

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

2 Sidra Aslam^{1,2,3} and Michael Mrissa^{1,2}

3¹ InnoRenew CoE, Livade 6, 6310 Izola, Slovenia
4 sidra.ssc@stmu.edu.pk

5² University of Primorska, Faculty of Mathematics, Natural Sciences
6 and Information Technology, Glagoljaška ulica 8, 6000 Koper, Slovenia

7³ Shifa Tameer-e-Millat University, Department of Computing, Islamabad, Pakistan
8 michael.mrissa@innorennew.eu

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

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

24 1. Introduction

25 With increasing the number of internet users, large amounts of data are being generated each day [18]. Cloud computing
26 provides the facility to store, access, and share data with other users anytime. The main limitation of the cloud paradigm
27 is its centralized storage design, which leads to a single point of failure issue. Cloud storage systems rely on Trusted
28 Third Party (TTP) to collect and store users' privacy-sensitive data, which is more vulnerable to security and attacks.
29 To address these challenges, blockchain has become popular as a decentralized and transparent data management
30 facility [23,42] that enables users to share and store information without any TTP. A blockchain is a peer-to-peer
31 distributed ledger in which a list of records called blocks are linked with each other and secured using a cryptographic
32 hash function [35]. It stores data on distributed nodes through a consensus mechanism that guarantees participant's
33 trust by having the same copy of the data [34,37].

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

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

44 – We propose a metadata extension based on existing research [1]. Our extension ensures privacy-aware data access
45 and enables trust between actors by recording each actor's actions on data.

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

- 46 – We propose a pointer system to manage the history of values that are stored in the DHT for a single piece of data.
47 It allows the data owner to maintain and access their transactions history in case of any updates in the pre-stored
48 data.
- 49 – We propose a traceability algorithm that enables actors to trace their data and verify the product's origin in a
50 decentralized platform.
- 51 – We design and evaluate our decentralized framework against linking, eavesdropping, spoofing, and modification
52 attacks.
- 53 – We provide a critical comparison of the proposed solution with state-of the-art decentralized solutions to show the
54 research gap.
- 55 – We also provide implementation details with security and privacy analysis and performance evaluation of our
56 framework over a wood supply chain scenario to demonstrate its feasibility.

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

62 2. Motivating Scenario and Research Problem

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

65 2.1. Motivating Scenario

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

71 – Forest manager

72 The forest manager identifies the trees that are good to make furniture (e.g oak) and cuts them into logs.

73 – Transporter

74 The transporter loads wood logs from the forest and transports them to the sawmill.

75 – Sawmill manager

76 It processes the logs and stores them for a specific time duration.

77 – Product assembler

78 It divides logs into pieces for further processing.

79 – Product seller owner

80 The product seller owner sells the furniture to the end customer.

81 – Customer

82 The customer takes the wooden furniture and confirms the origin of the wood using the proposed traceability
83 algorithm (see Section 4.4).

84 This scenario highlights the need for decentralized data management, security, privacy, traceability, and data updates
85 [36]. Frauds are common in the wood supply chain, for example, during transportation actors can replace high
86 quality wood with low quality wood [30]. Therefore, all actors participating in the supply chain want to trace products
87 to prevent frauds. To overcome this problem, Radio Frequency Identification (RFID) chips are used with the wood to
88 manage wood traceability [31]. However, existing solutions involve centralized storage to maintain the record of RFID
89 data, thus making single point of failure a major concern [26].

90 Therefore, blockchain, as a decentralized ledger technology that stores transactions in such a way that all participants
91 can easily access them without requiring any TTP, comes as an interesting technology for solving the single point
92 of failure issue. Each block of the blockchain keeps the hash of its previous block to make it impossible to modify

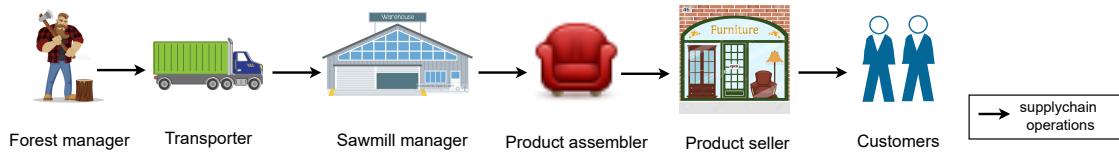


Figure 1. Wood supply chain and its actors.

93 the stored transactions thus ensuring immutability [25,13]. We can say that data cannot be modified once it has been
 94 recorded on the blockchain. However, our scenario highlights that actors of the wood supply chain need to write, read,
 95 and update data about their product. As well, they do not want to have their business information publicly available due
 96 to security and privacy concerns. There is a need for a solution that overcomes the immutability feature of blockchain,
 97 to enable actors to perform update operation on recorded data. At the same time, the designed solution must protect
 98 data from unauthorized access and guarantee data access depending on the actor's permission. The identified require-
 99 ments highlight our motivation to design decentralized data storage and management solution to ensure data access
 100 and updates, manage transactions history, security, privacy, data owner's control on their data, and product traceability
 101 in a single framework.

102 **2.2. Research Problems**

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

105 **– Data modification**

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

109 **– Data security and privacy**

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

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

118 **3. Background Knowledge and Related Work**

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

123 **3.1. Basic Concepts: Blockchain and DHT**

124 In 2008, blockchain technology [22] was introduced to the world and became popular due to its decentralized feature.
 125 The blockchain is a decentralized database that stores all the transactions that take place on the network. All participants
 126 on the network have the same copy of the transactions. Before adding each block to the blockchain, miners accept
 127 and verify the transactions using a consensus algorithm such as proof of work. By using proof of work or similar
 128 mechanism [15], miners solve very difficult mathematical calculations that should be accepted by other miners on the
 129 network [3]. After verifying the correctness of transactions by other miners, a block is appended to the end of the

130 chain [24]. Each block is comprised of a block version, timestamps, consensus signature, parent block hash, and many
 131 transactions. The parent block stores the hash of its previous block to form a blockchain that ensures the immutability
 132 of the stored data [32]. The hash is a unique value that ensures integrity of the entire blockchain from the initial block
 133 (known as genesis block) to the last.

