UNIVERSITY OF PRIMORSKA
FACULTY OF MATHEMATICS, NATURAL SCIENCES AND
INFORMATION TECHNOLOGIES

DOKTORSKA DISERTACIJA
(DOCTORAL DISSERTATION)

ZASEBNOST IN VARNOST PODATKOV ZA DECENTRALIZIRANI SPLET STVARI

(DECENTRALIZED DATA PRIVACY AND SECURITY FOR THE WEB OF THINGS)

SIDRA ASLAM

KOPER, 2022

UNIVERSITY OF PRIMORSKA
FACULTY OF MATHEMATICS, NATURAL SCIENCES AND
INFORMATION TECHNOLOGIES

DOKTORSKA DISERTACIJA
(DOCTORAL DISSERTATION)

ZASEBNOST IN VARNOST PODATKOV ZA DECENTRALIZIRANI SPLET STVARI

(DECENTRALIZED DATA PRIVACY AND SECURITY FOR THE WEB OF THINGS)

SIDRA ASLAM

KOPER, 2022

MENTOR: PROF. DR. MICHAEL MRISSA

Acknowledgement

I would like to express my deepest gratitude to my mentor Prof. Michael Mrissa, who always supported, encouraged, and motivated me during my Ph.D. His guidance helped me throughout my Ph.D. research. I am truly grateful to him for his patience, guidance, discussions, and immense knowledge.

I would like to 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). I would like also to acknowledge the Slovenian Research Agency ARRS for funding the project J2-2504.

I would like to acknowledge the European Cooperation in Science and Technology (COST) Action CA19126—Positive Energy Districts European Network (PED-EU-NET) for the financial support during my Short Term Scientific Mission (STSM) in Hungary.

Last, but not the least, my family deserves endless gratitude. Special thanks to my parents, husband, and siblings for their endless courage and motivation. My Ph.D. would not have been possible without their support all along the way. Furthermore, I would like to express my gratitude to my friends for their support and valuable time.

Abstract

Decentralized data privacy and security for the Web of things

The wood supply chain (WSC) involves several actors to store, process and transport wood, from raw logs, to lumber, to final products. Nowadays, the need for traceability in the WSC has become crucial. From a societal perspective, an improved quality of traceability enables limiting fraud and illegal logging in protected areas and preservation of our ecosystem. While traceability provides end users with complete transparency concerning the provenance of the wood products they are buying, it also provides a means for the other WSC actors to improve the management of their assets, guarantee the origin for the wood production, and optimize their operation. Typical solutions for traceability rely on embedded technology combined with sensors to track the items through the WSC and gather data in a central database. As most existing solutions to maintain product traceability depend on such a centralized infrastructure that acts as a Trusted Third Party (TTP), they suffer from single point of failure, security, and privacy issues. Over the past few years, blockchain technology has gained popularity due to its decentralized design, which overcomes some of these limitations, therefore providing an interesting alternative to existing centralized solutions.

However, relying on blockchain technology brings up new limitations related to data privacy, access control, and data updates. In this dissertation, we explore a range of solutions to address these limitations, according to the following research questions: 1) how to provide decentralized security, privacy, and traceability for data storage? 2) how to efficiently manage data updates when blockchain is involved in the process? 3) what is the best way to use distributed ledgers to provide decentralized trust for secure data storage? 4) how to protect users' privacy-sensitive information on a decentralized data storage solution?

To contribute to the above research questions this dissertation describes the design and development of a Web-based decentralized storage framework that provides data privacy, security, and mutability as well as traceability for the WSC. To do so, it explores, through literature review and original contributions, how to combine blockchain with a distributed hash table, access control ontology, and multiple encryption mechanisms. We design a decentralized framework in which each peer offers REST (Representational State Transfer) APIs (Application Programming Interfaces) to operate, thus ensuring interoperability over the Web. We rely on a metadata structure stored on the blockchain in combination with a distributed hash table to offer a

strong decoupling between data access and storage. We propose several algorithms to address the security and privacy issues we are facing, so that data owners keep control of their data. We present the operation of our framework and demonstrate how it enables run-time data protection. Concerning data updates, we propose a solution that uses a pointer system to connect the different versions of the data together. The proposed solution allows actors to access their update history. To provide end users with complete traceability, we propose an algorithm that enables the actors of the WSC to trace the data and verify product origin. Besides this, we design an access control ontology to manage role hierarchies as well as relationships between actors and handle complex permissions for data access. To navigate data, we propose a generic Web client that relies on the Hypermedia As The Engine Of Application State (HATEOAS) constraint. We design a proxy that is available as a REST service, and that injects links into the response messages and forwards them to the client. Our solution includes a REST API that enables the management of templates to generate HATEOAS links. We evaluate the performance of all our contributions with a proof-of-concept prototype and batteries of tests that implement time measurements to show their applicability, scalability, their advantages and limitations, with different numbers of actors.

Key words: Distributed ledger, Security, Privacy, Blockchain.

Izvleček

Zasebnost in Varnost Podatkov za Decentralizirani Splet Stvari

Dobavna veriga lesa (DVL) vključuje več akterjev kateri urejajo shranjevanje, predelavo in transport lesa, od surovih hlodov lesa, do končnih izdelkov. Dandanes je potreba po sledljivosti v DVL postala ključna. Iz družbenega vidika, sledljivost omogoča omejevanje goljufij in nezakonite sečnje lesa na zavarovanih območjih ter ohranjanje našega ekosistema. Sledljivost zagotavi končnim uporabnikom popolno preglednost v zvezi s poreklom lesnih izdelkov, in omogoči akterjem DVL, da izboljšajo upravljanje svojih sredstev, zagotovijo izvor lesa in optimizirajo njihovo delovanje. Tipična rešitev za sledljivost, se zanaša na vgrajeno tehnologijo (RFID čipi) v kombinaciji s senzorji za sledenje predmetov skozi DVL in zbiranje podatkov v centralni bazi podatkov. Ker je večina obstoječih rešitev za ohranjanje sledljivosti izdelkov odvisna od tako centralizirane infrastrukture, ki deluje kot zaupanja vredna tretja oseba, so taki sistem podvrženi problemu kritične točke odpovedi. V zadnjih letih je tehnologija veriženja blokov pridobila na popularnosti zaradi svoje decentralizirane zasnove, ki predstavlja zanimivo alternativo obstoječim centraliziranim rešitvam.

Vendar pa uporaba blockchain tehnologije prinaša nove omejitve, povezane z zasebnostjo podatkov, nadzorom dostopa in posodabljanjem podatkov. V tej disertaciji raziskujemo vrsto rešitev za odpravo teh omejitev glede na naslednja raziskovalna vprašanja: 1) kako zagotoviti decentralizirano varnost, zasebnost in sledljivost za shranjevanje podatkov? 2) kateri je najboljši način za uporabo tehnologije razpršene evidence za zagotavljanje decentraliziranega zaupanja za varno shranjevanje podatkov? 3) kako ravnati z dostopom do podatkov in zaščititi informacije, občutljive na zasebnost uporabnikov, s decentralizirano rešitvijo za shranjevanje podatkov? 4) kako učinkovito upravljati posodobitve podatkov na blockchainu?

Da bi prispeval k zgornjim raziskovalnim vprašanjem, ta disertacija opisuje zasnovo in razvoj decentraliziranega RESTful ogrodja za shranjevanje podatkov, ki zagotavlja zasebnost, varnost, spremenljivost in sledljivost za DVL. V ta namen s pregledom literature in izvirnimi prispevki se raziskuje, kako združiti blockchain z porazdeljeno zgoščeno tabelo, ontologijo nadzora dostopa in več mehanizmov šifriranja. V enem samem ogrodju oblikujemo več algoritmov in strukturo metapodatkov, shranjenih v verigi blokov in v kombinaciji z porazdeljeno zgoščeno tabelo, da ponudimo močno ločitev med dostopom do podatkov in shranjevanjem, hkrati pa nudimo priložnost za reševanje vprašanj varnosti in zasebnosti. Vsak enakovrednik v našem ogrodju ponuja API-je RESTful za delovanje in tako zagotavlja interoperabilnost prek spleta.

V disertaciji, predstavljamo delovanje našega ogrodja in prikazujemo, kako omogoča zaščito podatkov med izvajanjem. V zvezi s posodabljanjem podatkov predlagamo rešitev, ki uporablja sistem kazalcev za povezovanje različnih različic podatkov. Predlagana rešitev omogoča akterjem dostop do svoje zgodovine posodobitev. Da bi končnim uporabnikom zagotovili popolno sledljivost, predlagamo algoritem, ki akterjem DVL omogoča sledenje podatkom in preverjanje izvora izdelka. Poleg tega oblikujemo ontologijo nadzora dostopa za upravljanje hierarhij vlog, pa tudi odnosov med akterji in upravljanje kompleksnih dovoljenj za dostop do podatkov. Predlagamo generičnega odjemalca HATEOAS za krmarjenje po podatkih z uporabo API-jev RESTful.

V disertaciji predstavimo proxy, ki je na voljo kot REST storitev, in vnese povezave v povratno sporočilo in jih posreduje odjemalcu. Naša rešitev vključuje REST API, ki omogoča upravljanje predlog za ustvarjanje povezav HATEOAS. Učinkovitost vseh naših prispevkov ocenjujemo s prototipom za dokaz koncepta in serijo testov, ki izvajajo meritve časa, da pokažejo njihovo uporabnost, razširljivost, njihove prednosti in omejitve, z različnim številom akterjev.

Ključne besede: tehnologija razpršene evidence, Varnost, Zasebnost, Blockchain

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Over the last decades, the Internet, and then the Web, have been providing a reliable protocol stack to support various applications, from data storage to live streaming. The Internet acts as a global network that enables devices to communicate with each other all over the world, mostly thanks to the Internet Protocol (IP[1]), which provides identifiers for devices [6]. Concerning message transport, the Transmission Control Protocol (TCP[2]) supports reliable communication to exchange data between client and server and the User Datagram Protocol (UDP[3]) is suitable for applications such as live streaming [31].

A diverse range of applications and services are provided over the Internet such as communication services (e.g e-mail), file exchange services, peer-to-peer networks, and last but not least, the Web. These services are supported by various protocols such as e-mail, relying on the Simple Mail Transfer Protocol (SMTP[4]) and Post Office Protocol (POP3[5]), file exchange services depending on the File Transfer Protocol (FTP[6]), and peer-to-peer networks such as Kademlia [113].

