Computer Science and Information Systems 20(3):1235–1261 https://doi.org/10.2298/CSIS220110007A

A Framework for Privacy-aware and Secure Decentralized Data Storage*

Sidra Aslam[1,2] and Michael Mrissa[1,2]

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

Abstract. Blockchain technology gained popularity thanks to its decentralized and transparent features. However, it suffers from a lack of privacy as it stores data publicly and has difficulty to handle data updates due to its main feature known as immutability. In this paper, we propose a decentralized data storage and access framework that combines blockchain technology with Distributed Hash Table (DHT), a role-based access control model, and multiple encryption mechanisms. Our framework stores metadata and DHT keys on the blockchain, while encrypted data is managed on the DHT, which enables data owners to control their data. It allows authorized actors to store and read their data in a decentralized storage system. We design REST APIs to ensure interoperability over the Web. Concerning data updates, we propose a pointer system that allows data owners to access their update history, which solves the issue of data updates while preserving the benefits of using the blockchain. We illustrate our solution with a wood supply chain use case and propose a traceability algorithm that allows the actors of the wood supply chain to trace the data and verify product origin. Our framework design allows authorized users to access the data and protects data against linking, eavesdropping, spoofing, and modification attacks. Moreover, we provide a proof-of-concept implementation, security and privacy analysis, and evaluation for time consumption and scalability. The experimental results demonstrate the feasibility, security, privacy, and scalability of the proposed solution.

Keywords: Blockchain, Distributed Hash Table, Security, Privacy, Decentralized framework.

1. Introduction

With increasing the number of internet users, large amounts of data are being generated each day [18]. Cloud computing provides the facility to store, access, and share data with other users anytime. The main limitation of the cloud paradigm is its centralized storage design, which leads to a single point of failure issue. Cloud storage systems rely on Trusted Third Party (TTP) to collect and store users' privacy-sensitive data, which is more vulnerable to security and attacks. To address these challenges, blockchain has become popular as a decentralized and transparent data management facility [23], [42]

* This is an extended version of our previous paper [2].

that enables users to share and store information without any TTP. A blockchain is a peer-to-peer distributed ledger in which a list of records called blocks are linked with each other and secured using a cryptographic hash function [35]. It stores data on distributed nodes through a consensus mechanism that guarantees participant's trust by having the same copy of the data [34], [37].

However, blockchain allows anyone to read and write contents, which may raise data security issues [40], and does not handle privacy-sensitive data [21] by default. This is a limitation since data owners may not want to disclose their sensitive information (e.g statistics about their business activities) on the blockchain. Scalability is also an issue, as the data is replicated on every peer, storing large quantities has a prohibitive cost. Besides this, immutability of blockchain, while an important feature, prevents data modifications.

In this paper, we propose a privacy-aware decentralized data storage and management framework that enables actors to write, read, delete, update, and access their transactions history. Our solution allows data owners to control and secure their data in a decentralized ledger. Building on previous work [2], our proposed framework is scalable enough to handle an increasing number of actors while performing data write, read, update, and delete operations. The main contributions of this paper are as follows:

- We propose a metadata extension based on existing research [1]. Our extension ensures privacy-aware data access and enables trust between actors by recording each actor's actions on data.
- We propose a pointer system to manage the history of values that are stored in the DHT for a single piece of data. It allows the data owner to maintain and access their transactions history in case of any updates in the pre-stored data.
- We propose a traceability algorithm that enables actors to trace their data and verify the product's origin in a decentralized platform.
- We design and evaluate our decentralized framework against linking, eavesdropping, spoofing, and modification attacks.
- We provide a critical comparison of the proposed solution with state-of the-art decentralized solutions to show the research gap.
- We also provide implementation details with security and privacy analysis and performance evaluation of our framework over a wood supply chain scenario to demonstrate its feasibility.

This paper is structured as follows. Section 2 discusses the motivating scenario that highlights the research challenges. Section 3 provides some background knowledge together with an overview of existing decentralized solutions for data storage and their shortcomings. Section 4 provides the detailed discussion of our contribution with proposed algorithms. Section 5 shows the experimental results, analysis, and performance evaluation of our proposed framework. Finally, section 6 summarizes our results and gives guidelines for the future work.

2. Motivating Scenario and Research Problem

In this section, we first explain the wood supply chain scenario that motivates our work. We then describe the research problems that we address in this paper.

2.1. Motivating Scenario

Our scenario takes place in the context of the wood supply chain that motivates the need for decentralized solution and highlights our research problems. The wood supply chain includes the whole process from wood logs, production, transportation, and sell to the end customers. It enables the actors of the wood supply chain to verify the wood origin, transport, processing, and manufacturing. As depicted in Figure 1, we identified six actors that participate in the wood supply chain.

- **Forest manager**
  The forest manager identifies the trees that are good to make furniture (e.g oak) and cuts them into logs.
- **Transporter**
  The transporter loads wood logs from the forest and transports them to the sawmill.
- **Sawmill manager**
  It processes the logs and stores them for a specific time duration.
- **Product assembler**
  It divides logs into pieces for further processing.
- **Product seller owner**
  The product seller owner sells the furniture to the end customer.
- **Customer**
  The customer takes the wooden furniture and confirms the origin of the wood using the proposed traceability algorithm (see Section 4.4).

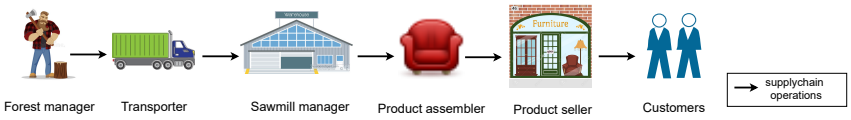

Fig. 1. Wood supply chain and its actors

This scenario highlights the need for decentralized data management, security, privacy, traceability, and data updates [36]. Frauds are common in the wood supply chain, for example, during transportation actors can replace high quality wood with low quality wood [30]. Therefore, all actors participating in the supply chain want to trace products to prevent frauds. To overcome this problem, Radio Frequency Identification (RFID) chips are used with the wood to manage wood traceability [31]. However, existing solutions involve centralized storage to maintain the record of RFID data, thus making single point of failure a major concern [26].

Therefore, blockchain, as a decentralized ledger technology that stores transactions in such a way that all participants can easily access them without requiring any TTP, comes as an interesting technology for solving the single point of failure issue. Each block of the blockchain keeps the hash of its previous block to make it impossible to modify the stored transactions thus ensuring immutability [25,13]. We can say that data cannot be modified once it has been recorded on the blockchain. However, our scenario highlights that actors

of the wood supply chain need to write, read, and update data about their product. As well, they do not want to have their business information publicly available due to security and privacy concerns. There is a need for a solution that overcomes the immutability feature of blockchain, to enable actors to perform update operation on recorded data. At the same time, the designed solution must protect data from unauthorized access and guarantee data access depending on the actor's permission. The identified requirements highlight our motivation to design decentralized data storage and management solution to ensure data access and updates, manage transactions history, security, privacy, data owner's control on their data, and product traceability in a single framework.

2.2. Research Problems

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

- **Data modification**
  Our scenario highlights that actors want to update data at each point of the supply chain (e.g wood location changes). However, it is not possible to update data once it has been recorded on the blockchain, due to its immutability feature. The challenge is to work around the original blockchain design to enable data updates.

- **Data security and privacy**
  Data stored on a blockchain is publicly accessible, highlighting the need for protection from unauthorized access. In other words, different actors shall be granted different access to specific data pieces according to their permissions. The challenge is to provide a decentralized solution that preserves privacy-sensitive data from unauthorized access to ensure data security and privacy.

To address those challenges, we rely on joint usage of blockchain and Distributed Hash Table (DHT), presented in the following to facilitate further understanding of the paper, together with an overview of existing work and its limitations.

3. Background Knowledge and Related Work

In this section, we introduce the basic concepts underpinning blockchain technology and Distributed Hash Table (DHT), we then explain their use in the context of decentralized data storage. We follow with a survey and analysis of existing decentralized data storage solutions. We compare our proposed solution with existing work and summarize the results in Table 1.