134 A distributed hash table (DHT) is a decentralized data storage system that stores data as (key, value) pairs over
 135 a set of nodes that distribute the storage, possibly with some level of replication. As an example, a well-known DHT
 136 implementation is Kademlia [19]. Each node in the DHT maintains the keys it is responsible for and their corresponding
 137 values. A key is a unique identifier to its corresponding data value. Each key is generated by applying a hash function
 138 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
 139 retrieve the data, that is associated with the given key. A DHT node contains a routing table that maintains the identifier
 140 of its neighbor nodes. To find a (key, value) pair, a requesting node contacts the multiple nodes in the network until it
 141 reaches the destination node and finds the (key, value) pair. DHT has an advantage in terms of fault-tolerance because
 142 (key, value) pairs are replicated on multiple nodes in the network, that ensures data availability [43]. In addition, and
 143 as opposed to BC, it is scalable enough to manage large data volumes.

144 3.2. Blockchain and DHT-based data storage

145 There is a large amount of literature that combines DHT with blockchain to provide decentralized data storage. There is
 146 a large amount of literature that combines DHT with blockchain to provide decentralized data storage. A framework to
 147 manage personal data is proposed in [43]. The solution stores encrypted data (with shared key) on DHT and its pointer
 148 on the blockchain. Both service and user can query the data. However, existing work supports one type of encryption.
 149 Most work use a shared symmetric key for data encryption/decryption, as in [43], to query the data. In contrast,
 150 our framework provides run-time flexibility, which provides various types of data encryption and decryption during
 151 execution depending on users' needs and application requirements. In [43], it is not clearly explained how symmetric
 152 keys are protected from unauthorized access and where they are stored. In our work, we encrypt symmetric key with
 153 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
 154 the data.

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

160 In [1], the authors propose a decentralized data storage for PingER (Ping End-to-End Reporting) framework. The
 161 proposed framework stores metadata of the daily PingER files on a permissioned blockchain, while the original data
 162 is stored off-chain. However, their solution writes monitoring agent name and file locations on the permissioned
 163 blockchain, which is immutable and shared with other participants on the network. In addition, this solution does
 164 not record the data modification history in case of any modification in the data. Our framework design relies on the
 165 PingER proposal for the metadata structure, however, we integrate privacy and security management to enable role-
 166 based access control and privacy protection. Our solution enables data owners to control and access their private data.
 167 We also provide a solution to manage the previous versions of data using pointers that enable authorized users to access
 168 their transaction history. In addition, our work includes proof of concept prototype as well as empirical performance
 169 evaluation, which is not the case in PingER.

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

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

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

181 3.3. Other Decentralized Data Storage Solutions

182 An Ethereum-based blockchain platform is presented in [16]. The proposed solution allows companies partners to share
 183 data with each other. Original data are stored on off-chain storage such as MySQL, while a hash sum of corresponding
 184 data is sent to the blockchain. However, MySQL database is not scalable as DHT to manage a large amount of data [12].
 185 In addition, MySQL database becomes a single point of failure. In our solution, we use a DHT to store data as (key,
 186 value) pair, which can handle a large amount of data easily. In our framework, any authenticated user can efficiently
 187 retrieve the value with the help of a corresponding key. As well, our solution is fully decentralized and eliminates the
 188 risk of single point of failure.

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

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

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

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

214 In [41], the authors propose a decentralized IoT data sharing solution using IOTA Tangle and IPFS technology.
 215 The proposed solution uses centralized data handling unit (such as a local server) to collect and encrypt the data
 216 using asymmetric encryption, which becomes a single point of failure. In contrast, our proposed solution manage and

217 store data without any central party. The IPFS is used to upload the encrypted data, while the corresponding hash and
 218 metadata are managed on the IOTA Tangle. However, the IPFS network is immutable and stores files and its content
 219 permanently [14]. On the other hand, we use DHT to store the data and we extend it to allow data modification at any
 220 time. In contrast, our solution allows going through the history of data values and supports querying it.

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

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

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

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

248 4. Contribution

249 In this paper, we propose a secure and privacy-aware decentralized framework to support data storage, authorized data
 250 access, data mutability, management of their update history, and traceability. This section starts with the metadata
 251 structure that is immutable record of data operations. Then, it describes the overview of our proposed framework and
 252 follows with the detail of its execution or sequence. After that, it details the proposed algorithms. Each actor of the
 253 framework runs the same code that is structured into a set of components as depicted in Figure 3.

254 4.1. Metadata Structure

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

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

267 (such as data write, read, update, and delete) that is performed on the data. RFID_number is a unique data id of the log,
 268 lumber and product which is used to trace the items in the chain.