As a service that operates over the Internet, the Web, since its first iteration, provides access to static HyperText Markup Language (HTML[7]) pages linked through hypermedia and easily located through Uniform Resource Identifiers (URI[8]) [129,170]. The HyperText Transfer Protocol (HTTP[9]) is responsible for the communication between a Web client (e.g browser) and a Web server using HTTP messages (request and response) [5].

Thanks to this protocol stack, the Web facilitates access and sharing of information. The first implementation of the Web was mostly read-only, meaning that it consisted of static pages with predefined contents [5]. At this stage, the purpose of Web clients was mostly to read and search for information. However, at the beginning of this century, the Web became dynamic, driven by the development of applications

[1] https://datatracker.ietf.org/doc/html/rfc791
[2] https://www.ietf.org/rfc/rfc793.txt/
[3] https://datatracker.ietf.org/doc/html/rfc8085
[4] https://datatracker.ietf.org/doc/html/rfc5321/
[5] https://datatracker.ietf.org/doc/html/rfc1081/
[6] https://datatracker.ietf.org/doc/html/rfc959/
[7] https://www.w3.org/html/
[8] https://datatracker.ietf.org/doc/html/rfc3986
[9] https://datatracker.ietf.org/doc/html/rfc2616

like YouTube, Twitter, Facebook, Instagram, etc [120]. Becoming dynamic means that Web servers were able to host not only contents, but programs (mostly scripts in languages like PHP, ASP.NET, JSP, or compiled server-side components like Java Servlets) that allowed users to interact with Web pages [5]. This development made it possible for users to contribute to Web contents [173].

At the same time, the industry started to see the benefit for business-to-business interactions over the Web through software clients. Following this trend, the development of the Web service protocol stack, enabled software clients to call remote applications using the Web, with complete abstraction of the underlying operating systems, protocols and programming languages. The Web service protocol stack consists of the Simple Object Access Protocol[10] (SOAP), the Web Service Description Language[11] (WSDL) and the Universal Description, Discovery, and Integration[12] (UDDI) registry. SOAP is based on the Extensible Markup Language (XML) data format and provides strict rules to exchange data between client and server independently of the transport protocol [48]. WSDL is used to provide for service description, which helps the client application to understand how to use the functionality of the service [45]. UDDI is a centralized registry that supports service discovery. However, this protocol stack did not receive long-term adoption, due to several design issues.

First, SOAP, while transport protocol agnostic, is in practice (mis)using HTTP as a *de facto* standard for data transfer, without properly exploiting the different features that HTTP offers as an application-level protocol [106, 119, 142, 184]. WSDL describes functionalities in a way that is similar to performing Remote Procedure Calls (RPC) over HTTP, and UDDI was never widely adopted when users simply find services online using search engines [19, 184].

Due to these limitations, amongst others, the design of Web services has evolved towards the guidelines from the REST architectural style, that properly exploits the Web and its HTTP protocol [85]. REST is useful to allow developers to design their Application Programming Interface (APIs), in this case Web APIs, by following some well-defined principles that enable, amongst others, generic clients using resource-oriented calls that make proper usage of the HTTP verbs or methods (mainly GET/POST/PUT/DELETE), better decoupling between clients and servers through stateless communication, in turn guaranteeing better scalability [61]. The HTTP verbs allow communication between client and server. The GET HTTP verb is used to retrieve the representation of the specific resource. The POST verb sends data to be processed on the server side, which may create a sub-resource. The PUT verb is used to record the contents of the request at a specific location, which might create or update an existing resource. Finally, the DELETE verb deletes the resource on the server [144]. The REST resources are identified through Uniform Resource Identifier (URI) [164]. HTTP status codes help to identify whether HTTP request is successfully completed or not. Unlike SOAP, REST is not restricted to the XML format and supports different data formats such as JavaScript Object Notation (JSON), which can be agreed on using the HTTP content negotiation mechanism [70]. JSON is a human-understandable language that is used to interchange data and it presents

[10] `https://www.w3.org/TR/soap/`
[11] `https://www.w3.org/TR/wsdl/`
[12] `http://www.uddi.org/pubs/uddi_v3.htm`

several advantages, such as a much reduced verbosity, as compared to XML [70].

While Web services started to be widely used, the need for large-scale infrastructures to support them and offer seamless access to applications became crucial, thus motivating the development of cloud computing [34,171]. By centralizing hardware resources into a homogeneous infrastructure and decoupling the management of hardware resources from the software operation through virtualization techniques, cloud computing brings large scale savings and facilitates the scalability of applications as it allows hardware to be allocated at run-time to the applications that require more resources than others.

However, one of the most critical issues related to cloud infrastructures is their centralized design, which is subject to security weaknesses as they become a single point of failure [172], more likely to be the target of attacks. Indeed, the cloud acts as a Trusted Third Party (TTP) to provide highly sensitive services such as authentication or storage of privacy-sensitive data and can be an interesting focus point due to the large amount of data available.

In this context, decentralized approaches offer an interesting alternative to provide users with the same services than the cloud provides, without the aforementioned limitations. However, algorithms for data management, processing and storage need to be redesigned to suit this new context.

In particular over the past few years, blockchain as a distributed ledger technology has gained much attention to enable trust between different parties without any TTP [121]. Due to its decentralized and transparent nature supported by verifiable cryptographic mechanism, blockchain eliminates the risks associated to the single point of failure and facilitates users' trust [166]. As such, blockchain technology has a significant impact on society with many applications related to business, healthcare, supply chain, etc. However, despite the advantages of decentralization, transparency, and trust, the blockchain technology also raises several limitations that hamper its capacity to be integrated into a solution. In this thesis, we investigate some of these limitations related to data immutability, privacy, and security.

The rest of this chapter is organized as follows. First, we describe the motivating use case that we use throughout this dissertation. We show how this use case raises the research challenges that we address in our work. Then, we detail our research challenges into specific research questions and hypotheses. We present our thesis goals and show how they cover the identified research questions. Finally, we detail our research methodology, scientific contributions, and outline the plan of the dissertation.

1.1 Motivating Scenario

Our motivating scenario takes place in the context of the wood supply chain (WSC). More specifically, we study the traceability problem in the context of massive wood furniture production, where the WSC transforms raw wooden logs into a final product to be delivered to the end customer. Traceability in the WSC involves keeping track of every piece of massive wood along its path through the chain to avoid frauds, for instance, during the transportation process replacing high-quality wood with

low-quality wood is very common. Therefore, it is of primordial importance that the involved stakeholders can monitor the location of their assets over time and jointly guarantee the origin of the wood. At the physical level, to tackle this issue, Radio Frequency Identification (RFID) chips are typically inserted into the wood to maintain product traceability [78]. However, concerning the digital level, many challenges remain, which we motivate in the following. Figure 1.1 gives a simplified overview of the different steps in the WSC and the actors involved, from the raw log that is being cut, transported, processed into logs, assembled into a final product, stored, to reach the last step where the final product is ready for sale.

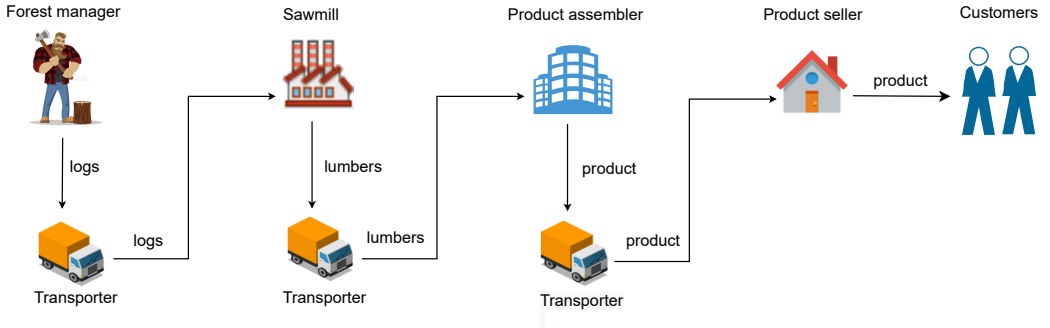

Figure 1.1: Simplified overview of the wood supply chain and its actors.

In the following, we describe the different activities of the identified actors that are involved in the process of massive wood furniture production:

- The **forest manager** manages the forest and answers the demand for wood, provides its clients with logs.
- The **transporter** conveys any product (in our example logs, lumber or furniture) from one location to another.
- The **sawmill** processes and cuts logs into lumbers and stores them temporarily.
- The **product assembler** cuts lumbers into specific dimensions and assembles them into a final product (e.g. furniture).
- The **product seller** exposes assembled wood product and organises marketing strategies (e.g. discount operations) to sell it.
- **Customers** order and purchase wooden products and should be able to track their product origin.

The scenario discussed above is generally realized upon a sufficient number of customer orders. Please note that for the sake of simplicity, in order to focus on the problems at stake, our scenario relies on a simplified version of the WSC with a limited number of actors, leaving aside the large scale aspects that occur when the different actors of the WSC connect with each other to obtain large scale benefits. Therefore, upon customer order, the product seller needs to provide for the ordered

furniture and asks the furniture assembly company to deliver assembled furniture. The furniture assembly company needs lumber to assemble final products, and in turn requires the sawmill to provide them with it. As a consequence, the sawmill company needs logs from the forest manager. The transporter is responsible for conveying the goods (logs, lumber or products) from an actor's location to another.

All these actors are involved in the WSC, they collaborate to realize a joint workflow and they need to exchange data about the goods they process in order to fulfil the customers' needs for traceability. To illustrate our example, the forest manager will record provenance and quality information about the logs it sends to the sawmill, this latter will in turn generate new data for lumber that needs to be connected to the logs they come from, and in turn the lumber pieces that will be assembled in a piece of furniture need to be identified as well, and so on until the final product reaches the customer. At this stage, we explain traceability to allow a customer to visualize exactly what the furniture is composed of and from which location each piece of it comes from.