3.1. Basic Concepts: Blockchain and DHT

In 2008, blockchain technology [22] was introduced to the world and became popular due to its decentralized feature. The blockchain is a decentralized database that stores all the transactions that take place on the network. All participants on the network have the same copy of the transactions. Before adding each block to the blockchain, miners

accept and verify the transactions using a consensus algorithm such as proof of work. By using proof of work or similar mechanism [15], miners solve very difficult mathematical calculations that should be accepted by other miners on the network [3]. After verifying the correctness of transactions by other miners, a block is appended to the end of the chain [24]. Each block is comprised of a block version, timestamps, consensus signature, parent block hash, and many transactions. The parent block stores the hash of its previous block to form a blockchain that ensures the immutability of the stored data [32]. The hash is a unique value that ensures integrity of the entire blockchain from the initial block (known as genesis block) to the last.

A distributed hash table (DHT) is a decentralized data storage system that stores data as (key, value) pairs over a set of nodes that distribute the storage, possibly with some level of replication. As an example, a well-known DHT implementation is Kademlia [19]. Each node in the DHT maintains the keys it is responsible for and their corresponding values. A key is a unique identifier to its corresponding data value. Each key is generated by applying a hash function to the value. A DHT is based on two main tasks: PUT(key, value) is used to add new data, while GET(key) is used to retrieve the data, that is associated with the given key. A DHT node contains a routing table that maintains the identifier of its neighbor nodes. To find a (key, value) pair, a requesting node contacts the multiple nodes in the network until it reaches the destination node and finds the (key, value) pair. DHT has an advantage in terms of fault-tolerance because (key, value) pairs are replicated on multiple nodes in the network, that ensures data availability [43]. In addition, and as opposed to blockchain, it is scalable enough to manage large data volumes.

3.2. Blockchain and DHT-based data storage

There is a large amount of literature that combines DHT with blockchain to provide decentralized data storage. A framework to manage personal data is proposed in [43]. The solution stores encrypted data (with shared key) on DHT and its pointer on the blockchain. Both service and user can query the data. However, existing work supports one type of encryption. Most work use a shared symmetric key for data encryption/decryption, as in [43], to query the data. In contrast, our framework provides run-time flexibility, which provides various types of data encryption and decryption during execution depending on users' needs and application requirements. In [43], it is not clearly explained how symmetric keys are protected from unauthorized access and where they are stored. In our work, we encrypt symmetric key with the public key of the data owner, and store it on the DHT together with the data, so that later the data owner can access the data.

In [28], a distributed access control and data management framework is presented. The framework enables secure IoT (Internet of Things) data sharing by combining blockchain with off-chain storage (i.e DHT). Fine-grained access control permissions are stored on the blockchain and are publicly visible, which raises privacy issues. Also, it is not possible to update access control permissions due to public blockchain immutability nature. On the other hand, our proposed framework is flexible to update access control permissions. We also maintain data owner anonymity for sharing data.

In [1], the authors propose a decentralized data storage for PingER (Ping End-to-End Reporting) framework. The proposed framework stores metadata of the daily PingER files on a permissioned blockchain, while the original data is stored off-chain. However, their solution writes monitoring agent name and file locations on the permissioned blockchain,

which is immutable and shared with other participants on the network. In addition, this solution does not record the data modification history in case of any modification in the data. Our framework design relies on the PingER proposal for the metadata structure, however, we integrate privacy and security management to enable role-based access control and privacy protection. Our solution enables data owners to control and access their private data. We also provide a solution to manage the previous versions of data using pointers that enable authorized users to access their transaction history. In addition, our work includes proof of concept prototype as well as empirical performance evaluation, which is not the case in PingER.

Table 1. Our proposed framework comparison with existing work

Solutions	**Decentralization**	**Data Privacy**	**Data Updates**	**Transaction History Support**	**Attacks Prevention**
[43]	Yes	Yes	No	No	No
[28]	Yes	No	No	No	No
[1]	Yes	No	No	No	No
[8]	Yes	No	No	No	No
[16]	Partial	No	No	No	No
[5]	Yes	Yes	No	No	No
[38]	Yes	No	No	No	No
[11]	Partial	Yes	No	No	No
[41]	Partial	Yes	No	No	No
[7]	Partial	Yes	No	No	Yes
[27]	Yes	Yes	No	No	No
Our solution	Yes	Yes	Yes	Yes	Yes

The authors in [8], propose the LightChain framework, which is a permissionless blockchain that operates over participating peers of a skip graph DHT. The proposed framework enables all participating peers to access blocks and transactions by using a skip graph overlay. LightChain allows every peer to join the blockchain without any restrictions. However, blocks and transactions are addressable and accessible to everyone on the network. In contrast to the existing framework, our solution uses Role-based Access Control (RBAC) model that allows only authorized users to access blocks and transactions. We store metadata with a pointer on the blockchain, which enables other actors to keep track of data changes with the help of this metadata.

Table 1 presents a global overview of existing work with respect to the following features: decentralization, data privacy, data updates, transaction history support, and attacks prevention. The table shows that some existing solutions ensure decentralization, data privacy, and attacks prevention [43,7]. However, some solutions did not address data updates and transaction history support [28,1,8,27].

3.3. Other Decentralized Data Storage Solutions

An Ethereum-based blockchain platform is presented in [16]. The proposed solution allows companies partners to share data with each other. Original data are stored on off-

chain storage such as MySQL, while a hash sum of corresponding data is sent to the blockchain. However, MySQL database is not scalable as DHT to manage a large amount of data [12]. In addition, MySQL database becomes a single point of failure. In our solution, we use a DHT to store data as (key, value) pair, which can handle a large amount of data easily. In our framework, any authenticated user can efficiently retrieve the value with the help of a corresponding key. As well, our solution is fully decentralized and eliminates the risk of single point of failure.

In [5], the authors propose a framework called u-share. It is a blockchain-based framework to maintain the owner's data traceability while sharing data with their friends and family. The proposed framework is based on a software client to share the private keys with corresponding circle members, keeps a record of shared keys, and encrypt the data using the circle's public key before to share it. However, sharing private key raises security issues. Additionally, the existing framework relies on one type of encryption method. Compared to the existing u-share framework, our proposed solution allows actors to directly generate their public and private keys at run time and control of their private keys. Our solution allows data owners to directly encrypt, decrypt, and share their data with other actors by using different types of encryption methods.

The authors in [29] present a blockchain-based framework that enables users to share their data with other users. A smart contract is used to store data sharing policies that control users' access to the data, while users' private data is stored on the off-chain storage called multi-chain. However, policies stored on the smart contract are immutable. In contrast, our solution enables data owners to update access control permissions. In addition, we ensure data owner anonymity to share data.

In [38], a decentralized supply chain system to keep track of goods and recipe ingredients is presented. The proposed framework uses a smart contract to handle the exchange of goods on a distributed ledger. The main limitation of this solution is the immutability and availability of data to everyone, which could lead to privacy and data modification concerns. On the other hand, our solution stores encrypted data on DHT to ensures data privacy. In addition, our framework allows actors to update data at each point of the chain.

In [11], a blockchain-based food supply chain traceability through smart contract is presented. The proposed framework uses blockchain to store data hash while corresponding data are stored on IPFS (InterPlanetary File System) off-chain storage. IPFS is a peer-to-peer storage network where data stores on the peers of the network [17]. However, a manufacturer node server is used to handle all modules of the framework, which subjects to a single point of failure. On the other hand, our framework modules are fully decentralized and independent of any central orchestrator. For the sake of simplicity, we use a registry server to connect nodes to each other, however, decentralized discovery protocol can easily be used instead of registry server [9].

In [41], the authors propose a decentralized IoT data sharing solution using IOTA Tangle and IPFS technology. The proposed solution uses centralized data handling unit (such as a local server) to collect and encrypt the data using asymmetric encryption, which becomes a single point of failure. In contrast, our proposed solution manage and store data without any central party. The IPFS is used to upload the encrypted data, while the corresponding hash and metadata are managed on the IOTA Tangle. However, the IPFS network is immutable and stores files and its content permanently [14]. On the other hand, we use DHT to store the data and we extend it to allow data modification at any time.

