the transactions from this block and prevents the attacker to change the state of the data. Our framework uses a proof of work consensus mechanism to add each block to the blockchain. To do so, miners solve the complex puzzle and receive a reward such as a new coin to validate the block. Miners validate the transactions in a block and add this block to the blockchain. Proof of work consensus mechanism prevents a malicious actor to compromise more than half of the hashing power on the blockchain. The process to verify the proof and its correctness is easy and fast. In the following we define the proposed metadata structure.
- *Metadata extension:* In [1], the authors allow storing metadata in the blockchain. We follow a similar approach and store the metadata information for each piece of data to maintain product traceability and actors' trust.
  In our framework (see Fig. 2), we have an `RBAC_manager` component to restrict user's actions on the data and we use a `blockchain` component to store metadata and pointer of actual data that are stored on the `DHT` component. We integrate all these components with each other to work together. We use REST APIs (`/chain`) that allow actors to copy blockchain and to store and read data on the distributed framework. Using REST APIs present many benefits, amongst them the possibility to use a generic HTTP client for any communication between nodes, better performance and scalability.

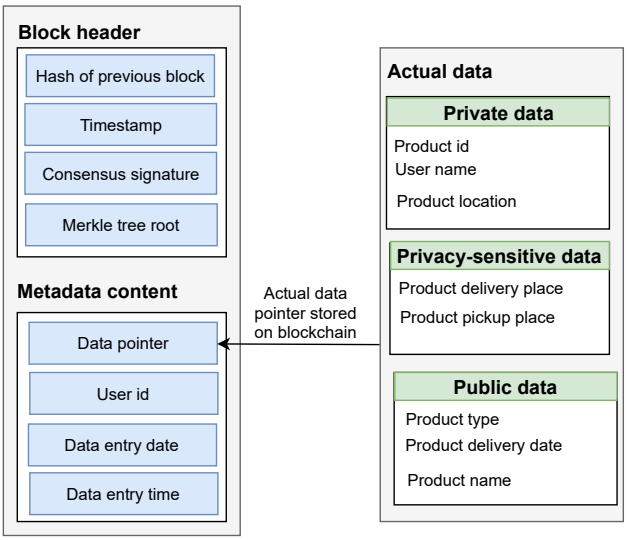


Fig. 3. Metadata structure on blockchain

We propose a metadata extension that relies on paper [1], to handle privacy constraints on data. To do so, we propose to encrypt user's sensitive information (e.g location) with encryption mechanisms, and we store this encrypted data on offline storage (DHT).
In our sample scenario, actual data on DHT consists of an actor's name, product identity, product location, quantity, and wood type. Fig. 3 illustrates the metadata structure on blockchain that contains the data entry date, data entry time, and data pointer.

DHT component: In the proposed framework the encrypted data of each actor is stored on off-blockchain (key, value) storage called DHT. We implement a DHT component of our framework by using the Kademlia library. DHT is comprised of network of nodes that enable actors to write/read data associated with a given key. Actor's data are randomized across the nodes of the network and replicated to eliminate the chance of data loss. Our proposed framework records the date and time of each new data entered by the actor. This enables a network to keep track of the product and maintains the order of product entries.

Encryption manager component: In our framework, the `encrypt_manager` component is responsible for data encryption and decryption according to the selected encryption method. Our framework allows actors to choose encryption methods for each data write operation. If the data owner chooses the asymmetric encryption method then data will be encrypted with the owner's public key and stored encrypted data on DHT. A public key is accessible publicly while the private key is kept private by the key's owner to decrypt the data. If the data owner chooses the symmetric encryption method then data will be encrypted with a symmetric key and this symmetric key again will be encrypted with the owner's public key to ensure that only the data owner can access it later. Both encrypted symmetric key and encrypted data will be stored on DHT.

Ring signature component: It is an option here to actor's ensure anonymity within a group. A signature is created by any member from a set of public keys called a ring. Therefore, the identity of the signer remains hidden and no one can identify that who is the actual signer of the data. In our framework, the data owner can allow other actors to read their data upon request by using(`/chain/<block_no>`, method `'GET'`). To read data, we rely on encryption according to data reading requirements:

- Private data will not be shared with anyone. Therefore, it will be encrypted with the owner's public key, so only the owner can decrypt data using their private key.
- Privacy-sensitive data is shared with only a specific number of users. It will be encrypted using the receiver's public key, so later data can be decrypted only with the corresponding private key. The data owner will also sign data by using ring signature to remain anonymous within a group, An authorized requester can read data and verify the signature.