As explained in the previous section, due to the limitations that typical centralized architectures show (i.e. cloud-based), our motivating example highlights the need to design a decentralized framework to support the WSC. The proposed framework must ensure interoperability, permanent trusted data storage, and guarantee of privacy between actors. It then comes as natural option to rely on the Web for interoperability, due to the multiplicity and diversity of actors. We develop a Web-based set of APIs to support our framework in Chapter 3. We also extend our framework with a generic client and proxy solution that extends HTTP messages to support HATEOAS in Chapter 6.

With respect to trusted decentralized data storage, typically, existing solutions rely on a centralized database to manage and store RFID data, which leads to a single point of failure, thus motivating research towards decentralized solutions. We need to look for a solution that provides data security, privacy, traceability, trust, and supports data updates [168]. Concerning decentralized trust, blockchain is worth exploring nowadays as a decentralized ledger technology that stores transactions in such a way that all participants can easily access them without requiring any TTP. It indeed comes as an interesting technology for solving the single point of failure issue. It is completely decentralized to manage the data where no single authority is controlling the whole network, which also eliminates the need for TTP [89]. Its decentralized nature increases the transparency where everyone on the network can access the transactions [89]. In addition, it ensures trust because each participant has the same copy of the transactions that are stored in the distributed ledger [174]. Participants may have to trust the distributed ledger instead of trusting each other. In the blockchain, each block stores the hash to its previous block to avoid the modification of stored data and ensure immutability [92,131]. It means that, once the data has been stored on the blockchain, it cannot be modified. However, blockchain is not designed to support data updates because of its immutability feature as discussed above [145]. It is also not adapted to store large amount of data because its nodes have limited storage size to store the data [57, 89]. In our scenario, when a log is transported from one point to another, its recorded current location must change, and the history of its previous location must be preserved as well. Other decentralized

solutions like Distributed Hash Table (DHT) are in this case better adapted than blockchain due to their design that does not necessarily entail complete replication of the data (DHTs are actually quite flexible in this respect). Our work explores how to combine them in a single design to gain the best of both worlds. This aspect is developed in Chapter 4.

In this context, actors do not want to disclose their business information publicly due to security and privacy concerns. There is a need to develop a solution that overcomes the immutability feature of blockchain to allow actors to perform update operations on recorded data. At the same time, the designed solution needs to guarantee data protection from malicious actors and ensures data access depending on the actor's permission. In the WSC, the authorizations for data access between different actors can be complex and dynamically evolve with the business relationships, thus justifying the use of ontologies to model them. Such aspect is develop in Chapter 5.

Overall, our WSC scenario motivates the design of a decentralized data storage and management solution that ensures data updates, manages transactions history, provides an acceptable level of security and privacy, allows data owners to control their data, and supports complex relationships between actors. All these features need to be smoothly integrated into a single framework to eventually provide a decentralized system for product traceability. While designing our contribution, we will make sure that it remains application domain-agnostic so that it can be widely applicable to any other supply chain scenario. The motivating use case discussed above highlights following research problems.

1.2 Research Problems

Existing work explored the use of blockchain as distributed ledger technology to support decentralized data storage and management, which eliminates the need for TTP to avoid the single point of failure issue [7, 66, 103, 175]. In the following we discussed the research problems (RP), that we address in the dissertation.

- **RP1: Data immutability and blockchain:** According to our scenario, actors may want to update their data (for example product location for the transporter). However, blockchain stores data permanently and does not allow to update and delete the data once it has been stored in the chain, because of its immutability feature [92]. The challenge consists in overcoming this limitation while keeping the main property that makes the blockchain interesting, the verifiable trust in data immutability. There is a need to develop a solution that, while respecting the immutability characteristic of blockchain, also enables update and delete operations on stored data.

- **RP2: Data privacy:** Blockchain stores data publicly [26, 117]. This is an issue when it comes to privacy-sensitive data. However, data privacy issues rise when unauthorized parties reveal privacy-sensitive information of actors. Additionally, a data privacy breach occurs when data is stored on a decentralized platform where data owners have no control over their data and are unable to decide who can have access to their data, and which part of the data can

be accessed. The research problem is about including a privacy management system into the blockchain operation in a decentralized way, without breaking the blockchain design.

- **RP3: Data security:** There is a need to enforce data security: 1) confidentiality to protect data from unauthorized access, 2) integrity to guarantee that data remains unaltered, 3) availability to make sure that an authorized actor can always access data when needed, 4) non-repudiation to ensure that the actor cannot deny a data operation once it has been performed.

  As well, the multiplicity of needs depending on the context in which the solution is utilized makes it interesting to develop different solutions for security. The research challenge is to integrate different security mechanisms that can be utilized together with the previously studied research problems to provide for different levels of data protection.

- **RP4: Complex and dynamic access control:** The typical Role-based access control (RBAC) model is static in nature and does not support complex relationships between actors [60, 66, 152]. There is a need to develop models that separate concepts and individuals and enable describing fine-grained access control rules between specific individuals, and not only hierarchical as in RBAC. Therefore, the research problem is about studying the most appropriate usage of ontologies to enable fine-grained access control over relationships that cannot be modeled with RBAC.

1.3 Research Questions and Hypotheses

In the following, we list a set of research questions and hypotheses that we defined according to the above research problems. These questions and hypothesis will contribute in turn to define the thesis goals that follow.

- **RQ-1:** How to provide decentralized security, privacy, and traceability for data storage?

- **H-1:** The joint use of distributed hash table for storage, distributed ledger technology for immutable storage of meta-data, and RBAC can provide a completely decentralized platform.

- **RQ-2:** How to efficiently manage data updates when blockchain is involved in the process?

- **H-2:** Reducing the use of blockchain to handling metadata enables tracking history and decoupling the log of data operations from the data itself.

- **RQ-3:** What is the best way to use distributed ledger to provide decentralized trust for secure data storage?

- **H-3:** A distributed ledger can be designed to efficiently keep immutable records of data operations on the data storage support.

- **RQ-4:** How to protect users' privacy-sensitive information on a decentralized data storage solution?

- **H-4:** It is possible to combine a decentralized data storage framework with RBAC to handle privacy for data access in a decentralized way.

1.4 Thesis Goals

According to the above research questions, this thesis aims at designing and implementing a RESTful decentralized storage framework that features data privacy, security, and mutability, to support traceability for the WSC. We take into consideration different technologies such as distributed ledger technology (in particular blockchain), Distributed Hash Table (DHT), symmetric and asymmetric encryption, and ontology-based access control. We integrate these technologies into a single framework to provide a secure and privacy-aware decentralized solution. We critically examine the methodologies and limitations of existing work to deliver a comparative analysis with respect to our work (H-1). This work has been published in these articles [10, 12, 14, 118].

Within our proposed framework, we design a distributed ledger as a blockchain to store our metadata (immutable record of the data operations) and DHT key (a hash pointer refer to the data in the DHT), while actual encrypted data is managed on the DHT. This design supports data updates and enables participants' trust by storing metadata as a record of immutable data operations on the blockchain. It enables the data owner to control and trace their data, who wants to access it, and which parts of the data are requested. We investigate managing both previous pointer (a hash key of the previous version of the data) and new DHT key in case of data updates, that allows data owners to access their transaction history (H-2, H-3). This work has been published in these articles [10, 14, 118].

We build our solution over the Web to ensure interoperability, so our proposed framework enables authorized participants to manage their data through Representational State Transfer (RESTful) Application Programming Interface (APIs). Our decentralized platform manages multi-level (class and individual level) data access control based on ontological descriptions that support reasoning thanks to the Semantic Web Rule Language (SWRL). Our SWRL rules enable fine-grained data access, and as the framework operates over RESTful APIs, the rules rely on HTTP verbs (GET, POST, PUT, DELETE) to define the permissions that different actors have on the data (H-4). This work has been published in these articles [11, 14].

1.5 Research Methodology

We have structured our research methodology through a systematic approach that consists in identifying the concerned research domains related to decentralized systems (Web framework/API design, trust, data storage, privacy and security), reviewing existing work in the different research domains, first individually, and then exploring works that combine them, to identify the most recent advances and their

limitations. Based on this literature review, partially summarized in [16], we could clearly identify the research contributions developed in this thesis.

1.5.1 Literature Review

To start with the research, we have gathered articles to get a deep understanding of blockchain, distributed hash table, encryption mechanisms, access control, and semantic Web technologies. Our research was carried out by using the following search engines and databases: Google Scholar, Science Direct, ACM Digital Library, IEEE Explore, and Google Search.

We searched research papers relevant to our thesis topic using the following keywords, "blockchain", "distributed ledger", and "decentralized data management". We analyzed the retrieved research papers based on title, abstract, and keywords. Then, we read research papers to gather knowledge and obtain an understanding of blockchain design, structure, and its usage. In the next step of our research, we refined the search phrases by using more keywords such as: "access control", "security", "privacy", "distributed hash table", "encryption" "cryptography" and "semantics" to find out more papers related to our topic. In the next step, we excluded those papers that were already considered or not completely relevant to our thesis topic. Articles resulting from this search are examined individually and we selected the papers that are relevant to our topic.

1.5.2 Critical Evaluation of Existing Methodologies

After getting a thorough understanding of the domain of study, the gaps in the literature were more visible, which helped us to provide a critical comparative analysis and evaluation of existing work with our proposed work. We presented the identified gaps of existing literature in Table 2.1 of Chapter 2, which we summarized in the following.

Most existing data storage solutions are suffering from security, privacy, immutability, and scalability issues. A range of solutions do not take into account privacy-sensitive data and rely on the blockchain for storage without additional encryption techniques. In addition, data stored on the blockchain is immutable which does not allow to make any modification after storing it. As well, it presents a scalability issue as the blockchain is designed to provide decentralized trust and not data storage. While some solutions address data access control, in general they do not address complex permissions to access data and one-to-one relationships between actors. This critical comparative analysis helped us to answer our research questions and build our scientific contributions.

1.6 Scientific Contribution

The results of this dissertation contribute to the research investigating decentralized privacy-aware data storage, in particular the integration aspects of decentralized data storage, data mutability with blockchain technology, data security and privacy,

multi-level ontology-driven data access, and Web client. We explained the following set of contributions of this dissertation:

- We propose a decentralized Web framework that enables data owners to control their data and manage fine-grained data access without a TTP. This framework builds on RESTful APIs to ensure interoperability over the Web. The proposed RESTful APIs are designed in such a way that the different actors of the WSC can query each other's data if they are allowed to, thus opening the way to automate those exchanges through semantic annotations. To enable our framework to operate, we combine blockchain with distributed hash table, ontology-driven access control, and encryption mechanisms into a single framework. This work has been published in the in the Workshop of MADEISD in the European Conference on Advances in Databases and Information Systems in August 2021 [14], SCI Journal of Energies, MDPI in October 2021 [10], and Conference of SWST International Society of Wood Science and Technology in July 2020 [12].