In contrast, our solution allows going through the history of data values and supports querying it.

The authors in [33] propose a blockchain-based framework that maintains the traceability of the food supply chain. RFID technology is used to automatically identify objects through radio frequency signals. However, blockchain technology is not scalable to store a large amount of data. In contrast to this solution, we propose to only store metadata and pointer on the blockchain, while original data is stored on a DHT, which better supports storing large amounts of data. In addition, our framework supports data mutability, thanks to the DHT, whereas blockchain is immutable and shows more difficulty to handle large amounts of data.

In [4] the authors discuss the distributed cloud storage system called Storj. It is a trust-based storage system between host and customer. In this system, people sell their free storage hardware space and earn money. Customers encrypt (using AES256-CTR) their data before storing it on the network. Storj allows the data owner to control and access their data on the network. However, Storj is very costly and depends on a centralized architecture to conclude storage data and payments [7,10]. In contrast, our solution is fully decentralized architecture and avoids a single point of failure. In addition, Storj uses one type of encryption method to establish trust between customer and host [39]. As compared to this, our solution offers different types of encryption methods and enables trust in the decentralized system instead of participants on the network.

In [27] the authors discuss a decentralized data storage framework that combines Solid Pods with blockchain technology. Solid (Social Linked Data) relies on RDF (Resource Description Framework) and semantic web to manage data. Solid enables people to store their personal data in Pods (Personal online data stores) hosted at the location according to the people's wish. The proposed framework discusses the following two cases to ensure data confidentiality. The first is to store file hash on the blockchain while Solid Pods is used to store the data. Second, they use smart contract to store the data on a Blockchain whereas solid pods are used to store the software wallet (public and private key pair). User can access their data using the software wallet. However, Solid Pods itself does not ensure data verification and trust [6]. In addition, it does not support storing large amounts of data as DHT does [20]. In contrast, our framework allows to manage large amounts of data in a decentralized way due to the use of a DHT. Therefore, our approach to data storage is quite different as we do not adopt a user-based isolated storage but rather a globally decentralized storage that relies on the network peers to ensure security and privacy.

In a summary, most existing data storage solutions are subject to the single point of failure issue, data mutability or adopt different designs. In the following, we detail our framework and proposed algorithms in detail.

4. Contribution

In this paper, we propose a secure and privacy-aware decentralized framework to support data storage, authorized data access, data mutability, management of their update history, and traceability. This section starts with the metadata structure that is immutable record of data operations. Then, it describes the overview of our proposed framework and follows with the detail of its execution or sequence. After that, it details the proposed algorithms.

Each actor of the framework runs the same code that is structured into a set of components as depicted in Figure 3.

4.1. Metadata Structure

In [1], authors write metadata such as names and locations only once a day on the permissioned blockchain, which is immutable and they shared this information with all users on the network. In contrast, we store metadata of each actor's action (such as data write date and time) to maintain the actor's trust. This allows actors to keep track of the data.

We propose a privacy-aware metadata extension discussed in the paper [1], to handle privacy restrictions on the data. Therefore, our framework encrypts the actor's private information (e.g name and location) with encryption mechanisms (illustrated in Algorithm 2), and store this encrypted data on the DHT. Our solution also allows only authorize actors to update the product location in case if wood drives from one place to another place. We use a blockchain to store the metadata and DHT key of this encrypted data. Our proposed metadata structure contains the DHT key, previous pointer, data owner's id, date, time, and RFID_number as shown in Figure 2. The DHT key is a hash pointer that points to the data in the DHT. Previous pointer is a hash key of the previous version of the data, which enables data owners to access their transaction history. In our framework, each actor has unique data owner id which is used to make a data request and identify who is the owner of the corresponding data. Our solution records data and time of each operation (such as data write, read, update, and delete) that is performed on the data. RFID_number is a unique data id of the log, lumber and product which is used to trace the items in the chain.

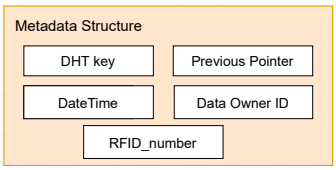

Fig. 2. Metadata structure on the blockchain

4.2. Architecture Overview

Our framework uses RESTful APIs to enable actors to communicate with other actors and support the framework functionalities.

Figure 3 depicts the execution workflow of the proposed framework and its components. In our framework, all actors are running the same `main` program and they call to `registry_server` (`/peers` resource, method `'GET'`) to retrieve the list of available actors (e.g peers) and connect with each other through APIs.

Let us illustrate the operation of our framework with the wood supply chain scenario developed earlier: an actor, for example a forest manager actor, starts the `main` program to store the number of logs and type of wood that he cuts. Then, he will call the `/peers`

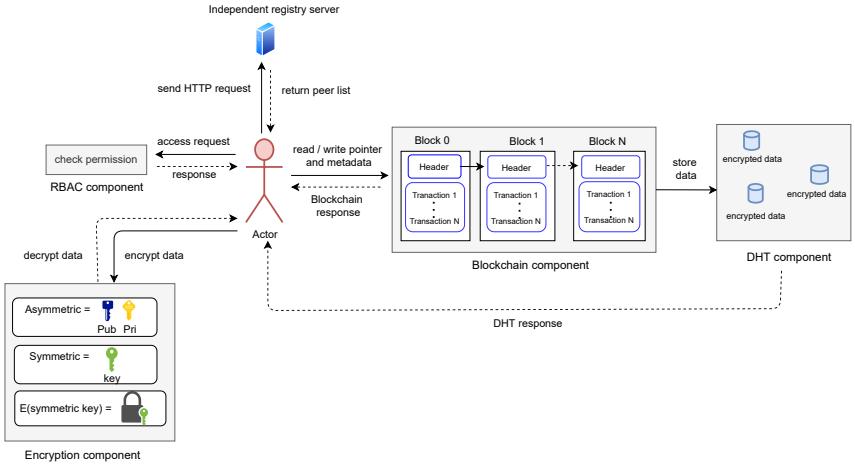

Fig. 3. Overview of the decentralized framework

resource of the `registry_server` with the `'POST'` method to add its public key and Uniform Resource Locator (URL) to the list of connected peers or actors. After that, he will send a `'GET'` request to the `/peers` resource to receive the information of available peers. Then, he will take a copy of the recent 40 transactions of the blockchain using `/chain` resource with a `'GET'` method[3].

In the proposed framework, the `RBAC` component called by the `main` component is responsible for checking the permission of the actor. It allows the only authorized actor to perform operations such as data write, read, delete, and update.

An authorized actor has a choice between multiple types of encryption techniques to secure their data in a decentralized ledger. Our `encryption_component` called by the `main` component generates keys (a public/private key pair, or a symmetric key) based on the encryption method chosen by the authorized actor and encrypts the data accordingly.

We store the encrypted data on the `DHT` component, while DHT key (a hash pointer of the data) and metadata are stored on the `blockchain` component. Later, an authorized actor can access their data using the DHT key stored on the `blockchain` component.

Accordingly, an authorized actor can create a new block using `/chain` resource with the method `'POST'`. To read the data, an actor will call the resource `/chain/<id>` with `'GET'` method. If an actor wants to update some part of the data, then it will call the `/chain/<id>` resource using `'PUT'` method. Similarly, to delete the data, an authorized actor will make a `'DELETE'` request to the `/chain/<id>` resource. An actor can access their public key using the resource `/public_key` with method `'GET'`.

Figure 4 shows the swagger user interface that enables authorized actors to use the proposed APIs discussed above.

[3] Please note that here we avoid downloading the whole blockchain due to performance issues, but only the most recent part, the rest being on-demand. This particular aspect of the work is out of the scope of this paper.