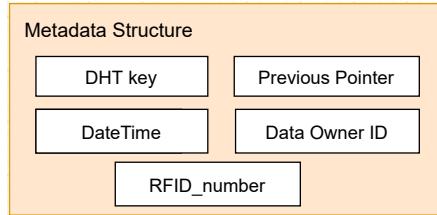


Figure 2. Metadata structure on the blockchain

269 4.2. Architecture Overview

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

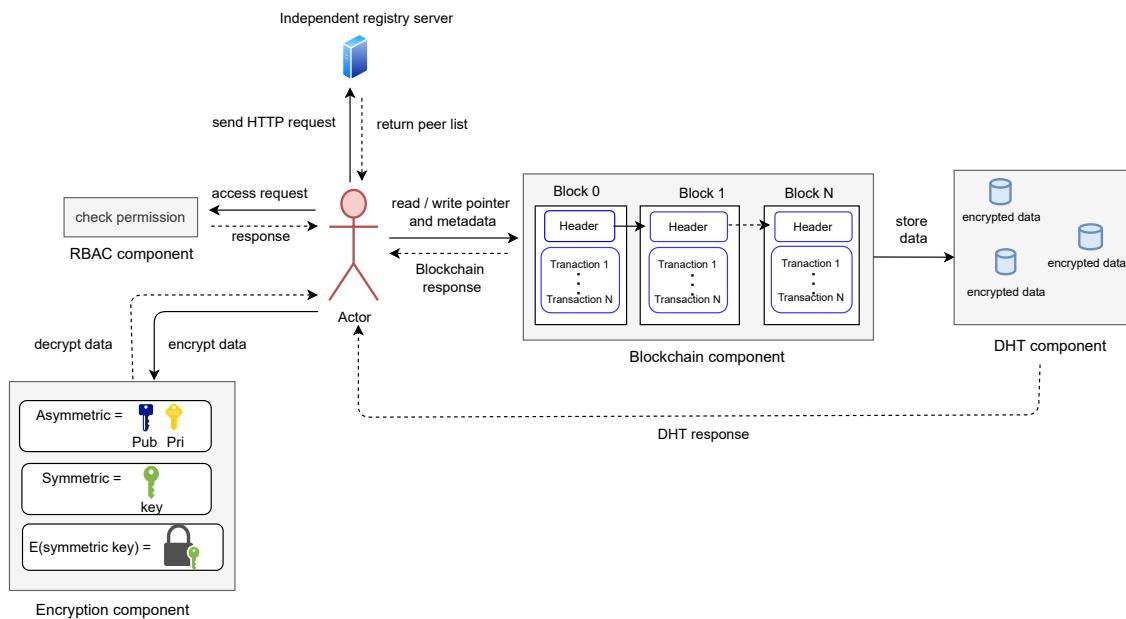


Figure 3. Overview of the decentralized framework

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

275 Let us illustrate the operation of our framework with the wood supply chain scenario developed earlier: an actor,
 276 for example a forest manager actor, starts the main program to store the number of logs and type of wood that he
 277 cuts. Then, he will call the `/peers` resource of the `registry_server` with the `'POST'` method to add its public
 278 key and Uniform Resource Locator (URL) to the list of connected peers or actors. After that, he will send a `'GET'`

279 request to the `/peers` resource to receive the information of available peers. Then, he will take a copy of the recent
 280 40 transactions of the blockchain using `/chain` resource with a 'GET' method⁴.

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

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

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

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

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



Figure 4. Overview of the proposed API using Swagger

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

300 4.3. Interaction via RESTful API

301 In this section, we detail the possible usage of our framework with a sequence diagram (Figure 6) that illustrates the
 302 interaction between an actor (e.g. a forest manager) and the framework using its RESTful API. We assume that every
 303 actor is already registered on the framework. An actor makes a 'POST' request to the `/chain` resource to write log
 304 data in the framework. Our solution assigns a unique data id (`RFID_number`) to the log that enables authorized actors
 305 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
 306 the response. Our framework stores the DHT key of this generated data in the metadata. Therefore, this DHT key points
 307 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
 308 sending another HTTP request as described in the links. To read the data, an actor would use the GET link that would
 309 call the `/chain/<id>` resource with method 'GET' to retrieve the representation of the log data. In the case of a
 310 successful response (HTTP code 200), our framework returns the representation of the log data. In case an actor wants
 311 to update their data, then they use the PUT link that makes a 'PUT' request to the `(/chain/<id>` resource). It will

⁴ 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.

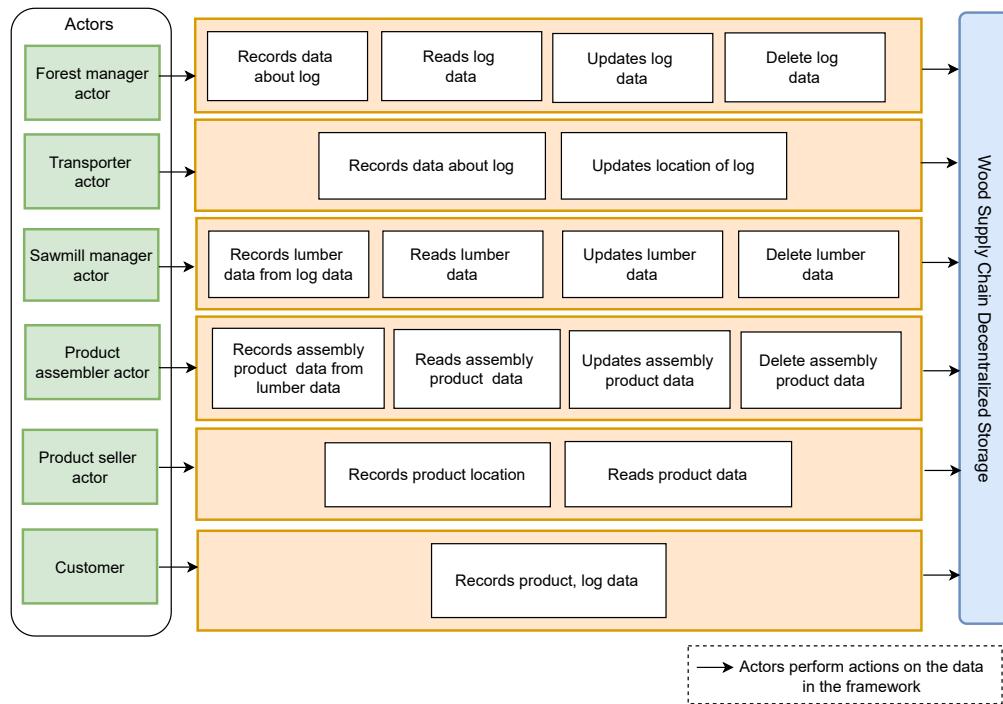


Figure 5. High-level representation of actors actions on the data

312 then write the new data against the same id. Then, a new metadata structure is created on the blockchain, and it contains
 313 the new DHT key of this updated data and the previous pointer of the old version of the data. Similarly, to delete the
 314 data, an actor may follow the DELETE link (`/chain/<id>` resource, method 'DELETE'). Our framework allows
 315 the authorized actor to delete specific data based on the id. After verifying the permission of the actor, it will delete the
 316 data. In this case, a new metadata structure is created on the blockchain that has a new DHT key with a NULL value.