- We propose a decentralized data storage solution. In this solution we have following set of contributions:

  - We demonstrate a solution that combines blockchain and DHT to manage data updates. Our solution stores metadata and DHT key on the blockchain, whereas actual encrypted data is managed on the DHT. Our proposed solution allows actors to keep track of and update their data without disclosing their private information on the public ledger. This result has been published in the SCI Journal of Energies, MDPI in October 2021 [10], Conference of InnoRenew CoE International, in June 2021 [13], and SCI Journal of Applied Sciences, MDPI in January 2022 [118].
  - We provide and develop a decentralized system that supports different types of encryption to ensure data protection at run time. It enables actors to choose between different types of encryption methods while storing and querying data. This security design ensures data authenticity and protects data against spoofing, linking, eavesdropping, modification, and sybil attacks. This result has been published in the SCI Journal of Energies, MDPI in October 2021 [10].
  - We propose a metadata structure that extends the metadata discussed in [7] to enable trust between actors. The proposed metadata structure maintains immutable record of each operation performed on the data. This result has been published in the SCI Journal of Energies, MDPI in October 2021 [10] and SCI Journal of Applied Sciences, MDPI in January 2022 [118].
  - We propose a pointer system to connect the different versions of the data together. One metadata structure contains a pointer to the previous version of the data, as well as the DHT key of the data on the DHT. It allows authorized actors to navigate through different versions of the data in our framework. This result has been published in the SCI Journal of Energies, MDPI in October 2021 [10].

  - We propose a traceability algorithm that allows actors to keep track of their data and verify the origin of the final product in a decentralized ledger, thus showing the applicability of our solution to the scenario that motivates our work. This result is submitted to the SCI Journal of Computer Science and Information Systems in January 2022 [15].

- We propose an ontology-driven role-based access control component to manage relationships between actors and handle complex permissions for data access through REST APIs. This result has been published in the SCI Journal of Energies, MDPI in October 2021 [10] and conference of sustainability in energy and buildings in July 2021 [11].

- We develop a generic HATEOAS client with proxy to navigate through data using REST APIs. Our proxy processes the response message and offers relevant links to the client with the help of REST APIs. We propose a REST API that enables actors to add, update, read, and delete templates with links.

- We provide details of a proof-of-concept implementation with security analysis and performance evaluation to show the feasibility of our solution. We evaluate the performance of our solution in terms of scalability with a proof-of-concept prototype, by implementing time measurements with different numbers of clients. This result has been published in these articles [10, 11, 14, 118].

The results of our dissertation are published in the following papers:

- Aslam, S., Tošic, A., Mrissa, M.: Secure and Privacy-Aware Blockchain Design: Requirements, Challenges and Solutions. *Journal of Cybersecurity and Privacy, 1(1), 164-194, 2021*
- Aslam, S., Bukovszki, B., Mrissa, M.: Decentralized Data Management Privacy-aware Framework for Positive Energy Districts. *Energies, 14(21), 7018, 2021*
- Aslam, S., Mrissa, M.: A RESTful Privacy-Aware and Mutable Decentralized Ledger. *European Conference on Advances in Databases and Information Systems (pp. 193-204). Springer, Cham, 2021*
- Aslam, S., Bukovszki, B., Mrissa, M.: Multi-level Data Access Control in Positive Energy Districts. *Sustainability in Energy and Buildings 2021 (pp. 553-565). Springer, Singapore, 2021*
- Aslam, S., Mrissa, M.: Privacy-aware Distributed Ledger for Product Traceability in Supply Chain Environments. *Conference of SWST International Society of Wood Science and Technology, 2020*
- Aslam, S., Mrissa, M.: Mutable and Privacy-aware Decentralized Ledger for Data Management in Wood Supply Chain Environments. *InnoRenew CoE International Conference (IRIC2021), 2021*
- Mrissa, M., Tošić, A., Hrovatin, N., Aslam, S., Dávid, B., Hajdu, L., Krész, M., Brodnik, A., Kavsek, B., : Privacy-aware and Secure Decentralized Air Quality Monitoring. *Applied Sciences, 12(4), 2147, 2022*

One paper has been submitted and is currently under review:

- Aslam, S., Mrissa, M.: A Framework for Privacy-aware and Secure Decentralized Data Storage. *Computer Science and Information Systems (ComSIS), June 2022*

1.7 Dissertation Outline

The rest of this dissertation is structured as follows:

Chapter 2 provides background knowledge of the main concepts that we use in this thesis. Then, it details the related work of existing technologies that are relevant to our research. After that, it highlights the advantages and limitations of the work identified in the literature review. It presents a feature comparison of the proposed solution with existing solutions.

Chapter 3 describes the design of the proposed decentralized Web framework and its components. We design RESTful APIs of our decentralized Web framework to ensure interoperability over the Web. The use of our framework components is illustrated with our example of the WSC. Then, it details the algorithm that shows actor registration process to the proposed decentralized framework using our REST APIs. It presents the interaction of each actor with the proposed framework using REST APIs. After that, it provides the implementation details and comparison of the proposed REST APIs with the Hyperledger APIs.

Chapter 4 explains the secure decentralized data storage, metadata structure, management of data updates, and traceability contributions. It presents the metadata structure to maintain the actor's trust in a decentralized framework. Then, it explains the algorithm for the data write (without pre-existing data) and flexible encryption design to enforce security on data. After that, it details a data update (with pre-existing data) solution that allows data owners to modify their data and access their update history. It also describes the data read and delete operations on a decentralized ledger. It discuss the proposed traceability algorithm that enables actors to keep track of the data in the chain. Then, it describes implementation details with security analysis and performance evaluation of our solution.

Chapter 5 presents an ontology-based access control solution to ensure data privacy and manage complex relationships between actors. It explains the proposed access control ontology component including ontology classes, object properties and semantic description of non-hierarchical relationships. After that, it describes the access control rules using HTTP verbs to manage authorized access to the data through Web APIs. Then, it details the experimental results of the proposed solution.

Chapter 6 discusses our generic HATEOAS client that navigates through different hyperlinks. After that, it details the proposed proxy as a resource that processes the responses and offers relevant links to the client with the help of our designed REST APIs. Then, it discusses the proposed proxy template management solution to update the proxy with templates and links. It explains the implementation details

and performance evaluation of the proposed solution.

Chapter 7 summarizes the results we obtained during the work related to this dissertation and presents directions for future work.

Chapter 2

Background Knowledge and Literature Review

The Result of this chapter is published in the following article:

- Aslam, S., Tošic, A., Mrissa, M.: Secure and Privacy-Aware Blockchain Design: Requirements, Challenges and Solutions. *Journal of Cybersecurity and Privacy, 1(1), 164-194, 2021*

2.1 Introduction

This chapter aims to provide an understanding of existing technologies and discuss relevant work related to our identified research challenges presented in Chapter 1. First, we provide the background knowledge of relevant technologies that we used throughout this dissertation. We categorize the background knowledge section into the following sub-sections: blockchain technology, Distributed Hash Table (DHT), Role-based Access Control (RBAC), semantic Web, and the Representational State Transfer (REST) architectural style.

Afterward, we provide the literature review of existing solutions that are relevant to our research. We categorize the related review section into the following subsections: decentralized data storage, decentralized user identity privacy, semantic approaches to RBAC, and Hypermedia as the Engine of Application State (HATEOAS) with REST. We highlight the advantages and limitations of existing work. Then, We provide a features comparison of our proposed solution with state of art approaches. Finally, we summarize the outcomes of this chapter and highlight limitations of existing solutions.

2.2 Background Knowledge

In this section, we provide the background knowledge to blockchain technology, DHT, RBAC, Semantic Web, REST Web services, and HATEOAS client because the contribution of this dissertation rely on those.

2.2.1 Blockchain Technology

A blockchain is a decentralized data structure, secured with cryptography mechanisms to ensure the integrity of its transactions [122]. A blockchain is a chain of blocks linked to each other. A block is composed of block header and the transactions. The block header is based on block version, the Merkle tree,timestamp, nonce, and previous block hash. Transactions contain information such as transaction amount, sender, receiver address, etc. and any participant of the network can verify the content of the transactions. Transactions are shared between peers of the network and are stored in a block after a process known as mining.

Concretely, a consensus mechanism such as proof of work is used to validate the transactions in a block and then add validated block to the blockchain [183]. It secures the blockchain against a double-spending attack. Miners obtain rewards such as new coin for their block validation.

To add each block to the chain, the proof of work algorithm needs miners to solve difficult mathematical puzzle that should be approved by all the miners. Once transactions are validated by the miners, a block is inserted to the end of the blockchain network. It helps to prevent an malicious user to take control on the more than half of the hashing power on the network. The process to to verify the proof and its correctness is easy and quick.

In the following, we elaborate the structure of the blockchain: block, Merkle tree, and digital signature.

- **Block:** The verified transaction are recorded in a block. Any peer of the blockchain can initiate a transaction and transfer a copy to all participating nodes on the network. Once participating peers verify the correctness of transactions in the block, then a block is added to the blockchain.

  Each block is comprised of specific information such as transaction time, number of occurred transactions, etc. In a blockchain, each block is connected to its parent block through the hash value of its previous block to avoid the alteration of stored data [155]. If a malicious user tries to modify the block, then a new generated hash of the current block would not match the hash of the next block. This ensures the immutability feature of the blockchain, which does not allow to change the stored data after adding it to the blockchain [92].

  The first block of the blockchain has no previous block and is known as the genesis block. Each block of the blockchain has a unique hash value generated by a hash function such as SHA-256 to make them unique in the blockchain.

- **Merkle Tree:** It is a binary hash tree where nodes are connected to each other via cryptography hash pointers. It stores a blockchain transactions as a tree structure. The hashes of all nodes are grouped in a root node to make the Merkle tree [115], where hash of two child nodes are grouped into its parent node as depicted in Figure 2.1. This procedure is repeated from bottom to top until it reaches the root node of the tree. All transaction are verified by the root node of the merkle tree. Merkle tree enables peers to confirm the integrity and validity of data. If an attacker tries to modify the transaction then it is required to modify all the block hashes.