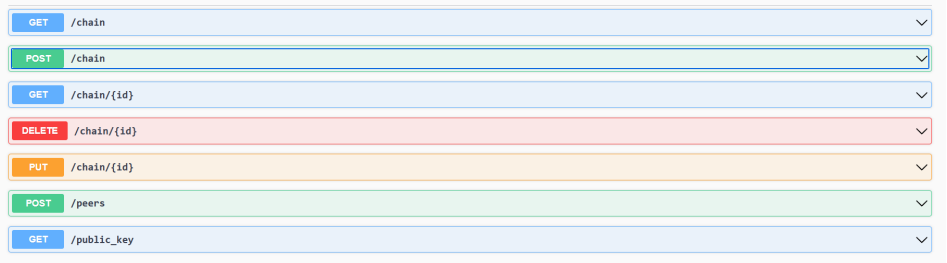

Fig. 4. Overview of the proposed API using Swagger

The overview of each actor's actions (such as write, read, update, and delete) on the data is depicted in figure 5. The data represents in the figure 5 is stored on the DHT component, while corresponding metadata is managed on the blockchain component. Please see the detail of the metadata structure in section 4.1.

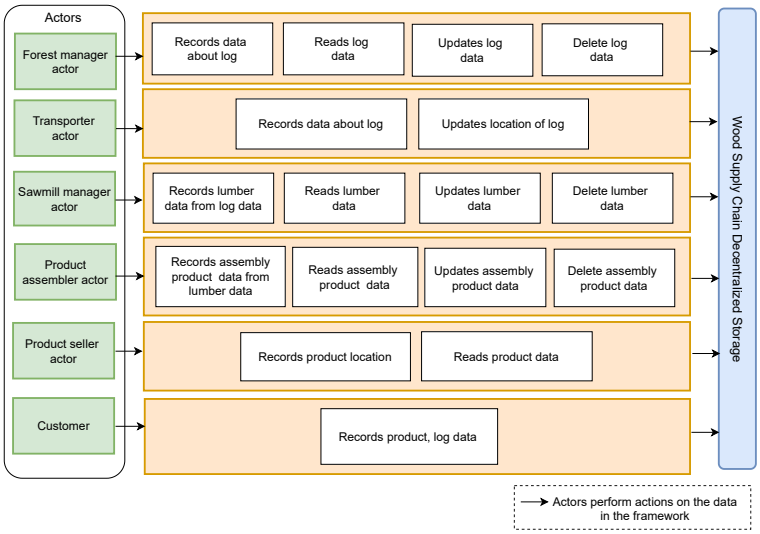

Fig. 5. High-level representation of actors actions on the data

4.3. Interaction via RESTful API

In this section, we detail the possible usage of our framework with a sequence diagram (Figure 6) that illustrates the interaction between an actor (e.g. a forest manager) and the framework using its RESTful API. We assume that every actor is already registered on the framework. An actor makes a `'POST'` request to the `/chain` resource to write log data in the framework. Our solution assigns a unique data id (RFID_number) to the log that

enables authorized actors to trace the log in the chain. In the case of a successful response (HTTP code 201), it returns the links including the id in the response. Our framework stores the DHT key of this generated data in the metadata. Therefore, this DHT key points to the location of the log data on the DHT. The actor can use these links to perform further actions on the log data by sending another HTTP request as described in the links. To read the data, an actor would use the GET link that would call the `/chain/<id>` resource with method `'GET'` to retrieve the representation of the log data. In the case of a successful response (HTTP code 200), our framework returns the representation of the log data. In case an actor wants to update their data, then they use the PUT link that makes a `'PUT'` request to the `(/chain/<id>` resource). It will then write the new data against the same id. Then, a new metadata structure is created on the blockchain, and it contains the new DHT key of this updated data and the previous pointer of the old version of the data. Similarly, to delete the data, an actor may follow the DELETE link `(/chain/<id>` resource, method `'DELETE'`). Our framework allows the authorized actor to delete specific data based on the id. After verifying the permission of the actor, it will delete the data. In this case, a new metadata structure is created on the blockchain that has a new DHT key with a NULL value.

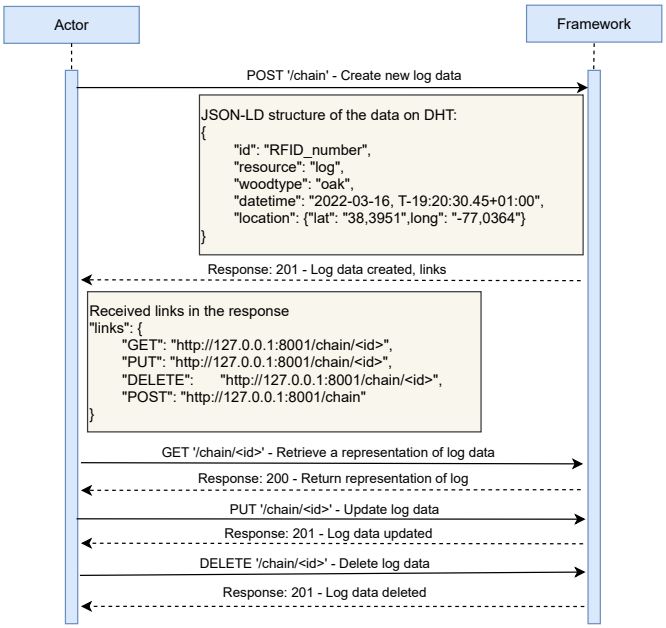

Fig. 6. Sequence diagram of possible actor interactions with the framework

4.4. Registration and Data Management

This section presents the proposed algorithms that support our solution including actor registration using designated REST APIs, data management on the decentralized storage and on the blockchain, and traceability algorithm to keep track of the data history.

– **Actor Registration**
Algorithm 1 describes the actor's connection or registration procedure to the proposed framework using our RESTful APIs. Once actor would successfully connect to the framework then they can perform different actions on the data such as write, read, update etc, and actors can also connect to other actors through HTTP requests. Each new actor needs to connect to the framework once to perform actions.
Firstly, the actor calls the `/peers` resource with `'GET'` method to receive the available peer list (`pl`). After that, it calls the `/peers` resource `'POST'` method to add its public key to the list of available peers and registers to the registry server. Then it sends a request to other peers to acknowledge the connected peer (`/peers` resource, `'POST'` method). If the current actor is already in the list then it will be disconnected or removed from the peer list using the `/peers` resource with `'DELETE'` method. Then it sends a request to other available peers to acknowledge the disconnected peer.

Algorithm 1 Actor registration algorithm

Input: ca: current actor
Output: pl: peer list, boolean value
▷ GET: HTTP verb GET request (constant)
▷ POST: HTTP verb POST request (constant)
▷ pe: endpoint of the peer (constant)
▷ Req.Method: identify request type (variable)
▷ p: peer in loop (variable)

```
1:  if Req.Method == GET then
2:      return pl
3:  end if
4:  if Req.Method == POST then
5:      pl.Append(ca)
6:      for each p ∈ pl do
7:          RequestsPost(p(pe), ca)
8:      end for
9:      return true
10: end if
11: if Req.Method == DELETE then
12:     pl.Remove(ca)
13:     for each p ∈ pl do
14:         RequestsDelete(p(pe), ca)
15:     end for
16:     return true
17: end if
```

– **Data Management on the DHT**
The process to write or store the data including metadata and corresponding DHT key (a hash pointer of the encrypted data) is shown in Algorithm 2. Our proposed framework combine blockchain with a DHT in a way that allows authorized actors to write and update the data about their activities. For instance, if an actor has a role "data owner" and wants to store their log data such as:

```
{
    "id": "RFID_number",
    "resource": "log",
    "woodtype": "maple",
    "datetime": "2022-06-01, T-11:16:25.45+01:00",
    "location":
    {
        "lat": "14,2472",
        "long": "-43,2135"
    }
}
```