317 4.4. Registration and Data Management

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

321 – Actor Registration

322 Algorithm 1 describes the actor's connection or registration procedure to the proposed framework using our REST-
 323 ful APIs. Once actor would successfully connect to the framework then they can perform different actions on the
 324 data such as write, read, update etc, and actors can also connect to other actors through HTTP requests. Each new
 325 actor needs to connect to the framework once to perform actions.

326 Firstly, the actor calls the `/peers` resource with 'GET' method to receive the available peer list (`p1`). After that,
 327 it calls the `/peers` resource 'POST' method to add its public key to the list of available peers and registers to
 328 the registry server. Then it sends a request to other peers to acknowledge the connected peer (`/peers` resource,
 329 'POST' method). If the current actor is already in the list then it will be disconnected or removed from the
 330 peer list using the `/peers` resource with 'DELETE' method. Then it sends a request to other available peers to
 331 acknowledge the disconnected peer.

332 – Data Management on the DHT

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

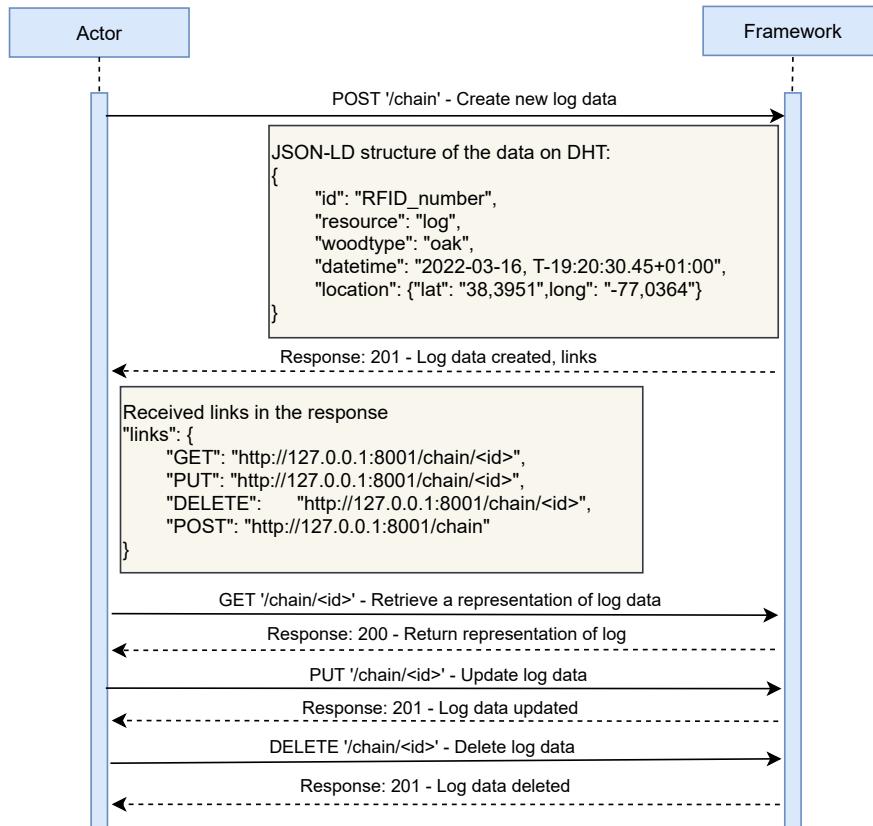


Figure 6. Sequence diagram of possible actor interactions with the framework

```

{
  "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"
  }
}

```

337 Then, `Authenticate(actor, role)` and `CheckPermission(actor, role, v)` verify that the
 338 current actor has the right permissions to store their data or not. The `CheckPermission` checks if the current actor
 339 has a role 'forest manager' then he is allowed to write, read, update, and delete their data in the decentralized
 340 platform.

341 After verifying the permission of the current actor, our framework provides different encryption methods (*em*)
 342 to encrypt the data before storing it on the decentralized ledger that ensures data security. An authorized actor is
 343 allowed to choose between `asymmetric em` and `symmetric em`. Asymmetric encryption is based on separate
 344 public and private keys. A public key is used to encrypt the data, while a corresponding private key is used to
 345 decrypt the data. In our motivating scenario, if a forest manager actor chooses `asymmetric em` then data will be
 346 encrypted with the data owner's public key, so later he can only access his data using his private key.

347 The authorized actor also has an option to choose `symmetric em` to encrypt the data, if he wants to enable other
 348 actors to read their data. A symmetric key is based on a single key to encrypt and decrypt the data. If the data
 349 owner chooses `symmetric em`, then our framework encrypts the data with the `symmetric` key and then this
 350 symmetric key will be encrypted with a data owner public key to protect the key from unauthorized actors.