– Public data is available to anyone. It will optionally be signed by ring signature or encrypted with the data owner's public key to guarantee data ownership.

5 Implementation and Evaluation

This section discusses the implementation and evaluation of our proposed work. We discuss the implementation details in section 5.1 and section 5.2 presents the evaluation setup.

5.1 Implementation

We implemented the key components of our framework by using an open source blockchain library[6] and the Kademlia DHT library[7]. The blockchain library is used to achieve consensus on a distributed network and creation of blocks. While, we used the DHT to store and retrieve data link with a key in a network of peer nodes. We performed all the experimental process using Python 3. The experiments are performed on the data (privacy-sensitive, private, and public) entered by the actors into the framework.

5.2 Evaluation

We evaluated the key components of our proposed framework on 64-bit Microsoft Windows Operating System with 16GB of memory. In the following, we discuss the qualitative security and privacy analysis as well as quantitative performance evaluation.

Security analysis: According to the design of proposed framework, only authorized actors are allowed to access the system to perform write, read, update, and delete operations. A malicious user cannot modify existing data unless he/she controls more computation power than all other miners.

Our framework ensures following security properties: we achieve `confidentiality` using asymmetric and symmetric encryption. We encrypt data with the owner's public key and store the corresponding pointer on the blockchain to achieve `integrity`. Our framework archives `availability` through the access control model. We ensures `non-repudiation` by adding metadata to the chain.

Linking attack: Our framework uses a unique public key for each transaction. It prevents a malicious user to link multiple data and transactions with the same ID.

Modification attack: In our solution, data owner has ability to encrypt data with their public key and store hash of the encrypted data on the blockchain. It also records evidence of data entry date and data entry time to trace last modification of data. An attacker can not modify owner's data.

Privacy: Our proposed solution ensures that the owner owns and control their private data. Actor's private data will not be shared with other actors on the network. To share privacy-sensitive data and public data with other actors, this data will be encrypted using requester's public key to protect the data from malicious actors who tries to read the data during data sharing process. In our proposed solution, we achieve anonymity using ring signature.

Scalability: Currently, we tested our prototype with six actors and achieve reasonable performance. Our framework is flexible and scalable to work with a large number of actors.

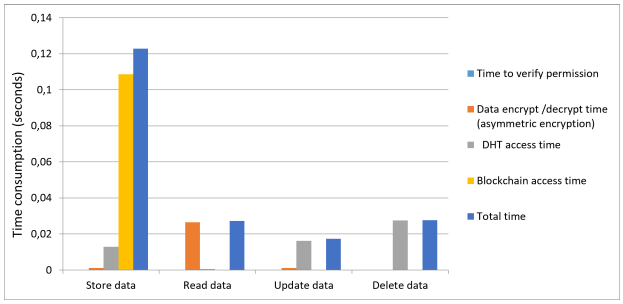
Performance evaluation: We evaluate the time overhead to verify permission, data encryption/decryption using a symmetric or asymmetric method, DHT access, blockchain access, and overall total time while data store, read, update and delete operations. Fig. 4 outlines the time processing for both asymmetric encryption without ring signature 4(a) and asymmetric encryption with ring signature 4(b).

The results demonstrate that the total time of asymmetric encryption without ring signature is larger than the total time of asymmetric encryption with ring signature while store, update and delete data.

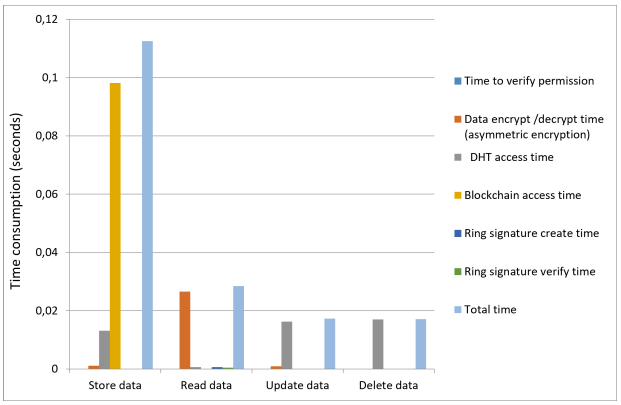
We calculated the overall time for symmetric encryption as depicted in Fig. 5. We compare results symmetric encryption without ring signature 5(a) with symmetric encryption using ring signature 5(b). It is seen from the results that the total time of storing and deleting data for symmetric encryption without ring signature is larger than the symmetric encryption with ring signature. The total time to read data for symmetric encryption without ring signature is less than the symmetric encryption with ring signature. Total time to update data for both 5(a) and 5(b) are not much affected by the ring signature and symmetric encryption.