Then, `Authenticate(actor, role)` and `CheckPermission(actor, role, v)` verify that the current actor has the right permissions to store their data or not. The `CheckPermission` checks if the current actor has a role 'forest manager' then he is allowed to write, read, update, and delete their data in the decentralized platform.
After verifying the permission of the current actor, our framework provides different encryption methods (em) to encrypt the data before storing it on the decentralized ledger that ensures data security. An authorized actor is allowed to choose between `asymmetric` em and `symmetric` em. Asymmetric encryption is based on separate public and private keys. A public key is used to encrypt the data, while a corresponding private key is used to decrypt the data. In our motivating scenario, if a forest manager actor chooses `asymmetric` em then data will be encrypted with the data owner's public key, so later he can only access his data using his private key.
The authorized actor also has an option to choose `symmetric` em to encrypt the data, if he wants to enable other actors to read their data. A symmetric key is based on a single key to encrypt and decrypt the data. If the data owner chooses `symmetric` em, then our framework encrypts the data with the `symmetric` key and then this `symmetric` key will be encrypted with a data owner public key to protect the key from unauthorized actors.
Upon data read request, the data owner would encrypt this symmetric key using the requester's the public key to enable the authorized actors to read the data.
We store this encrypted symmetric key (ek) and encrypted data (ed) on the DHT. The ed stores on the DHT contains resource, woodtype, location (such as latitude and longitude) that shows the geographical location of the resource in the wood supply chain. Then, the function `FindLastTransaction` takes the data id such as ($rfid\_number$) as input and returns previous pointer (pp) if it exists otherwise it returns 0. We store the metadata on the blockchain. The metadata includes DHT key (dk), pp, datetime, data owner id ($doid$), and data id ($rfid\_number$).

Algorithm 2 Algorithm for the data write operation

Input: d: data, actor: current actor, role: role of the actor, v: HTTP verb POST, PUT, em: encryption method, pp: pointer of previous transaction when data is updated

Output: ed: encrypted data, encrypted symmetric key (ek) ▷ pk: public key of data owner (constant)
▷ doid: id of the data owner (constant)
▷ sk: symmetric key (variable)
▷ dht: variable to store the ed and ek
▷ dk: dht key points to the data in dht (variable)
▷ rfid_number: data id (variable)
▷ datetime: timestamp (variable)
▷ pp: previous pointer (variable)

```
1:  if Authenticate(actor, role) then
2:      if CheckPermission(actor, role, v) then
3:          if em == true  then                    ▷ if true we use asymmetric encryption)
4:              ed← Encrypt(d, pk)
5:          else                                   ▷ if false we use symmetric encryption)
6:              encrypd← Encrypt(d, sk)
7:              ek← Encrypt(sk, pk)
8:              ed← encrypd, ek
9:          end if
10:         dk← Digest(ed)
11:         dht ← SetValue(ed)
12:         pp ← FindLastTransaction(rfid-number)
13:         AddTransaction(dk, pp, datetime, doid, rfid_number)
14:     end if
15: end if
```

– **Data Management on the Blockchain**

As an extension to the work in [1], we propose a metadata structure that manages the pointer and connects the different values attached to a specific piece of data to maintain its history. For example, a forest manager actor, as a data owner, would write a log information such as:

```
{
    "id": "RFID_number",
    "resource": "log",
    "woodtype": "maple",
    "datetime": "2022-05-03, T-10:12:21.45+01:00",
    "location":
    {
        "lat": "13,2351",
        "long": "-15,5142"
    }
}
```

In this case, the proposed solution stores the DHT key as a new pointer of the log data in the metadata. Later, the data owner can access the data using a data id (RFID_number

of the corresponding data). An actor can update some parts of the data against the same data id such as:

```
{
    "id": "RFID_number",
    "resource": "log",
    "woodtype": "maple",
    "datetime": "2022-08-06, T-14:16:23.45+01:00",
    "location":
    {
        "lat": "11,2256",
        "long": "-21,1525"
    }
}
```

Our solution allows the data owner to perform different operations (such as update, read and delete) on their data for the specific RFID_number. In case of data update, new metadata will be generated on the blockchain that includes a new DHT key of the updated data and the previous pointer that refers to the previous version of the data that is stored on the DHT (illustrated in Algorithm 2). Therefore, the DHT key of the previous version of the transaction becomes the previous pointer which is stored in the new metadata. The proposed metadata structure also stores the datetime of the updated data. This way if the data owner wants to see their transactions history, then the function `FindLastTransaction(did)` returns the recent version of the transaction against this data id as RFID_number containing the DHT key of new data and previous pointer of the updated data. This way an actor can access their update history. To read the data, an authorized actor can decrypt and access their data in the decentralized platform. In case, if data is encrypted with the data owner's public key then a data owner can use their private key to decrypt and read the data. If the data is encrypted with a symmetric key then the authorized actor first decrypts the symmetric key using their private key and then this decrypted symmetric key will be used to access the data that is stored on the DHT. Similarly, an authorized actor can delete their data against a specific RFID_number, then a new transaction is created on the blockchain that includes a new metadata structure. This metadata includes a new DHT key with a NULL value.

– **Traceability**
We propose an solution that maintains data id references to ensure traceability. It enables actors to verify the origin of the final product in the chain. Our solution assigns a unique data id (such as RFID_number) to the log, lumber, and product. We assume that, RFID chips are inserted into the logs and then into the lumbers and final products. The following code shows the log data in JSON format such as.

```
{
    "id": "RFID_number",
    "resource": "log",
    "woodtype": "maple",
    "datetime": "2022-05-10, T-13:10:20.45+01:00",
    "location":
    {
```

```
        "lat": "25,1324",
        "long": "-45,1326"
    }
}
```

A log produces different pieces of lumbers and each lumber has unique id as RFID_number. The following code shows the lumber data.

```
{
    "id": "RFID_number",
    "resource": "lumber",
    "datetime": "2022-05-13, T-14:12:23.45+01:00",
    "location":
    {
        "lat": "12,2425",
        "long": "-23,1526"
    },
    "log":
    {
          "id": "RFID_number"
    }
}
```

The data described above contains a reference id (RFID_number) of the log that was turned into lumbers. The different pieces of lumbers participate to build a final product such as wooden furniture. The following is a JSON representation of product data.

```
{
    "id": "RFID_number",
    "resource": "product",
    "datetime": "2022-06-02, T-16:14:26.45+01:00",
    "location":
    {
        "lat": "52,5323",
        "long": "-24,3316"
    },
    "lumber":
    {
        "id": "RFID_number"
    }
}
```

The product data represented above contains an id reference of lumber that was used to build it. This way an authorized actor can verify the origin of the wooden product and can identify where it comes from. The process to trace the data and verifies the product origin in the wood supply chain is shown in Algorithm 3. For instance, a customer buys a wooden product such as a bed and he wants to trace this product. Then, he can use the product id as a data id (such as RFID_number) to keep track of their origin. The proposed algorithm enables actors to trace the product's origin using the data id's references discussed above.

In Algorithm 3, the `did` is an RFID_number of the item in the wood supply chain, and data (e.g location) of the item changes for the same `did`. Therefore, we can have multiple transactions on the blockchain against this `did`. Whenever the location of the item would change then new metadata of the same `did` will be recorded on the blockchain, and the corresponding data is stored to the DHT. The `FindLastTransaction` function returns the last or recent transaction `t` of this `did`, which is a RFID_number. For instance, if we have `did` of the log then it finds the last transaction of this log.
This transaction `t` has the metadata that contains DHT key that points to the data recorded on the DHT. The function `CheckPermission` verifies if the current data requester is authorize to read the data or not depending on their role and HTTP verbs permission `'GET'`. Then, the function `GetReferences` has the `t` as input and takes the `did` of the items. After that, it gets the previous references of this `did`. For example, if we have a input `did` as product id then it finds the previous references such as RFID_number of the lumbers. Then, it checks items (e.g lumbers) in the list and add items (e.g lumbers references) in the output list (`o`). Then, the `Traceability` function takes `i` such as lumber as input and call recursively to find out the log and add them in the list `o`. In case the list `o` is empty it is returned anyway, and it means that the log does not contain any previous reference.