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

```

351 Upon data read request, the data owner would encrypt this symmetric key using the requester's the public key to
 352 enable the authorized actors to read the data.

353 We store this encrypted symmetric key (*ek*) and encrypted data (*ed*) on the DHT. The *ed* stores on the DHT contains
 354 resource, woodtype, location (such as latitude and longitude) that shows the geographical location of the resource
 355 in the wood supply chain. Then, the function *FindLastTransaction* takes the data id such as (*rfid_number*)
 356 as input and returns previous pointer (*pp*) if it exists otherwise it returns 0. We store the metadata on the blockchain.
 357 The metadata includes DHT key (*dk*), pp, datetime, data owner id (*doid*), and data id (*rfid_number*).

358 – **Data Management on the Blockchain**

359 As an extension to the work in [1], we propose a metadata structure that manages the pointer and connects the
 360 different values attached to a specific piece of data to maintain its history. For example, a forest manager actor, as
 361 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"
    }
}

```

362 In this case, the proposed solution stores the DHT key as a new pointer of the log data in the metadata. Later, the
 363 data owner can access the data using a data id (RFID_number of the corresponding data). An actor can update
 364 some parts of the data against the same data id such as:

```

{
    "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

```

```

"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.

380 – **Traceability**

381 We propose an solution that maintains data id references to ensure traceability. It enables actors to verify the origin
 382 of the final product in the chain. Our solution assigns a unique data id (such as `RFID_number`) to the log, lumber,

383 and product. We assume that, RFID chips are inserted into the logs and then into the lumbers and final products.
 384 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"
  }
}
```

385 A log produces different pieces of lumbers and each lumber has unique id as RFID_number. The following code
 386 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"
  }
}
```

387 The data described above contains a reference id (RFID_number) of the log that was turned into lumbers. The
 388 different pieces of lumbers participate to build a final product such as wooden furniture. The following is a JSON
 389 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"
  }
}
```

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

396 In Algorithm 3, the did is an RFID_number of the item in the wood supply chain, and data (e.g location) of the
 397 item changes for the same did. Therefore, we can have multiple transactions on the blockchain against this did.

398 Whenever the location of the item would change then new metadata of the same did will be recorded on the
 399 blockchain, and the corresponding data is stored to the DHT. The `FindLastTransaction` function returns the
 400 last or recent transaction t of this `did`, which is a `RFID_number`. For instance, if we have `did` of the log then it
 401 finds the last transaction of this log.

402 This transaction t has the metadata that contains DHT key that points to the data recorded on the DHT. The
 403 function `CheckPermission` verifies if the current data requester is authorize to read the data or not depending
 404 on their role and HTTP verbs permission 'GET'. Then, the function `GetReferences` has the t as input and
 405 takes the `did` of the items. After that, it gets the previous references of this `did`. For example, if we have a input
 406 `did` as product id then it finds the previous references such as `RFID_number` of the lumbars. Then, it checks items
 407 (e.g lumbars) in the list and add items (e.g lumbars references) in the output list (o). Then, the `Traceability`
 408 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
 409 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
 $\triangleright l$: items list (variable)

```

1:  $l \leftarrow \emptyset$ 
2:  $t \leftarrow \text{FindLastTransaction}(did)$ 
3: if CheckPermission (actor, role, v) then
4:    $l \leftarrow \text{GetReferences} (t)$ 
5:   if  $l \neq \emptyset$  then
6:      $o \leftarrow \emptyset$ 
7:     for each  $i \in l$  do
8:        $o.\text{Append}(i)$ 
9:        $o.\text{Append}(\text{Traceability}(i))$ 
10:    end for
11:   return  $o$ 
12: end if
13: end if
14: return  $\emptyset$ 

```

410 5. Results and Evaluation

411 This section presents the results and performance evaluation of the proposed decentralized data storage framework.
 412 The evaluation framework is discussed in Section 5.1. The security and privacy analysis are presented in Section 5.2.
 413 Section 5.3 discusses the performance evaluation of our proposed framework.

414 5.1. Evaluation Framework

415 To implement and evaluate the performance of our framework, we used Python 3.7.0. We used a Python library⁵ to
 416 implement a blockchain to store the DHT key and metadata. We implemented a DHT using the Kademlia library⁶,
 417 which allows to store and get data linked with a given key on the peer-to-peer network. We used the cryptography
 418 RSA library to generate private/public keys and encrypt/decrypt the data. We conducted experiments and evaluated our
 419 framework on a 64-bit Windows operating system, Core i7 1.80 GHz processor, and 16 GB RAM.

⁵ https://github.com/satwikkansal/python_blockchain_app/tree/ibm_blockchain_post

⁶ <https://github.com/bmuller/kademlia>

420 **5.2. Privacy and Security Analysis**

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

429 **– Linking attack**

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

438

439 **– Eavesdropping attack**

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

443

444 **– Spoofing attack**

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

448

449 **– Modification attack**

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

455 **5.3. Performance Evaluation**

456 We evaluated the results according to time consumption and scalability with respect to the number of peers. We com-
457 puted the time consumption of the proposed solution according to the following parameters: actor's check permission,
458 data encryption/decryption using asymmetric or symmetric techniques, DHT access, and blockchain access. We ob-
459 served time consumption while performing data write, update, read, delete, and traceability operations. Figure 7 and 8
460 show the time consumption of the different parts of our solution, respectively using symmetric encryption and asym-
461 metric encryption.