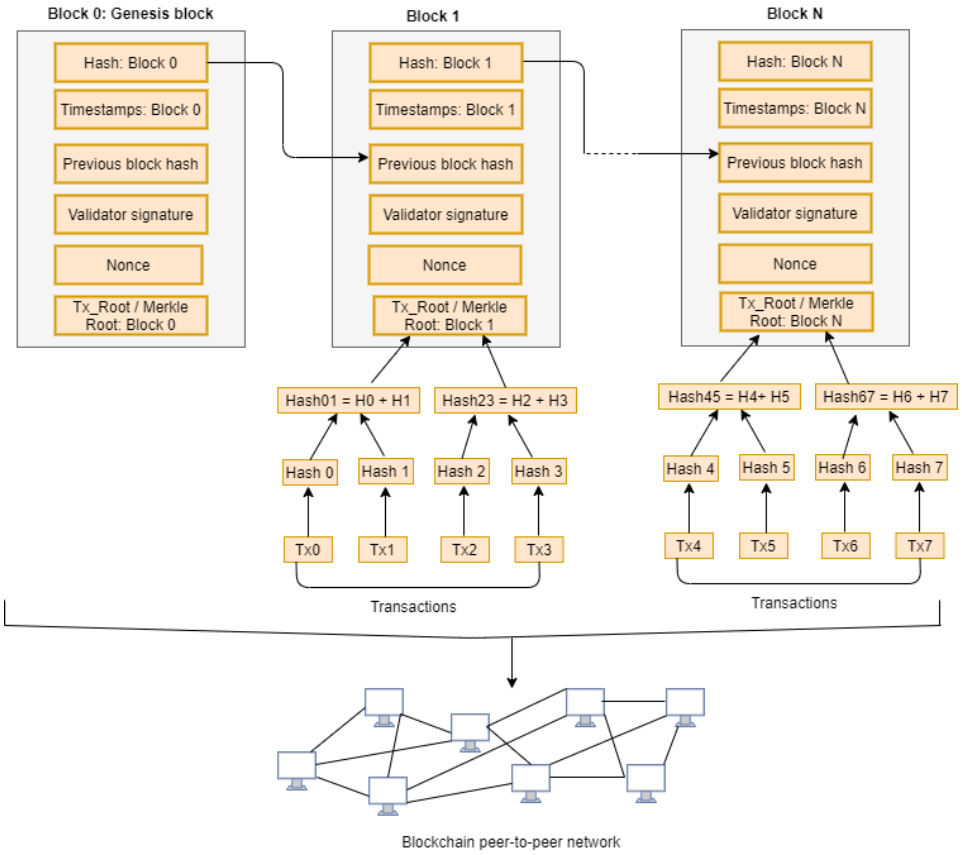

Figure 2.1: Merkle tree of blockchain.

- **Digital Signature:** It is based on three algorithms such as key generation, signing, and verification. Th key generation algorithm is used to creates a key pair such as a private key and public key. The private key is not disclosed to anyone and is used to sign a transaction, while the public key is publicly available and is used to verify the transaction. Then, signing algorithm signs the input transaction using given private key. The verification algorithm takes signature, a transaction, and public key as a input and validates the transaction signature with the help of public key. The digital signature has an advantages in terms of data validity and prevention against non-repudiation so users on the blockchain network cannot decline their own activities.

Development of Blockchain Implementations

Over time, it has been observed that the development of blockchain can be organized into the following three generations [33, 44, 179].

- **First Generation: Digital Currency.** The decentralized digital currency (e.g., bitcoin or digital coin) is the first generation of blockchain that enables actors to make transactions without any intermediate party [33]. The first gen-

eration of blockchain solved major problems to generate decentralized currency, this generation mainly depended on a proof-of-work consensus mechanism.

This generation has an advantage in terms of decentralized storage, enables nodes to share data directly, and ensure transparency during transaction processing [179]. The main limitations are the energy consumption of consensus mechanisms and the fact that proof-of-work provides rewards to the participants who already have the most computation power, which could be a security issue.

- **Second Generation: Smart Contracts.** The second generation of blockchain is a smart contract that enables programs to execute autonomously when specific conditions are fulfilled [160]. The smart contract is stored on a blockchain and cannot be modified. A contract has the following three main parts: a unique contract address to find them on the public ledger, private storage, and amount of balance (e.g, Ethereum the first implementation to provide smart contracts uses the Ether cryptocurrency). The solidity programming language (high-level language) is used to write an Ethereum based smart contract and it can compile into low-level bytecode for the Ethereum virtual machine (EVM) code [30].

  The main advantages of smart contracts are to ensure tamper-proof fraud prevention and minimize verification costs. It is important to highlight that some blockchain implementations do not fully, or at all, implement a smart contract. The main drawbacks of this second generation are performance and scalability, as witnesses, for instance, the low number to process transactions per second [44].

- **Third Generation: Scalability.** The third generation of blockchain improves the scalability issue that highlighted in previous generations. Most disadvantages relate to mining delay, energy consumption, low number of transactions, mostly related to the use of proof-of-work type of consensus and application of smart contracts. We have also examined growth in the number and different applications, for instance, e-Health [56] and supply chain management systems [169]. This generation of blockchain platforms includes Dfinity [71] and NEO [55], which support various programming languages and the development of mobile-based applications [179].

Generally, blockchain has different types depending on data availability and on what actions are permitted to perform on data by the actor [59]. Thus, the available types of blockchains are as follows: public, private and consortium.

- **Public Blockchain:** The public blockchain is also known as a permissionless ledger that is accessible publicly and anyone can view, read and write data on the blockchain [178]. Ethereum and bitcoin are an example of the public blockchain [93].

- **Private Blockchain:** The private blockchain is a permissioned blockchain, which enables only specific actors to verify and insert transaction blocks to

the blockchain [49]. Examples of private blockchain include monax and multichain [93].

- **Consortium Blockchain:** The consortium blockchain is also called federated or public permission blockchain, which enables only a group of organizations to verify and write data to the blockchain. It can be an open ledger or restricted to a particular group. Examples of consortium blockchain include R3 and Corda [133].

2.2.2 Distributed Hash Table

The main idea behind Distributed Hash Table (DHT) is to represent a single logical key-value namespace across multiple computers. To do so, a hash function (such as SHA 256) is applied to the data value to generate a unique identifier called a key. Hash tables store the hash keys and corresponding values. A DHT takes a hash table and distributes it over a large number of peers [113].

Two main methods are used in a DHT to store and retrieve the data. New data can be stored on a DHT node by using the PUT (key, value) method. Whereas, DHT enables participating nodes to get the value associated with a given key using Get (key) method. Each peer in the network knows about a small number of other peers to make a query and exchange data. Nodes store routing tables that keep the identifier of the neighbor's nodes. A requesting node contact with other nodes to find the (key, value) pair in the network. The process to find the data is quick.

DHT is scalable to handle a large amount of data. In addition, it is fault-tolerance because (key, value) pairs are copied on various nodes in the network, which guarantees data availability [185].

Distributed Hash Table Implementations

In the following, we discuss the existing DHT implementations, such as Kademlia [113], Chord [158], Pastry [156] , and Tapestry [181] . Kademlia and Chord are the two main algorithms of the DHT.

- **Kademlia:** The Kademlia [113] is a balanced binary tree representation of the nodes. In Kademlia, each node has an address (e.g IP address, port), and this address is used by other nodes in the network to connect to this node. Data stores as key-value pair on the k nodes which IDs are the nearest to the key. Therefore, in Kademlia, k is used as key parameter to define the data redundancy. Each node contains a routing table, which is a data structure, where each node keeps the information about other nodes to contact them later. Each routing table contains a list of k-buckets, and these buckets are used to store information of other nodes.

  Let us develop an example, node 0 is alive in the network that is known as the bootstrap node. To join the second node e.g node 1, it should know the address of node 0. Node 1 sends a request to node 0 and asks for the list of closest nodes. Node 0 adds node 1 to its routing table and sends a response

to node 1. Then, one more node that is node 2 wants to join the network and sends a request to node 0. Node 0 adds node 2 to its routing table and returns the list of closest nodes that also contains node 1. Now, node 2 has both node 0 and 1 in its routing table. Node 2 sends the ping request to node 1 to make sure it is alive in the network. As node 1 receives the request from node 2, then its add node 2 to its routing table.

Kademlia routing is based on the XOR distance function, which determines the distance between nodes. This function has symmetric property, which means that the distance from 0 to 1 and 1 to 0 are the same. In addition, the symmetric feature allows both movements such as clockwise and counterclockwise. Kademlia has an advantage to prevent loss of data, as if a node fails or shutdown then data is still accessible from the closest nodes. Noticeably, the main disadvantage of Kademlia is also that necessary data replication on nearest nodes.

- **Chord:** The Chord is an algorithm of DHT introduced by Stoica et al. in 2003 [158]. It forms a logical ring of circles, and each of the different circle is a node in the network. The key idea behind the Chord is that it is the closest successor node that stores the data and ID. To retrieve the data, it makes a query to its successor node. It is based on consistent hashing, in which a flat keyspace map between the nodes and data. The node ID can be the IP address of the machine or a unique number called key generated by the hash function. It forms a logical ring with finger tables to reduce the access time.

  Each node in the network contains the finger table that allows them to access data through directly connected edges between nodes instead of going through all nodes one by one in the circle. The Chord has advantages in terms of distribution, scalability, and load balancing. When a node joins or leaves the network, only a few keys need to be moved from one node to another. However, it is static and needs to continuously update finger tables, keys, and successors.

- **Pastry:** The Pastry [156] is similar to Chord, the main aim is to generate completely centered, structured Peer to peer networks, and efficient message routing. In Pastry, nodes are arranged in ascending order in a ring. Node IDs can be a public key or IP address that is generated by the cryptography hash function. Pastry identifier space is different from than Chord ring, as its routing is based on the numerical identifier. The basic idea of routing is, it goes through the nodes in the ring and finds that node ID which is numerically closest to the hash value of the given key. The identified node is responsible to store the key, value pair of the given key. The advantage of Pastry is scalable. However, it is time-consuming to search the node IDs in the ring.