Algorithm 3 Traceability algorithm

Input: did: data id (DHT key)
actor: requester actor, role: requester role, v: HTTP verb GET
Output: o: DHT keys of tracked items
▷ l: items list (variable)

```
 1: l ← ∅
 2: t ← FindLastTransaction(did)
 3: if CheckPermission (actor, role, v) then
 4:     l ← GetReferences (t)
 5:     if l ≠ ∅ then
 6:         o ← ∅
 7:         for each i ∈ l do
 8:             o.Append(i)
 9:             o.Append(Traceability(i))
10:         end for
11:         return o
12:     end if
13: end if
14: return ∅
```

5. Results and Evaluation

This section presents the results and performance evaluation of the proposed decentralized data storage framework. The evaluation framework is discussed in Section 5.1. The

security and privacy analysis are presented in Section 5.2. Section 5.3 discusses the performance evaluation of our proposed framework.

5.1. Evaluation Framework

To implement and evaluate the performance of our framework, we used Python 3.7.0. We used a Python library[4] to implement a blockchain to store the DHT key and metadata. We implemented a DHT using the Kademlia library[5], which allows to store and get data linked with a given key on the peer-to-peer network. We used the cryptography RSA library to generate private/public keys and encrypt/decrypt the data. We conducted experiments and evaluated our framework on a 64-bit Windows operating system, Core i7 1.80 GHz processor, and 16 GB RAM.

5.2. Privacy and Security Analysis

The proposed solution supports data privacy and enables data owners to own and control their data in a decentralized platform. Our check permission method prevents unauthorized actors to perform operations on data such as data write, read, update, and delete. In addition, to protect privacy-sensitive data from unauthorized access, our framework provides multi layers of encryption to ensure privacy and security. The data stored on the DHT are encrypted before uploading. Even if an unauthorized actor gains access to the DHT nodes then they can only see the cipher texts and cannot achieve any information about the data. Moreover, in our solution, we used blockchain and DHT because of their decentralized and distributed design. This can solve the single-point failure issue, and ensures data replication and availability. We analyzed and evaluated the security of our framework under the following threats:

- **Linking attack**
  A linking attack happens when the attacker tries to link various transactions or data with the corresponding public key. In our design, we use different encryption mechanisms to encrypt the data, such as the data owner's public key, symmetric key, and requester's public key. We generate public, private, and symmetric keys at run-time according to the encryption method chosen by the actor. To secure the symmetric key from unauthorized access, our framework encrypts the symmetric key with the data owner's public key and stores it on the DHT. This way only the authorized user is allowed to use this symmetric key to decrypt and access their data. For this reason, an attacker cannot link different transactions to the same public key, because our solution encrypts the data using different encryption mechanisms and public keys.

- **Eavesdropping attack**
  In an eavesdropping attack, an attacker tries to listen to privacy-sensitive information in the network. To protect against this attack, we encrypt privacy-sensitive data with the requester's public key upon data read request. This way only authorized actors can access and read the data using their private key.

[4] `https://github.com/satwikkansal/python_blockchain_app/tree/ibm_blockchain_post`

[5] `https://github.com/bmuller/kademlia`

– **Spoofing attack**

A spoofing attack happens when a malicious actor uses the ID of another actor and tries to access the data. In our framework, a malicious user cannot spoof the ID of another actor because they could not spoof its private key. In our solution, each actor has a private key that is kept secret and not shared with others.

– **Modification attack**

A modification attack occurs when an attacker tries to change the data content. In our framework design, we allow data owners to encrypt the data using their public key and store the corresponding pointer on the blockchain. Our proposed metadata design keeps the track of data entry date and time to recognize the changes in the data. An attacker cannot modify the data because data can only be decrypted with a data owner's private key that is kept secret by the data owner.

5.3. Performance Evaluation

We evaluated the results according to time consumption and scalability with respect to the number of peers. We computed the time consumption of the proposed solution according to the following parameters: actor's check permission, data encryption/decryption using asymmetric or symmetric techniques, DHT access, and blockchain access. We observed time consumption while performing data write, update, read, delete, and traceability operations. Figure 7 and 8 show the time consumption of the different parts of our solution, respectively using symmetric encryption and asymmetric encryption.

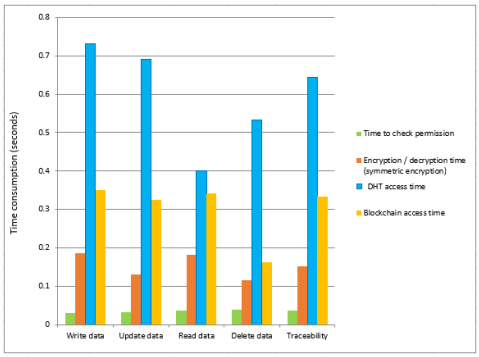

Fig. 7. Time consumption using symmetric encryption

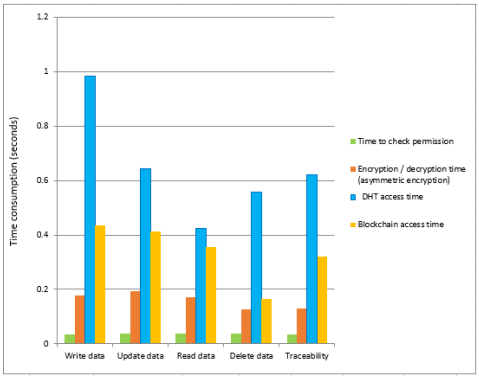

Fig. 8. Time consumption using asymmetric encryption

The general trend of our measurements shows that DHT access takes most of the time needed, followed by blockchain access, encryption/decryption and then permission check, which makes sense since the DHT deals with data storage and is therefore I/O-bound. We believe however that some low-level optimization is performed at this stage (see the scalability tests and discussion).

In general, the usage of symmetric or asymmetric encryption does not impact the solution much, except a slight increase of time consumption if asymmetric encryption is used. We make sense of these results by acknowledging the higher number of keys and costly computation that are needed when using asymmetric cryptography.

The most time-consuming operation is, without surprise, the write operation, since it requires the most from the system. Second comes the update operation which is similar to a write except it is already related to an existing piece of data. Third comes traceability, which does not modify the existing data but requires following the history of different pieces of data. Finally, the delete operation is less costly, and the read operation only consists in resolving the DHT pointer and if granted, fetching the data.

Moreover, we tested the scalability of our solution with a growing number of actors 1, 100, 200, 300, 400 and obtained a reasonable performance with 400 actors (please note that increasing the number of actors to more than 400 would lead to additional synchronization problem, which would slow down the speed and performance. These problems are out of the scope of this paper.) The HTTP requests will be only partially processed in parallel, since they share the CPU time, and we tested our prototype with a quad-core CPU. In our solution, actors are the same as blockchain nodes and DHT nodes. We tested our solution with a number of 400 actors which are considered as 400 nodes.

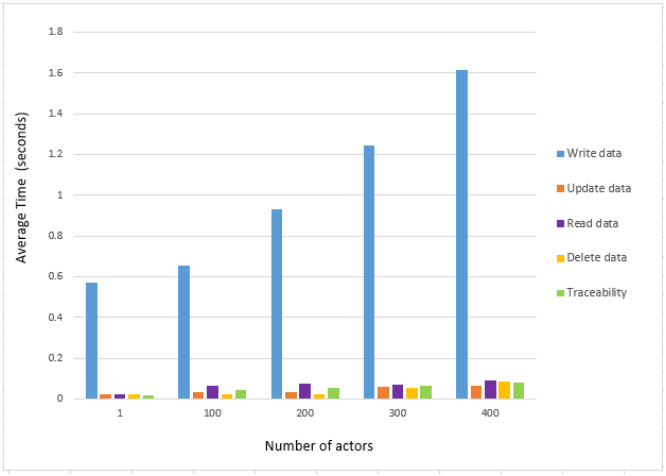

Fig. 9. Average time consumption under different number of actors

We calculated the average time consumption of our prototype with an increasing number of actors. The actor registration operation is performed only once for 1, 100, 200, 300, and 400 actor and the time costs is 0,0034 seconds, 0,0039 seconds, 0,0041 seconds, 0,0046 seconds, and 0,0049 seconds respectively. Therefore, we tested our prototype 100 times for all operations such as write data, update data, read data, delete data, and traceability. After that, we calculated the average time, Standard Deviation (SD), minimum (min), and maximum (max) values in seconds. Figure 9 depicts the average time consumption between a different number of actors, and detailed results statistics are summarized in Table 2..