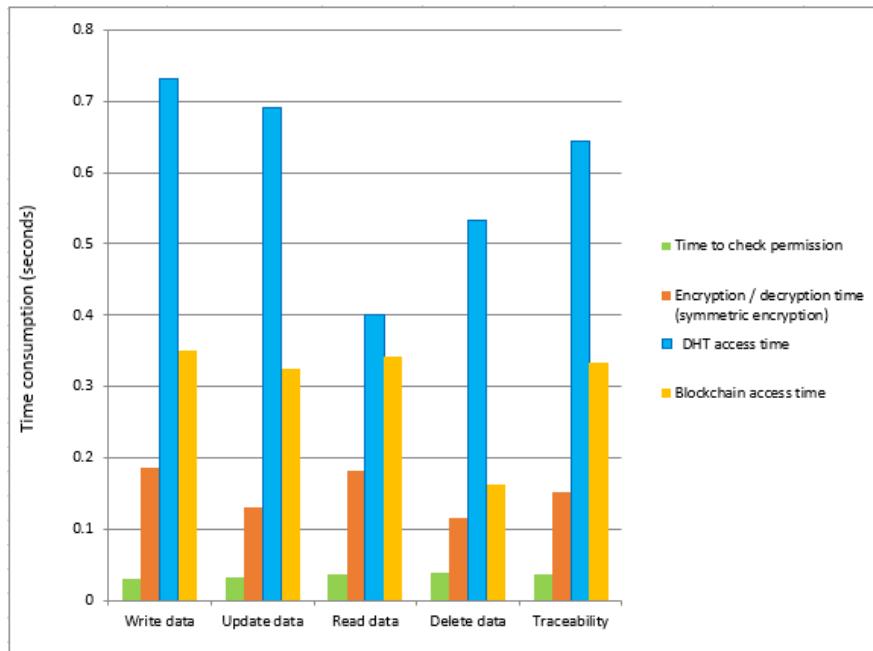


Figure 7. Time consumption using symmetric encryption

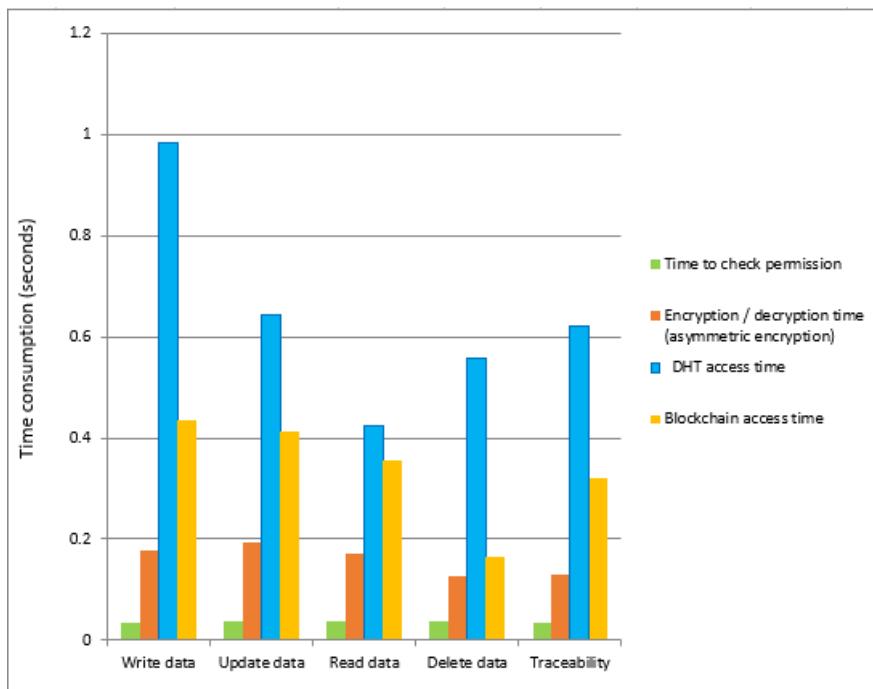


Figure 8. Time consumption using asymmetric encryption

462 The general trend of our measurements shows that DHT access takes most of the time needed, followed by
 463 blockchain access, encryption/decryption and then permission check, which makes sense since the DHT deals with

464 data storage and is therefore I/O-bound. We believe however that some low-level optimization is performed at this
 465 stage (see the scalability tests and discussion).

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

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

474 Moreover, we tested the scalability of our solution with a growing number of actors 1, 100, 200, 300, 400 and
 475 obtained a reasonable performance with 400 actors (please note that increasing the number of actors to more than
 476 400 would lead to additional synchronization problem, which would slow down the speed and performance. These
 477 problems are out of the scope of this paper.) The HTTP requests will be only partially processed in parallel, since
 478 they share the CPU time, and we tested our prototype with a quad-core CPU. In our solution, actors are the same as
 479 blockchain nodes and DHT nodes. We tested our solution with a number of 400 actors which are considered as 400
 480 nodes.

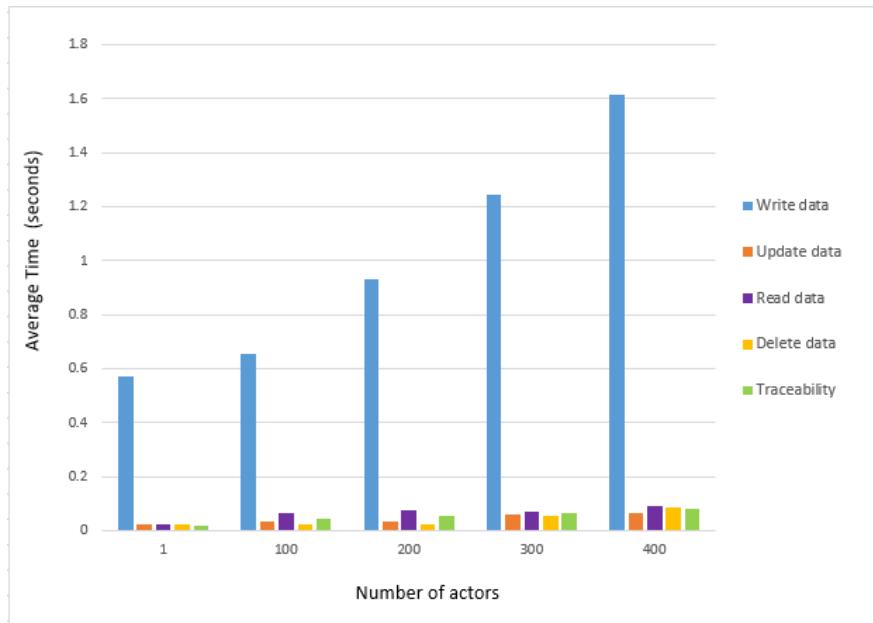


Figure 9. Average time consumption under different number of actors

481 We calculated the average time consumption of our prototype with an increasing number of actors. The actor regis-
 482 tration operation is performed only once for 1, 100, 200, 300, and 400 actor and the time costs is 0,0034 seconds,
 483 0,0039 seconds, 0,0041 seconds, 0,0046 seconds, and 0,0049 seconds respectively. Therefore, we tested our proto-
 484 type 100 times for all operations such as write data, update data, read data, delete data, and traceability. After that,
 485 we calculated the average time, Standard Deviation (SD), minimum (min), and maximum (max) values in seconds.
 486 Figure 9 depicts the average time consumption between a different number of actors, and detailed results statistics are
 487 summarized in Table 2.