[6] https://github.com/satwikkansal/python_blockchain_app/tree/ibm_blockchain_post.
[7] https://github.com/bmuller/kademlia

We also calculated average, standard deviation, min, and max value for both asymmetric and symmetric encryption while store, read, update, and delete data. We ran our prototype 50 times and experimental results show that asymmetric encryption gives a standard deviation of 0,022 seconds and symmetric encryption has a standard deviation of 0,023 seconds during data storing operation. To read data, asymmetric encryption has a minimum value of 0,124 seconds and symmetric encryption gives 0,142 seconds. For data update operation, asymmetric encryption has maximum value of 0,068 seconds and symmetric encryption gives 0,052 seconds maximum value. Experimental results clearly show that our proposed framework achieves a low overhead that is acceptable for the actor.

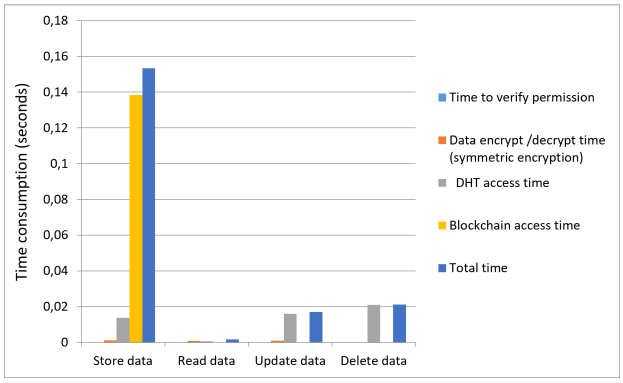


(a) Time overhead of asymmetric encryption without ring signature

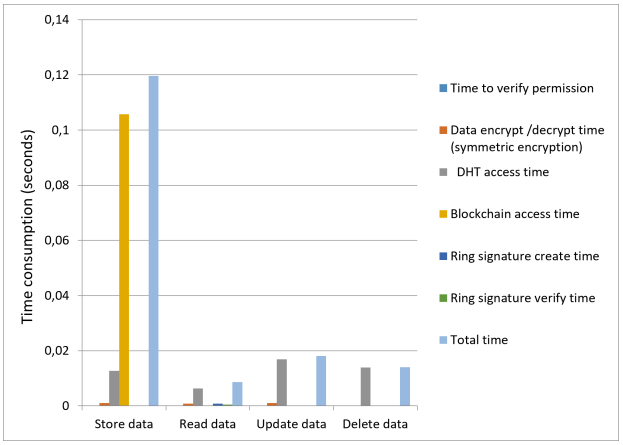


(b) Time overhead of asymmetric encryption with ring signature

Fig. 4. Overall time overhead for asymmetric encryption



(a) Time overhead of symmetric encryption without ring signature



(b) Time overhead of symmetric encryption with ring signature

Fig. 5. Overall time overhead for symmetric encryption

6 Conclusion

In this paper, we illustrate the need for privacy-aware decentralized data storage, access control, data mutability, and actor anonymity in the wood supply chain scenario. Our framework enables this by combining the blockchain with DHT, role-based access control, and multiple encryption mechanisms that allow only authorized actors to access and modify their data without disclosing their identity on a distributed ledger.

Thanks to its RESTful (between peers) and component-based (inside a peer) design, our framework is fully reusable across the wide diversity of possible application domains and use cases. We also presented a performance evaluation regarding its operation. Our simulation results demonstrate that our framework shows promising results and achieves an acceptable overhead.

To the best of our knowledge, this research is the first work that integrates this combination of technologies in a single framework. In future work, we plan to compare our solution to similar blockchain implementations. Furthermore, we will study how the behaviour of our prototype evolves over larger number of peers, and devise optimizations to improve its performance over large scale networks, in real or simulated environments.

Acknowledgment

The authors gratefully acknowledge the European Commission for funding the InnoRenew project (Grant Agreement #739574) under the Horizon2020 Widespread-Teaming program and the Republic of Slovenia (Investment funding of the Republic of Slovenia and the European Regional Development Fund). They also acknowledge the Slovenian Research Agency ARRS for funding the project J2-2504.