As we can see from Figure 9 and Table 2, for the case of 1 actor, write data gives an average of 0,5712 seconds which is less than the average time of data write for 100, 200, 300, and 400 actors. The update data has an SD of 0,0211 seconds which is close to the SD of update data for the case of 200 actors. The data read provides an max value of 0,0456 seconds which is less than the may value of read data for the case of 100, 200, 300, and 400 actors. The delete data takes an average time of 0,0214 seconds which is close to the average time of delete data for 100 and 200 actors. The traceability data operation has a min value of 0,0112 seconds and a max value of 0,0312 seconds.

Table 2. Detailed results under different number of actors

Number of actors	Data operations	Average Time	St Deviation	Minimum	Maximum
1	Write data	**0,5712**	0,4321	0,4635	0,6564
	Update data	0,0224	**0,0211**	0,0221	0,0412
	Read data	0,0254	0,0113	0,0124	**0,0456**
	Delete data	**0,0214**	0,0212	0,0213	0,0434
	Traceability	0,0201	0,0101	**0,0112**	**0,0312**
100	Write data	**0,6552**	0,5352	0,5432	0,7681
	Update data	0,0346	**0,0321**	0,0334	0,0571
	Read data	0,0632	**0,0512**	0,0542	0,0724
	Delete data	0,0233	0,0221	**0,0223**	0,0342
	Traceability	**0,0464**	0,0413	0,0421	0,0641
200	Write data	**0,9325**	0,6215	0,6316	1,8622
	Update data	0,0356	**0,0241**	0,0256	0,0392
	Read data	**0,0738**	0,0635	0,0641	0,0956
	Delete data	0,0215	0,0153	**0,0171**	0,0516
	Traceability	0,0521	0,0439	0,0472	**0,0695**
300	Write data	**1,2455**	0,7529	0,7924	1,9372
	Update data	0,0573	**0,0543**	0,0561	0,0635
	Read data	0,0713	0,0537	0,0655	**0,0836**
	Delete data	0,0531	**0,0457**	0,0461	0,0734
	Traceability	**0,0636**	0,0531	0,0571	0,0913
400	Write data	**1,6121**	1,3163	1,3223	2,4692
	Update data	0,0626	**0,0551**	0,0569	0,0931
	Read data	0,0911	0,0815	**0,0857**	0,2419
	Delete data	**0,0882**	0,0731	0,0765	1,4271
	Traceability	0,0791	0,0682	0,0693	**0,0975**

For the case of 100 actors, the write operation gives an average of 0,6552 seconds which is slightly higher than the average time to write data with 1 actor. The update operation gives an SD time of 0,0321 seconds which is slightly higher than the SD to update data with 1 actor and 200 actors. The read operation has a SD of 0,0512 seconds which is slightly close to the SD of read data for 300 actors. The delete operation gives a min value of 0,0223 seconds which is close to the min value for 1 actor. The traceability algorithm has an average time of 0,0464 seconds which is less as compared to the average time for 200, 300, and 400 actors.

Similarly, with the number of 200 actors, the average time to write data is 0,9325 seconds which is slightly higher than the average time to write data for the number of 1 and 100 actors. The update operation provides an SD of 0,0241 seconds which is less than the SD of update data for the case of 100 actors. The read operation gives an average time of 0,0738 seconds which is slightly close to the average time to read data for the case of 300 actors. The delete operation has a min value of 0,0171 seconds which is less than the min value for 1, 100, 300, and 400 actors. The traceability data operation gives a max value of 0,0695 seconds which does not show much difference from the max value of 100 actors.

For the case of 300 actors, the write data operation gives an average of 1,2455 seconds which is slightly higher as compared to the average time to write data for 200 actors. The

update gives an SD value of 0,0543 seconds which is close to the SD for the number of 400 actors. The read data operation gives a max value of 0,0836 seconds which is less as compared to the max value to read data for the number of 200 and 400 actors. The delete operation provides an SD of 0,0457 seconds which is less than the SD for 400 actors. The traceability takes an average time of 0,0636 seconds which is higher than the average time for 1, 100, and 200 actors.

For the number of 400 actors, the average time to write data is 1,6121 seconds which is higher than the average time to write data for 1, 100, 200, and 300 actors. The update data operation gives an SD of 0,0551 seconds which is close to the SD value of update data for 300 actors. The read data provides a min value of 0,0857 seconds and a max value of 0,2419 seconds. The average time to delete data operation is slightly higher than the average time to update operation for 1, 100, 200, and 300 actors. The traceability provides a max value of 0,0975 seconds which is close to the max value for 300 actors.

We interpret the reasonable increase in time consumption despite the large increase in the number of actors as a consequence of the efficiency of DHT access, which is known to be logarithmic, combined with a number of low-level optimization from the Python language, together with operating system and hardware optimization mechanisms related to data management and process execution.

Overall, our experimental results demonstrates that the proposed solution is scalable and able to manage many actors at the same time. The results show that each operation take average time less than 1 minute, while increasing the number of actors, therefore, we can conclude that our solution is acceptable for the end user.

6. Conclusion

In this paper, we present a decentralized data storage and access framework that ensures data security, privacy, and mutability in wood supply chain scenario. The proposed framework integrates blockchain technology with DHT, a role-based access control model, and different types of encryption techniques. Our solution allows authorized actors to write, read, delete, update their data and manage transaction history on a decentralized system. The proposed traceability algorithm enables authorized actors to trace the product data in a decentralized ledger. We provided a critical comparative analysis of our work with existing solutions to show the research gap. The main limitations of existing solutions are a single point of failure, data mutability, and public availability of the data.

Our prototype design is flexible to expand and can be easily reused for different application domains such as medicine, agriculture, etc. We discussed the security and privacy analysis of our proposed solution and evaluate its performance in terms of time cost and scalability. The experimental results show that the proposed solution is scalable, secure, and achieves an acceptable time cost.

In future work, we plan to test our framework with different real-life use-cases and enhance data access with semantic annotation to identify data concepts that are stored and in turn exploit this information to drive the RBAC model. We believe the richness of description logic can contribute to better fine-grained access control and facilitate data management. Another step forward relates to the possibility to adapt semantically annotated data to specific local interpretation depending on the context of the qeury issuer - for example, converting data units between countries.