- **Tapestry:** Like other DHT algorithms, nodes in Tapestry also have their IDs such as IP addresses generated using a hash function. In Tapestry [181], a root node stores the references or identifies the nodes that have an object. A node

can be chosen as a root node if it shares the leftmost prefix numbers with the hash value of the object. Each node contains a routing table that has reference to the subset of nodes, which enable the node to search object stored by other nodes. The Tapestry network is fault tolerant to some extent, however its fault tolerance is limited due to its structure. The main disadvantage of the Tapestry network is loss of root node data as root node shut down, which is mitigated with redundancy on the network.

2.2.3 Role-based Access Control

Access control is used to restrict unauthorized access to the data. It defines which data a user would access, and if the user has access to the data then what type of access is. Access control maintains data confidentiality by ensuring that only data is disclosed to the authorized user.

The Role-based Access Control (RBAC) model was introduced to manage and enforce security in large-scale systems. It simplifies security management because it allows users to access data according to their role within an organization.

The basic RBAC is comprised of user, role, permission, and object. In RBAC, a user is a person that accesses the system. Roles are generated to perform specific jobs within a system or organization such as woodcutter. Each user can have more than one role. A Permission is an authorization to access an object within the system [24]. Each role can have more than one permission at the same time. In RBAC, permissions are given to the roles, and then roles are assigned to the users. The objects are resources that can be accessed by role according to their given permissions.

The RBAC is an alternative to discretionary and mandatory access control models and ensures that only authorized users have permission to access data or resources. In particular, it allows pre-determine the role-permission relationships, which makes it simple to associate users to the pre-determined roles.

RBAC is based on the following three basic rules, which include role assignment, role authorization, role execution.

- **Role Assignment:** A user can perform a task only if the role is assigned to the user.
- **Role Authorization:** A user can have only those roles for which they are authorized.
- **Role Execution:** A user can only have a permission to execute a task if the permission is assigned to the role.

Without RBAC, it is hard and time-consuming to define what permissions should be given to which users. With RBAC, it is easy to change a user's permissions without making any modification in the access structure by simply inserting or removing actors to the roles [108]. The main advantage of RBAC is to prevent data leakage because only authorized users can access the data.

The RBAC model contains the following components, such as hierarchical RBAC, static separation of duty, and dynamic separation of duty.

- **Hierarchical RBAC:** It is an extension of the basic RBAC model that supports role hierarchy [147]. It is a tree representation of roles where the senior role is a root node and juniors roles are at the bottom. The hierarchical RBAC defines the inheritance relationship between roles.

  For instance, the senior role would have all permissions of their junior role in the role hierarchy. The main advantage of hierarchical RBAC is its flexible structure and it is easy to understand. However, it does not support horizontal relationships.

- **Static Separation of Duty:** The static separation of duty is also known as strong exclusive which means that one user is only authorized for one role at a time [150]. In other words, two roles would not have any shared principle. Static separation of duty enforces constraint, which controls the role membership to a single user. The main advantage of static separation of duty is simplicity. However, it is not practical in real organizations because one user may need to have two roles at the same time [154].

- **Dynamic Separation of Duty:** The dynamic separation of duty is also known as weak exclusive which means that one user is allowed to have two or more roles but both roles could not be active at the same time [150]. If one role has the same permission as the other role then it can become a conflicting role. In this case, the system would maintain the record of each action that has been performed. The dynamic separation of duty is flexible and practical for organizations. However, it is a time-consuming process to assign the non-conflicting permissions of junior roles to the senior roles [67].

2.2.4 Semantic Web

The concept of the semantic Web was proposed by Berners-Lee et al.in 2001 [23]. The semantic Web is an extension of the World Wide Web (WWW). The aim of the semantic Web is to describe the meaning of information on the Web and to make it accessible by both human users and machines. In fact, both the semantic Web and Web services are relying on Web resources, which are identified through URIs.

In the following, we discuss the main layers of semantic Web architecture [161].

- **URI and Unicode:** The first layer of the semantic Web is uniform Resource identifiers (URIs) and Unicode. The URIs are used to identify resources on the Web. It assigns a unique name to each resource that differentiates one resource from others. Unicode is the standard code language on the Web and it enables all user's languages can be read and written over the Web in one standardized form.

- **Extensible Markup Language (XML):** It is a second layer of the semantic Web. The XML is a markup language that aims to represent structured documents. It provides a flexible data format instead of vocabulary. It contains

a set of tags and allows user to write their own tags. It is machine-readable and allows to exchange of several types of data on the Web. Each document contains the XML namespace that assigns a unique name to each element and attribute in an XML document.

- **The Resource Description Framework (RDF):** The third layer is RDF which is used to link and represent resources over the Web. It is based on URIs to identify Web resources and represents the relations between these resources, through triples such as subject, object, property. These triples are modeled in a directed graph, where resources are nodes and connected through the property.

- **RDF Schema (RDF-S):** As part of the third layer RDF-S provides the possibility to declare classes that allow to group RDF statements and structure data. RDF-S allows therefore developers to check conformity of data against a RDF-S schema. Outside of class declaration, RDF-S remains on purpose very limited to describing the basic structure of data.

- **Web Ontology Language (OWL):** The Web Ontology Language (OWL) is a language to represent domain knowledge in ontologies. An ontology is useful to describe and share domain knowledge between organizations and people [91]. It defines the concepts, properties and relationships between classes and concepts [28]. It is comprised of classes, objects, data type properties, individuals, etc. In an ontology, the parent class is known as a super-class and other child classes are called subclasses. It enables users to check the consistency of data and to use a reasoner to infer new implicit facts from explicit existing ones.

  An ontology can be constructed using different languages such as RDF and OWL. OWL is the knowledge representation language of the semantic Web that defines the meaning of data in vocabularies and the relationships between the data elements. OWL is declined in different subsets (OWL Lite, OWL DL, and OWL Full), that we discuss in the following.

  - **OWL Lite:** It is a least-expressive OWL subset and extension of RDF. It supports expressing a simple hierarchy and few constraints. It supports fixed cardinality values such as 0 or 1. It does allow to define classes as an instance of other classes. It supports only the `IntersectionOf` class description.

  - **OWL DL:** It has more expressiveness as compared to OWL Lite. It is based on the Description Logic (DL) which is used to reason the domain knowledge. It is flexible to define cardinality and max-cardinality values equal to or greater than 0. In OWL DL, a class, instance, and property name must be distinguished to each other for better compatibility with reasoners. It allows UnionOf, ComplementOf, and IntersectionOf class descriptions. It has an advantage in terms of reasoning support. In addition, OWL DL is compatible with OWL 2[1] because both guarantees to

[1] `https://www.w3.org/TR/2012/REC-owl2-profiles-20121211/`

complete computations within a time.

- **OWL Full:** It is the most expressive OWL language. It supports UnionOf, ComplementOf, IntersectionOf class descriptions. It also allows using cardinality and max-cardinality values as 0 or greater than 0. It does not have restrictions to use the different class and instance names and facilitates the integration of RDF(S). A class can be similar to an instance. However, a decidable reasoning procedure might not exist for some OWL Full ontologies, thus making it difficult to utilize [9].

- **Logic and Proof:** The logic layer allows to write the rules wheres the proof layer enables execution of the rules. A reasoner is used to identify and resolve inconsistency and the duplication of the concepts.

- **Trust:** The final layer of the semantic Web is trust. It depends on the rules available in the data which can avoid unauthorized users to access this data.

Rule Languages

Knowledge representation languages (e.g RDF and OWL) are used to describe the application domain. Typically, they define concepts, properties, individuals, and relationships. In contrast, rule languages are used to define data transformation rules that are able to generate new facts from the existing data.

In the following, we discuss the most famous rule languages such as Datalog, the Rule Markup Language (RuleML), the Semantic Web Rule Language (SWRL), and TRIPLE.

- **Datalog:** It is a simple form of logic programming in which function symbols are not used [38]. It is a query-based rule language for a relational database. Logic programming is based on facts and rules. A fact is a declarative statement such as `Bob is the father of Alice`. A rule infers facts from the knowledge base. For instance, If A is the father of B and if B is the father of C, then A is the grandfather of C.

  In Datalog, both facts and rules are described as Horn clauses [151]. The fact discussed above can be written in the Datalog as follows:

  mother (Tina, Alice)

  The rule represented above can be written in Datalog as follows:

  grandmother(C, A) : - mother(B, A), mother(C, B)

  Here, mother and grandmother are known as a predicate, Tina and Alice are constants, and A, B, and C are variables. In the rule discussed above, the left Datalog clause is known as its head and the right side is known as its body. The main advantage of Datalog is its simple syntax which is easy to understand and learn. However, Datalog is limited to domain-specific descriptions.

- **RuleML:** It is a semantic Web language. It defines the rules in XML syntax for Datalog [114]. It also allows both top-down and bottom-up rules in XML. Rules are defined within open and closing tags such as <implies>, <head>, <body>. In RuleML, <rel>, <var>, and <ind> are written within the open and closing tags know as Atom.

  Let us develop an example, Datalog clause such as JPaper (Security): - Paper(Security, Journal, IEEE) can be written in the RuleML as follows: todo make shorter

```
<Implies>
 <head>
 <Atom>
    <Rel>Jpaper</Rel>
     <Var>Security</Var>
 </Atom>
 </head>
 <body>
   <Atom>
     <Rel>Paper</Rel>
     <Var>Security</Var>
     <Var>Journal</Var>
     <Ind>IEEE</Ind>
    </Atom>
 </body>
 </Implies>
```

  Here, <Rel> tag defines the predicate symbol, <Ind> tag defines as constant, and <Var> tag indicate variable. RuleML is easy to understand. However, it is complex to manage nested tags for large documents.

- **SWRL:** It stands for Semantic Web Rule Language that follows the OWL syntax to write the rules [20]. It infers new facts from the knowledge base. It is comprised of the following two parts such as antecedent and consequent.

$$\text{antecedent} \rightarrow \text{consequent}$$

  In SWRL, the consequent part depends on the antecedent part, such as, if atoms in the antecedent are positive, then the consequent will also be positive.