488 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
 489 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
 490 seconds which is close to the SD of update data for the case of 200 actors. The data read provides an max value of
 491 0,0456 seconds which is less than the max value of read data for the case of 100, 200, 300, and 400 actors. The delete

492 data takes an average time of 0,0214 seconds which is close to the average time of delete data for 100 and 200 actors.
 493 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

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

500 Similarly, with the number of 200 actors, the average time to write data is 0,9325 seconds which is slightly higher
 501 than the average time to write data for the number of 1 and 100 actors. The update operation provides an SD of 0,0241
 502 seconds which is less than the SD of update data for the case of 100 actors. The read operation gives an average time of
 503 0,0738 seconds which is slightly close to the average time to read data for the case of 300 actors. The delete operation
 504 has a min value of 0,0171 seconds which is less than the min value for 1, 100, 300, and 400 actors. The traceability
 505 data operation gives a max value of 0,0695 seconds which does not show much difference from the max value of 100
 506 actors.

507 For the case of 300 actors, the write data operation gives an average of 1,2455 seconds which is slightly higher as
 508 compared to the average time to write data for 200 actors. The update gives an SD value of 0,0543 seconds which is
 509 close to the SD for the number of 400 actors. The read data operation gives a max value of 0,0836 seconds which is
 510 less as compared to the max value to read data for the number of 200 and 400 actors. The delete operation provides
 511 an SD of 0,0457 seconds which is less than the SD for 400 actors. The traceability takes an average time of 0,0636
 512 seconds which is higher than the average time for 1, 100, and 200 actors.

513 For the number of 400 actors, the average time to write data is 1,6121 seconds which is higher than the average
 514 time to write data for 1, 100, 200, and 300 actors. The update data operation gives an SD of 0,0551 seconds which is
 515 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
 516 value of 0,2419 seconds. The average time to delete data operation is slightly higher than the average time to update

517 operation for 1, 100, 200, and 300 actors. The traceability provides a max value of 0,0975 seconds which is close to
518 the max value for 300 actors.

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

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

526 6. Conclusion

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

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

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

543 Acknowledgment

544 The authors gratefully acknowledge the European Commission for funding the InnoRenew project (Grant Agreement
545 #739574) under the Horizon2020 Widespread-Teaming program and the Republic of Slovenia (Investment funding
546 of the Republic of Slovenia and the European Regional Development Fund). They also acknowledge the Slovenian
547 Research Agency ARRS for funding the project J2-2504.

548 References

- 549 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
550 International Conference on Big Data Science and Engineering (TrustCom/BigDataSE). pp. 1303–1308. IEEE (2018)
- 551 2. Aslam, S., Mrissa, M.: A restful privacy-aware and mutable decentralized ledger. In: European Conference on Advances in
552 Databases and Information Systems. pp. 193–204. Springer (2021)
- 553 3. Aslam, S., Tošić, A., Mrissa, M.: Secure and privacy-aware blockchain design: Requirements, challenges and solutions. Journal
554 of Cybersecurity and Privacy 1(1), 164–194 (2021)
- 555 4. Benisi, N.Z., Aminian, M., Javadi, B.: Blockchain-based decentralized storage networks: A survey. Journal of Network and
556 Computer Applications 162, 102656 (2020)
- 557 5. Chakravorty, A., Rong, C.: Ushare: user controlled social media based on blockchain. In: Proceedings of the 11th international
558 conference on ubiquitous information management and communication. pp. 1–6 (2017)
- 559 6. Domingue, J., Third, A., Ramachandran, M.: The fair trade framework for assessing decentralised data solutions. In: Companion
560 Proceedings of The 2019 World Wide Web Conference. pp. 866–882 (2019)
- 561 7. de Figueiredo, S., Madhusudan, A., Reniers, V., Nikova, S., Preneel, B.: Exploring the storj network: a security analysis. In:
562 Proceedings of the 36th Annual ACM Symposium on Applied Computing. pp. 257–264 (2021)

564 8. Hassanzadeh-Nazarabadi, Y., Küpcü, A., Özkasap, Ö.: Lightchain: A dht-based blockchain for resource constrained environments. arXiv preprint arXiv:1904.00375 (2019)

565 9. He, Q., Yan, J., Yang, Y., Kowalczyk, R., Jin, H.: A decentralized service discovery approach on peer-to-peer networks. IEEE

566 Transactions on Services Computing 6(1), 64–75 (2011)

567 10. Hei, Y., Liu, Y., Li, D., Liu, J., Wu, Q.: Themis: An accountable blockchain-based p2p cloud storage scheme. Peer-to-Peer

568 Networking and Applications 14(1), 225–239 (2021)

569 11. Huang, H., Zhou, X., Liu, J.: Food supply chain traceability scheme based on blockchain and epc technology. In: International

570 Conference on Smart Blockchain. pp. 32–42. Springer (2019)

571 12. Khamphakdee, N., Benjamas, N., Saiyod, S.: Performance evaluation of big data technology on designing big network traffic

572 data analysis system. In: 2016 Joint 8th International Conference on soft computing and Intelligent Systems (SCIS) and 17th

573 International Symposium on Advanced Intelligent Systems (ISIS). pp. 454–459. IEEE (2016)

574 13. Kumar, M.V., Iyengar, N.: A framework for blockchain technology in rice supply chain management. Adv. Sci. Technol. Lett

575 146, 125–130 (2017)

576 14. Legault, M.: A practitioner's view on distributed storage systems: Overview, challenges and potential solutions. Technology