References

1. Saqib Ali, Guojun Wang, Bebo White, and Roger Leslie Cottrell. A blockchain-based decentralized data storage and access framework for pinger. In *2018 17th IEEE International Conference on Trust, Security and Privacy in Computing and Communications/12th IEEE International Conference on Big Data Science and Engineering (TrustCom/BigDataSE)*, pages 1303–1308. IEEE, 2018.
2. Elisa Bertino. Rbac models - concepts and trends. *Computers & Security*, 22(6):511–514, 2003.
3. Antorweep Chakravorty and Chunming Rong. Ushare: user controlled social media based on blockchain. In *Proceedings of the 11th international conference on ubiquitous information management and communication*, pages 1–6, 2017.
4. Tien Tuan Anh Dinh, Rui Liu, Meihui Zhang, Gang Chen, Beng Chin Ooi, and Ji Wang. Untangling blockchain: A data processing view of blockchain systems. *IEEE Transactions on Knowledge and Data Engineering*, 30(7):1366–1385, 2018.
5. Yahya Hassanzadeh-Nazarabadi, Alptekin Küpçü, and Öznur Özkasap. Lightchain: A dht-based blockchain for resource constrained environments. *arXiv preprint arXiv:1904.00375*, 2019.
6. Qiang He, Jun Yan, Yun Yang, Ryszard Kowalczyk, and Hai Jin. A decentralized service discovery approach on peer-to-peer networks. *IEEE Transactions on Services Computing*, 6(1):64–75, 2011.
7. Haihui Huang, Xiuxiu Zhou, and Jun Liu. Food supply chain traceability scheme based on blockchain and epc technology. In *International Conference on Smart Blockchain*, pages 32–42. Springer, 2019.
8. Nattawat Khamphakdee, Nunnapus Benjamas, and Saiyan Saiyod. Performance evaluation of big data technology on designing big network traffic data analysis system. In *2016 Joint 8th International Conference on soft computing and Intelligent Systems (SCIS) and 17th International Symposium on Advanced Intelligent Systems (ISIS)*, pages 454–459. IEEE, 2016.
9. M Vinod Kumar and NCS Iyengar. A framework for blockchain technology in rice supply chain management. *Adv. Sci. Technol. Lett*, 146:125–130, 2017.
10. Francesco Longo, Letizia Nicoletti, Antonio Padovano, Gianfranco d'Atri, and Marco Forte. Blockchain-enabled supply chain: An experimental study. *Computers & Industrial Engineering*, 136:57–69, 2019.
11. Malte Moser. Anonymity of bitcoin transactions. In *Münster Bitcoin Conference (MBC), Münster, Germany*, July 2013.
12. Satoshi Nakamoto and A Bitcoin. A peer-to-peer electronic cash system. *Bitcoin.–URL: https://bitcoin. org/bitcoin. pdf*, 4, 2008.
13. Michael Nofer, Peter Gomber, Oliver Hinz, and Dirk Schiereck. Blockchain. *Business & Information Systems Engineering*, 59(3):183–187, 2017.
14. Alex Pazaitis, Primavera De Filippi, and Vasilis Kostakis. Blockchain and value systems in the sharing economy: The illustrative case of backfeed. *Technological Forecasting and Social Change*, 125:105–115, 2017.
15. Hossein Shafagh, Lukas Burkhalter, Anwar Hithnawi, and Simon Duquennoy. Towards blockchain-based auditable storage and sharing of iot data. In *Proceedings of the 2017 on Cloud Computing Security Workshop*, pages 45–50, 2017.
16. Feng Tian. An agri-food supply chain traceability system for china based on rfid & blockchain technology. In *2016 13th international conference on service systems and service management (ICSSSM)*, pages 1–6. IEEE, 2016.
17. Ioakeim Tzoulis and Zaharoula Andreopoulou. Emerging traceability technologies as a tool for quality wood trade. *Procedia Technology*, 8:606–611, 2013.
18. Shangping Wang, Yinglong Zhang, and Yaling Zhang. A blockchain-based framework for data sharing with fine-grained access control in decentralized storage systems. *Ieee Access*, 6:38437–38450, 2018.

19. Martin Westerkamp, Friedhelm Victor, and Axel Küpper. Blockchain-based supply chain traceability: Token recipes model manufacturing processes. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 1595–1602. IEEE, 2018.
20. Guy Zyskind, Oz Nathan, et al. Decentralizing privacy: Using blockchain to protect personal data. In *2015 IEEE Security and Privacy Workshops*, pages 180–184. IEEE, 2015.