  In the following, we present the example of the SWRL rule that shows the inference such as a person with a female sibling has a sister.

$$\text{Person(?a)} \wedge \text{hasSibling(?a, ?b)} \wedge \text{Female(?b)} \rightarrow \text{hasSister(?a, ?b)}$$

  It is different from RuleML because it works with both unary atoms such as Person(a) and binary atoms such as hasSibling (a, b). In unary atom Person(a),

a is an instance of OWL class or data range. Similarly, in atom Female (b), b is an OWL class instance or data range. While, in binary atoms hasSibling (a, b), a is linked with the b through property hasSibling. An atom hasSister (a, b), where a is linked with b through property hasSister.

SWRL rule has an advantage in terms of flexibility because it allows using a class name in rules. In addition, it accepts arithmetic operators. However, in SWR rules, users need to follow the proper syntax to avoid errors.

- **TRIPLE:** It is based on namespaces, RDF statements, and first-order logic rules. A namespace contains names that are identified using a URI reference. An RDF statement is based on the Triple such as (s, p, o) where, s is a subject, p is a predicate, and o indicates an object. A predicate defines the relationship between subject and object. Let us consider an example, a female hasSibling sister. In this example, female indicates the subject, hasSibling indicates predicate, and sister indicates an object. Here, the female is linked to her sister using hasSibling property. TRIPLE rules are easy to understand. However, it is difficult to infer the fact if the statement is not written in the form of triple such as (s, p, o).

2.2.5 The REST Architectural Style

REST is an architecture style for distributed hypermedia systems introduced by Fielding in 2000 [61], that allows distributed components to communicate with each other through a network. REST offers several advantages: it is data format agnostic (although JSON and XML are nowadays the most commonly adopted data formats, it can rely on the HTTP content negotiation mechanism to potentially support any data format), it enables loose coupling between clients and servers through uniform interface, it support caching techniques and in general offers more scalability as the resource-oriented paradigm separates the different resources of applications into distinct elements behind different URIs.

To gain the most benefits of REST, some constraints need to be respected, as explained in the following:

Client-Server: A client can request services exposed by the server. The principle behind the client-server architecture is the separation of concerns where the same machine can act both as a client and as a server. The advantage of client-server separation is that both can act independently. The client does not need to know anything about the data which is used to process the request. Similarly, the server does not need to know anything about the user interface.

Stateless: The communication between client and server is stateless. The server would not store any session related to the client data in main memory. This means that everything that server needs to understand about the particular resource has to be contained within the particular request. The server would only understand the request and send a response back to the client. While the client would store and provide the information regarding the current state of the session. This constraint improves the scalability of the system because the server is not required to store

any state between requests. For example, when various servers are offering the same functionality, if one of the server fails, then any other server is able to process the client's request. One disadvantage of this constraint is that it can increase network usage because all the information needs to be included in a request to the server. It means that the server has no control over the system, so the server is depending on the right implementation of the client.

Cache: The cache constraints state that when a server sends a response to the client, so, in its response, it should indicate whether the response can be cached or not. If the response is cacheable, then the client can reuse that data for further requests and does not need to send the request again to fetch the same data which improves the network efficiency and optimization.

Uniform Interface: The uniform interface constraint is a key constraint to differentiate REST APIs from non-REST APIs. This constraint allows any client to communicate with any server using the same limited set of possible actions. The concept of uniform interface has been successful in operating systems where every device is accessed as a file with a limited set of low-level functions (open, read, write, close). The uniform interface constraint brings this principle to the Web where everything is accessible as a resource with a limited set of functions (HTTP verbs: GET, POST, PUT, DELETE...). In the following, we discuss the four key elements or sub-constraints of the uniform interface.

- **Identification of Resources:** Each resource has its unique global identifier known as a Uniform Resource Identifier (URI). This facilitates the modularity of applications, better readybility of code, and easier maintenance through the applicaiton lifecycle.

- **Manipulation of Resources through Representations:** When a client sends a request to the server, they can receive different representations of a resource such as HTML, JSON, and XML response. According to these representations and metadata a client can also delete or update the resource.

- **Self-descriptive Messages:** When a client is sending any request to the server, they need to include all the details (e.g resource URI, context, content, preferred format) in their request so the server would completely understand the nature of the request. To do this, messages or requests should be self-descriptive. Self-descriptive messages are needed to support the statelessness constraint of REST.

- **HATEOAS:** This acronym stands for Hypermedia As The Engine Of Application State. It is one of the constraints of REST that follows the principle of Web browsing to drive the interactions between servers and clients. According to the HATEOAS principle, where users are going from one Web page to another Web page through hypermedia links [128], each response message coming from a Web resource must contain links that enable the client to perform the next possible actions. HATEOAS makes APIs self-descriptive, in such way that the client does not need to know anything about an API before using it. HATEOAS enables APIs to evolve independently of consuming applications

without breaking the client. Indeed, in case of changes, only the links provided in the response message will change, thus enabling loosely-coupled interactions between clients and servers.

Layered System: The layered system constraint is that the architecture is need to be composed of multiple layers, where each layer is responsible for the specific function and only interacts with the layer next to it. So, each layer does not know anything about the rest of the layers other than the one next to it. It limits the complexity of the system because layers are completely decoupled. One disadvantage of this constraint is latency because, for the particular request, the request has to travel through several different layers to generate a response.

Code-On-Demand: This is a final constraint of the REST. Usually, the client would extend their functionality by downloading the data from the server. The aim is to simplify clients as implementation work on the client is minimized. This is an optional constraint of the REST because it increases the system complexity.

2.3 Literature Review

In this section, we review the existing work on decentralized data storage including data security, decentralized user's identity privacy, access control models to control access to the data, and HATEOAS with REST to navigate through the data. We summarize the advantages and limitations of related work in Table 2.1. We also provide a features comparison of our proposed solution with state of art approaches in Table 2.2.

2.3.1 Decentralized Data Storage

We discuss existing literature on blockchain, DHT, and other data storage approaches. The authors in [185], present the decentralized personal data management framework. The proposed framework uses DHT to store encrypted data (with symmetric key) and corresponding hash key is sent to the blockchain. However, this solution uses only one type of encryption, as symmetric key for both encryption and decryption. Additionally, they did not consider symmetric key protection for data access or the question of the key storage location. They used fine-grained access control to manage access to blockchain. However, access control permissions are recorded on blockchain, thus making permission immutability a concern.

The authors in [152] present a blockchain-based data management and access control framework. The proposed framework stores hash pointer and fine-grained access control permissions on the blockchain, while actual data is stored on the DHT. The main limitation of this work is the public visibility of permissions on the blockchain, which is subject to privacy concerns. In addition, access control permissions stored on the blockchain is immutable and could not change later.

A blockchain-based data storage for Ping-ER (Ping End-to-End Reporting) is presented in [7]. A permissioned blockchain is used to store metadata of Ping-ER files while original data references are recorded on DHT. Additionally, this solution stores monitoring agent names and file locations on the blockchain, which is immutable

and shared with other users on the network. In addition, they did not manage the previous pointer in case of data updates.

The authors in [72], presents a blockchain-based Light Chain framework that replicates transactions and blocks within the nodes of DHT. The proposed framework enables all nodes to access transactions and blocks through a skip graph. However, transactions and blocks are publicly available, which raises data security and privacy issues.

A blockchain-based software connector framework is presented to share data between stakeholders of supply chain and companies [103]. A MySQL database is used to store the original data, while a hash sum of this data is sent to the blockchain. However, a MySQL database is centralized and forms a single point of failure due to its design. It is therefore not scalable as a DHT to deal with large number of data and users [87].

The authors in [40] present a blockchain-based framework called u-share, that maintains data traceability. The proposed framework enables data owners to share their data with their circle of friends and family. A software client is used to encrypt the shared data using public key of the circle, distribute the private keys to the circle members, and keep the record of shared keys. However, sharing private key is subject to security problems. Key management is also raising security and performance issues because of the blockchain size increase with circles and family. Additionally, the u-share framework is based on one type of encryption only and does not offer much flexibility for data protection.

The authors in [175], present a decentralized supply chain system to maintain traceability of goods and recipe ingredients. A smart contract is used to exchange the information of goods on a distributed ledger. However, product data is publicly accessible and immutable, thus making privacy and data modification issues.

A blockchain-based agri-food supply chain traceability framework is proposed in [163]. The proposed framework in based on RFID (Radio-Frequency Identification) technology to automatically detect objects using radio frequency signals. The main limitation of this work is that the data is stored directly on the blockchain, which is a problem in terms of scalability.

The authors in [97] proposed a blockchain-based access control framework called CapChain. The proposed framework enables users to share their access rights to IoT devices. A blockchain is used to store the access rights that are encrypted using a secret key. However, this work shares secret keys between owners and devices, which raises security issues.

The authors in [76] present a blockchain-based food traceability framework. In the proposed framework, hash pointers are stored on the blockchain whereas corresponding data are managed on off-chain storage called IPFS (InterPlanetary File System). IPFS is a peer-to-peer network to store and share data in a distributed file system [109]. However, this solution in [76] is based on a manufacturer node server to manage all components of the framework, which leads to a single point of failure.

In [182], the authors use a a IOTA Tangle and IPFS to share Internet of Things (IoT) data. In the proposed framework, a centralized data handling unit (local server) is used to fetch the data and then encrypt it with asymmetric encryption, which is subject to a single point of failure problem. The IOTA Tangle is used to

manage the hash pointer and metadata, whereas corresponding encrypted data is stored on the IPFS. However, the IPFS network does not allow to update file and its content once it has been stored, because of its immutability nature [98]. Additionally, this work did not explain the content of metadata.

The authors in [22] present a distributed cloud storage system known as Storj. It is a trust-based storage system among host and client, where user offer their free storage hardware space and earn money. Data is encrypted at client side (using AES256-CTR) before uploading it on the network. Storj enables the data owner to control and access their data on the network. The main limitation of Storj is high cost to store the data and it is relying on a centralized architecture to conclude storage data and payments [46, 74]. Also, Storj uses one type of encryption mechanism to maintain trust between client and host [176].

In paper [139] authors present a decentralized data storage framework that integrates Solid Pods with blockchain. Solid (Social Linked Data) is based on RDF and semantic Web to handle data. Solid allows users to store their personal data in Pods (Personal online data stores) hosted at the location where users want. The proposed work considers the following two cases to guarantee data confidentiality. First, they use blockchain to manage file hash whereas data is stored on the Solid Pods. Second, smart contracts are used to manage the data on a Blockchain while software wallets (public and private key pair) are stored on the solid pods. Users can access their data through the software wallet. However, Solid Pods itself does not ensure data verification and trust [51]. In addition, it could not handle large amounts of data as DHT does [116].