577 Innovation Management Review 11(6), 32–41 (2021)

578 15. Li, W., Andreina, S., Bohli, J.M., Karame, G.: Securing proof-of-stake blockchain protocols. In: Data Privacy Management,

579 Cryptocurrencies and Blockchain Technology, pp. 297–315. Springer (2017)

580 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)

581 17. Lykousas, N., Koutsokostas, V., Casino, F., Patsakis, C.: The cynicism of modern cybercrime: Automating the analysis of

582 surface web marketplaces. arXiv preprint arXiv:2105.11805 (2021)

583 18. Marr, B.: How much data do we create every day? the mind-blowing stats everyone should read. forbes. may, 21 2018 (2018)

584 19. Maymounkov, P., Mazieres, D.: Kademia: A peer-to-peer information system based on the xor metric. In: International Work-

585 shop on Peer-to-Peer Systems. pp. 53–65. Springer (2002)

586 20. Mikroyannidis, A., Third, A., Domingue, J.: A case study on the decentralisation of lifelong learning using blockchain technol-

587 ogy. Journal of Interactive Media in Education 2020(1) (2020)

588 21. Moser, M.: Anonymity of bitcoin transactions. In: Münster Bitcoin Conference (MBC), Münster, Germany (July 2013)

589 22. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. Decentralized Business Review p. 21260 (2008)

590 23. Nakamoto, S., Bitcoin, A.: A peer-to-peer electronic cash system. Bitcoin.–URL: <https://bitcoin.org/bitcoin.pdf> 4 (2008)

591 24. Ølnes, S., Ubacht, J., Janssen, M.: Blockchain in government: Benefits and implications of distributed ledger technology for

592 information sharing (2017)

593 25. Pazaitis, A., De Filippi, P., Kostakis, V.: Blockchain and value systems in the sharing economy: The illustrative case of backfeed.

594 Technological Forecasting and Social Change 125, 105–115 (2017)

595 26. Podduturi, P.R., Maco, T., Ahmadi, P., Islam, K.: Rfid implementation in supply chain management. International Journal of

596 Interdisciplinary Telecommunications and Networking (IJITN) 12(2), 34–45 (2020)

597 27. Ramachandran, M., Chowdhury, N., Third, A., Domingue, J., Quick, K., Bachler, M.: Towards complete decentralised verifica-

598 tion of data with confidentiality: Different ways to connect solid pods and blockchain. In: Companion Proceedings of the Web

599 Conference 2020. pp. 645–649 (2020)

600 28. Shafagh, H., Burkhalter, L., Hithnawi, A., Duquennoy, S.: Towards blockchain-based auditable storage and sharing of iot data.

601 In: Proceedings of the 2017 on Cloud Computing Security Workshop. pp. 45–50 (2017)

602 29. Shrestha, A.K., Vassileva, J., Deters, R.: A blockchain platform for user data sharing ensuring user control and incentives.

603 Frontiers in Blockchain 3, 48 (2020)

604 30. da Silva, D.L., Corrêa, P.L.P., Najm, L.H.: Requirements analysis for a traceability system for management wood supply chain

605 on amazon forest. In: 2010 Fifth International Conference on Digital Information Management (ICDIM). pp. 87–94. IEEE

606 (2010)

607 31. Sirkka, A.: Modelling traceability in the forestry wood supply chain. In: 2008 IEEE 24th International Conference on Data

608 Engineering Workshop. pp. 104–105. IEEE (2008)

609 32. Swan, M.: Blockchain thinking: The brain as a decentralized autonomous corporation [commentary]. IEEE Technology and

610 Society Magazine 34(4), 41–52 (2015)

611 33. Tian, F.: An agri-food supply chain traceability system for china based on rfid & blockchain technology. In: 2016 13th interna-

612 tional conference on service systems and service management (ICSSSM). pp. 1–6. IEEE (2016)

613 34. Toyoda, K., Mathiopoulos, P.T., Sasase, I., Ohtsuki, T.: A novel blockchain-based product ownership management system

614 (poms) for anti-counterfeits in the post supply chain. IEEE access 5, 17465–17477 (2017)

615 35. Tschorsch, F., Scheuermann, B.: Bitcoin and beyond: A technical survey on decentralized digital currencies. IEEE Communica-

616 tions Surveys & Tutorials 18(3), 2084–2123 (2016)

617 36. Tzoulis, I., Andreopoulou, Z.: Emerging traceability technologies as a tool for quality wood trade. Procedia Technology 8,

618 606–611 (2013)

619 37. Voronchenko, K.: Do you need a blockchain? Supervised by Ivo Kubjas 22 (2017)

620

621

622 38. Westerkamp, M., Victor, F., Küpper, A.: Blockchain-based supply chain traceability: Token recipes model manufacturing pro-
623 cesses. In: 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Commu-
624 nications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData). pp.
625 1595–1602. IEEE (2018)

626 39. Wilkinson, S., Boshevski, T., Brandoff, J., Buterin, V.: Storj a peer-to-peer cloud storage network. <https://www.storj.io/storj2014.pdf> (2014)

627 40. Xu, L., Shah, N., Chen, L., Diallo, N., Gao, Z., Lu, Y., Shi, W.: Enabling the sharing economy: Privacy respecting contract
628 based on public blockchain. In: Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts. pp. 15–21
629 (2017)

630 41. Zheng, X., Lu, J., Sun, S., Kiritsis, D.: Decentralized industrial iot data management based on blockchain and ipfs. In: IFIP
631 International Conference on Advances in Production Management Systems. pp. 222–229. Springer (2020)

632 42. Zheng, Z., Xie, S., Dai, H.N., Chen, X., Wang, H.: Blockchain challenges and opportunities: A survey. International Journal of
633 Web and Grid Services 14(4), 352–375 (2018)

634 43. Zyskind, G., Nathan, O., et al.: Decentralizing privacy: Using blockchain to protect personal data. In: 2015 IEEE Security and
635 Privacy Workshops. pp. 180–184. IEEE (2015)

636