Acknowledgments. 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. Ali, S., Wang, G., White, B., Cottrell, R.L.: 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). pp. 1303–1308. IEEE (2018)
2. Aslam, S., Mrissa, M.: A restful privacy-aware and mutable decentralized ledger. In: European Conference on Advances in Databases and Information Systems. pp. 193–204. Springer (2021)
3. Aslam, S., Tošić, A., Mrissa, M.: Secure and privacy-aware blockchain design: Requirements, challenges and solutions. Journal of Cybersecurity and Privacy 1(1), 164–194 (2021)
4. Benisi, N.Z., Aminian, M., Javadi, B.: Blockchain-based decentralized storage networks: A survey. Journal of Network and Computer Applications 162, 102656 (2020)
5. Chakravorty, A., Rong, C.: Ushare: user controlled social media based on blockchain. In: Proceedings of the 11th international conference on ubiquitous information management and communication. pp. 1–6 (2017)
6. Domingue, J., Third, A., Ramachandran, M.: The fair trade framework for assessing decentralised data solutions. In: Companion Proceedings of The 2019 World Wide Web Conference. pp. 866–882 (2019)
7. de Figueiredo, S., Madhusudan, A., Reniers, V., Nikova, S., Preneel, B.: Exploring the storj network: a security analysis. In: Proceedings of the 36th Annual ACM Symposium on Applied Computing. pp. 257–264 (2021)
8. Hassanzadeh-Nazarabadi, Y., Küpçü, A., Özkasap, Ö.: Lightchain: A dht-based blockchain for resource constrained environments. arXiv preprint arXiv:1904.00375 (2019)
9. He, Q., Yan, J., Yang, Y., Kowalczyk, R., Jin, H.: A decentralized service discovery approach on peer-to-peer networks. IEEE Transactions on Services Computing 6(1), 64–75 (2011)
10. Hei, Y., Liu, Y., Li, D., Liu, J., Wu, Q.: Themis: An accountable blockchain-based p2p cloud storage scheme. Peer-to-Peer Networking and Applications 14(1), 225–239 (2021)
11. Huang, H., Zhou, X., Liu, J.: Food supply chain traceability scheme based on blockchain and epc technology. In: International Conference on Smart Blockchain. pp. 32–42. Springer (2019)
12. Khamphakdee, N., Benjamas, N., Saiyod, S.: 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). pp. 454–459. IEEE (2016)
13. Kumar, M.V., Iyengar, N.: A framework for blockchain technology in rice supply chain management. Adv. Sci. Technol. Lett 146, 125–130 (2017)
14. Legault, M.: A practitioner's view on distributed storage systems: Overview, challenges and potential solutions. Technology Innovation Management Review 11(6), 32–41 (2021)
15. Li, W., Andreina, S., Bohli, J.M., Karame, G.: Securing proof-of-stake blockchain protocols. In: Data Privacy Management, Cryptocurrencies and Blockchain Technology, pp. 297–315. Springer (2017)
16. Longo, F., Nicoletti, L., Padovano, A., d'Atri, G., Forte, M.: Blockchain-enabled supply chain: An experimental study. Computers & Industrial Engineering 136, 57–69 (2019)
17. Lykousas, N., Koutsokostas, V., Casino, F., Patsakis, C.: The cynicism of modern cybercrime: Automating the analysis of surface web marketplaces. arXiv preprint arXiv:2105.11805 (2021)

18. Marr, B.: How much data do we create every day? the mind-blowing stats everyone should read. forbes. may, 21 2018 (2018)
19. Maymounkov, P., Mazieres, D.: Kademlia: A peer-to-peer information system based on the xor metric. In: International Workshop on Peer-to-Peer Systems. pp. 53–65. Springer (2002)
20. Mikroyannidis, A., Third, A., Domingue, J.: A case study on the decentralisation of life-long learning using blockchain technology. Journal of Interactive Media in Education 2020(1) (2020)
21. Moser, M.: Anonymity of bitcoin transactions. In: Münster Bitcoin Conference (MBC), Münster, Germany (July 2013)
22. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. Decentralized Business Review p. 21260 (2008)
23. Nakamoto, S., Bitcoin, A.: A peer-to-peer electronic cash system. Bitcoin.–URL: https://bitcoin. org/bitcoin. pdf 4 (2008)
24. Ølnes, S., Ubacht, J., Janssen, M.: Blockchain in government: Benefits and implications of distributed ledger technology for information sharing (2017)
25. Pazaitis, A., De Filippi, P., Kostakis, V.: Blockchain and value systems in the sharing economy: The illustrative case of backfeed. Technological Forecasting and Social Change 125, 105–115 (2017)
26. Podduturi, P.R., Maco, T., Ahmadi, P., Islam, K.: Rfid implementation in supply chain management. International Journal of Interdisciplinary Telecommunications and Networking (IJITN) 12(2), 34–45 (2020)
27. Ramachandran, M., Chowdhury, N., Third, A., Domingue, J., Quick, K., Bachler, M.: Towards complete decentralised verification of data with confidentiality: Different ways to connect solid pods and blockchain. In: Companion Proceedings of the Web Conference 2020. pp. 645–649 (2020)
28. Shafagh, H., Burkhalter, L., Hithnawi, A., Duquennoy, S.: Towards blockchain-based auditable storage and sharing of iot data. In: Proceedings of the 2017 on Cloud Computing Security Workshop. pp. 45–50 (2017)
29. Shrestha, A.K., Vassileva, J., Deters, R.: A blockchain platform for user data sharing ensuring user control and incentives. Frontiers in Blockchain 3, 48 (2020)
30. da Silva, D.L., Corrêa, P.L.P., Najm, L.H.: Requirements analysis for a traceability system for management wood supply chain on amazon forest. In: 2010 Fifth International Conference on Digital Information Management (ICDIM). pp. 87–94. IEEE (2010)
31. Sirkka, A.: Modelling traceability in the forestry wood supply chain. In: 2008 IEEE 24th International Conference on Data Engineering Workshop. pp. 104–105. IEEE (2008)
32. Swan, M.: Blockchain thinking: The brain as a decentralized autonomous corporation [commentary]. IEEE Technology and Society Magazine 34(4), 41–52 (2015)
33. Tian, F.: 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). pp. 1–6. IEEE (2016)
34. Toyoda, K., Mathiopoulos, P.T., Sasase, I., Ohtsuki, T.: A novel blockchain-based product ownership management system (poms) for anti-counterfeits in the post supply chain. IEEE access 5, 17465–17477 (2017)
35. Tschorsch, F., Scheuermann, B.: Bitcoin and beyond: A technical survey on decentralized digital currencies. IEEE Communications Surveys & Tutorials 18(3), 2084–2123 (2016)
36. Tzoulis, I., Andreopoulou, Z.: Emerging traceability technologies as a tool for quality wood trade. Procedia Technology 8, 606–611 (2013)
37. Voronchenko, K.: Do you need a blockchain? Supervised by Ivo Kubjas 22 (2017)
38. Westerkamp, M., Victor, F., Küpper, A.: 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). pp. 1595–1602. IEEE (2018)
39. Wilkinson, S., Boshevski, T., Brandoff, J., Buterin, V.: Storj a peer-to-peer cloud storage network. https://www.storj.io/storj2014.pdf (2014)
40. Xu, L., Shah, N., Chen, L., Diallo, N., Gao, Z., Lu, Y., Shi, W.: Enabling the sharing economy: Privacy respecting contract based on public blockchain. In: Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts. pp. 15–21 (2017)
41. Zheng, X., Lu, J., Sun, S., Kiritsis, D.: Decentralized industrial iot data management based on blockchain and ipfs. In: IFIP International Conference on Advances in Production Management Systems. pp. 222–229. Springer (2020)
42. Zheng, Z., Xie, S., Dai, H.N., Chen, X., Wang, H.: Blockchain challenges and opportunities: A survey. International Journal of Web and Grid Services 14(4), 352–375 (2018)
43. Zyskind, G., Nathan, O., et al.: Decentralizing privacy: Using blockchain to protect personal data. In: 2015 IEEE Security and Privacy Workshops. pp. 180–184. IEEE (2015)

Sidra Aslam has completed her Ph.D. in Computer Science from the University of Primorska, Slovenia in June 2022. She published research papers in scholarly journals and conference proceedings. She worked 3 years as an assistant researcher at the international company InnoRenew CoE, Izola, Slovenia, where she was involved in different European industrial projects. She worked as a teaching assistant and taught a master course at the University of Primorska, Koper, Slovenia. In 2021, she was awarded grant for Short Term Scientific Mission (STSM) and worked with the European company 'Advanced Building and Urban Design (ABUD)', Budapest, Hungary. In 2015, she received the best research paper award at the National Software Engineering Conference (NSEC), IEEE in Pakistan.

Michael Mrissa received his PhD in computer science from the University of Lyon, France, in 2007. His main areas of research are related to distributed systems and include service-oriented computing, semantic web, privacy, web of things. He has authored 80+ peer-reviewed publications in international conferences and journals, and he has been involved in several European, French and Slovenian research projects. He is currently researcher in the ICT research group of the InnoRenew CoE. He also holds a full professor position at the Faculty of Mathematics, Natural sciences and Information Technologies of the University of Primorska.

Received: January 10, 2022; Accepted: November 14, 2022.