In [1], the authors propose a decentralized edge-computing framework to manage IoT data. The proposed framework used blockchain to store data and ensures data security. However, data stored on the blockchain is immutable and cannot be modified later.

The authors in the paper [3] discuss the cloud-based personal data storage framework called mydex. It stores encrypted data by default and only the metadata is visible to the service provided that includes data type of the profile (such as a history of credit card transactions, phone calls, bank transactions) and associated permission. The personal data can be shared with third parties such as organizations after signing the contract with mydex. The contract contains the conditions to share the data. However, mydex is a centralized personal data storage that is managed by the owner of the company. Also, it is not free of cost, and it charges a fee to offer its services.

The authors in [165] presented a framework for on-chain data management for ONTOCHAIN called graph chain. The proposed framework provides collaboration between ontology and blockchain. Data is stored in the semantic format, and it can be used through a smart contract. However, graph chain does not encrypt the data before storing it, which raises security and data leakage issues.

Similarly, another work based on the graph chain framework is presented to manage semantic data on the blockchain called onto space [157]. ONTOCHAIN is an ecosystem of different blockchains that are connected to the main chain of the system called onto space. An Ethereum smart contract is used to store the semantic data objects and enable the user to access the graph data. However, this work did not

discuss any solution to manage unauthorized access to the data. Also, all blockchains and graph chains are managed by the centralized mainchain component, which is a single point of failure. In addition, It is immutable and does not allow to update and delete the stored graph data.

Furthermore, DECODE (Decentralized Citizens Owned Data Ecosystem) enables users to keep control of their personal data [35]. DECODE is combined with blockchain technology that enables the data owner to keep track of their data accessed and used. A smart contract is used to write the rules to manage data sharing and allows users to decide on which personal data they would like to share publicly. DECODE enables public users to contribute, access, and use the data shared by the data owner. They combine Ontology with DECODE to allow modification of parameters. However, a smart contract is immutable and does not allow updating rules once it has been written. Also, this work did not provide any specific solution to prevent attacks such as Linking attack, modification attack, and eavesdropping attack.

The authors in [2], propose a decentralized data management and access control framework. The proposed framework used a smart contract to manage access control rules to access the IoT data. Different storage services such as cloud and MQTT are used to store the data. They used oracles as an interface between blockchain, data hosts, and users to ensure trust. The main limitation of the smart contract is immutability that stores access control rules permanently.

Another framework is introduced to manage personal data and record data processing activities (e.g data create, use, share) called ROPA (Record of Personal Data Processing Activity) [143]. It enables any user, company, or public organization to process personal data according to the rules enforced by the GDPR (General Data Protection Regulation). ROPA is based on ontology and blockchain to ensure trust and easy access. SPARQL queries are used to query the ROPA. However, it does not fulfil the security aspects and prevents attacks.

In [186], the authors present an Enigma: a decentralized framework that allows users to control their personal data. A DHT is used to store the data while access control rules and identities are managed on the blockchain. However, access control rules stored on the blockchain are immutable and publicly available, which leads to privacy issues.

BOWLER (Blockchain-Oriented Warehouse & Low-Code Engine and Reasoner) framework is introduced that allows developers to tackle the smart contract modeling complexity by using a low code environment [105]. The Web IDE (integrated development environment) to make the development of smart contracts easy and low cost [75] However, the content written on the smart contract is publicly accessible and immutable, which raises data privacy and modification issues.

Furthermore, decentralized CARECHAIN (Supporting CARE through micro-insurance using blockchains) is proposed to create a platform to execute a smart contract [162]. The proposed project is based on the blockchain and tangle technologies that use smart contracts to manage micro-insurance. Blockchain is used to store the encrypted transactions and run smart contracts automatically once certain conditions are met and read/verified by the oracle. CARECHAIN ensures security and transparency while managing smart contracts and micro-insurance. However,

the proposed solution stores transactions on the blockchain that is subject to immutability concerns.

The authors in [63] proposed a decentralized application called Copyright. The proposed application is based on the blockchain and ontology to ensure copyright management of social media such as YouTube. It allows video owner to store their content off-chain and generate a hash of this content. The authorship claim of that hash will be added to the on-chain transaction. The hash is also stored in the video explanation. Reuser can use the hash to check the authorship claim, reuse conditions, and register agreement on-chain. It enables users to reuse the videos uploaded on YouTube. Ontology is used to model and reason about the reuse conditions. However, the hash stored on-chain is publicly accessible, which leaves information about original content and data owner, thus making data security concerns.

Furthermore, In [94] DESMO-LD (Decentralized Smart Oracles for Trusted Linked Data) is presented. Ethereum smart contract is used to collect and manage off-chain LD (Linked Data) information using decentralized smart oracles. End-user is allowed to read the data using SPARQL queries. However, this work did not encrypt the data before storing it, which is vulnerable to security breaches and attacks.

Another project, DART (Distributed-Oracles Framework for Privacy-Preserving Data Traceability) is proposed to manage data storage and traceability on the blockchain [43]. In the proposed project, users' data will be stored on the smart contract and managed on the blockchain through the oracles. The proposed framework allows end user to receive the data from the blockchain and maintain traceability to verify the original content of the data. However, data stored on the smart contract is publicly visible, which raises data security issues.

In [18] presents a DKG (Decentralized and Scalable Knowledge Graph Economy Tools) to support the trusted, traceable, and transparent ontological knowledge on the blockchain. The proposed work used blockchain to discover and analyze data and data sources indexed are managed using knowledge graphs such as ontology. It ensures data integrity and privacy. However, permissionless Ethereum blockchain is used to ensure data immutability, which stores data permanently.

In [125] proposed a DW-marking (Data Watermarking) to ensure the off-chain data ownership for data trade in the distributed data marketplace. A data marketplace, in which data sellers make sure the availability of the data to buy by the data buyer [123]. Oracle is used to enable off-chain data marketplace to store transactions on the blockchain. However, this work did not encrypt the data before storing it, thus making data privacy and attacks risks.

Furthermore, Gimly ID is introduced that is based on the set of software applications (such as mobile and Web) to manage the self-sovereign identity and secure data sharing [37]. It allows developers to create and verify the decentralized identifiers (DID) and Verifiable Credentials (VC) into their systems. A DID is a unique identifier of the user and associated with a DID document, which maintains the information about an identity (such as creation time, public keys linked to the identity). The DID document is immutable, owned, and managed by the DID owner. VCs are used to store and manage machine-readable credentials such as name, age, date of birth. A VC is linked to identity [32].

Gimly ID enables users to securely share and verify identity and credentials. It enables a company to manage the sovereign of the data and permissions through ID management Web portal. A user can manage their sovereign identity by creating a Gimly ID account in the mobile app and storing their personal data Gimly ID app. A user can share their personal data with companies and then companies can choose a user as an employee and assign them credentials such as role and access authorizations. A user can access company services through the Gimly ID app. However, this work mentions personal data sharing or disclosure such as name, age, date of birth, which is a major privacy risk that could lead to data leakage issues. This work also does not allow the data owner to delete their information.

In [111, 140] a blockchain-based identity system called HIBI (Human Identity Blockchain Initiative) is presented. HIBI is used to manage users' identities on the blockchain and ensure trust to generate knowledge in the semantic Web by linking electronic identification and trust services (eIDAS) to identities. An eIDAS is used to verify the identity of the user and authenticate the electronic document [52]. The proposed HIBI is based on smart distributed key recovery (key management tool) which maps the blockchain keys to eIDAS identities to maintain the backup and recovery. Blockchain transactions are signed with eID and authenticate via eIDAS. Users can use an eID app such as "AusweisApp2" that allows to access the European eID servers to obtain data from it. The proposed work ensures information provenance. The major disadvantage of the proposed solution is blockchain's immutability.

The authors in [84] proposed the ISLAND (Interlinked Semantically enriched Blockchain Data) framework which allows users to gain access to unstructured data from the blockchain, annotate with metadata using ontology, and represent as RDF (Resource Description Framework). A smart contract is used to process the transactions on the blockchain and to monitor the ISLAND framework, while RDF graphs are stored on the off-chain storage called IPFS. This framework allows users to query the data using SPARQL queries. However, this framework did not discuss any solution to protect data from unauthorized access, which rises data privacy and security issues that could lead to data leakage. In addition, smart contracts and IPFS network are immutable and store data permanently.

In [80] proposed an NFTWATCH framework to collect, store, analyze the non-fungible token (NFT) information. NFT is a digital token that is permanently linked with the piece of data stored on the blockchain. It ensures the piece of data ownership. An RDF ontology is used to represent the NFT information and enable users to query the domain knowledge. However, this work did not discuss any solution to prevent data from unauthorized access.

In [21] presented a POC4COMMERCE (Practical ONTOCHAIN for Commerce) that facilitates a semantic representation of ONTOCHAIN stakeholders and their activities. The proposed work enables providers to store products and services in the Ethereum blockchain. End-users are allowed to search for products and services using the search engine. The smart contract is used to create and exchange the digital token of the user. However, this work did not discuss any solution to prevent attacks.

The authors in [95, 104] integrate the Data Agreement (DA) protocol with data

provenance to keep track of the data origin. A DA is a set of rules for an organization or company to process personal data according to privacy law (such as GDPR). Data is stored on an ONTOCHAIN. A smart contract is used to exchange personal data between organizations and enforce GDPR requirements. The proposed work ensures trust and transparency while data sharing. However, the proposed work did not discuss a solution to enforce security on data and prevent attacks.

The authors in [83,110] propose a provenance-aware decentralized reputation system to ensure the trustworthiness of off-chain services (e.g third-party Web services). The proposed framework is designed to gather, process, and store users' feedback. A blockchain is used to store the reputation data of off-chain services through the execution of smart contract, while reputation metadata is stored and managed on off-chain storage called IPFS. However, data stored on the blockchain is publicly visible, which raises data privacy issues. In addition, blockchain is not scalable to store a large amount of data.

In [50, 132] authors present a decentralized SEIP (Service for Encrypted Information Provider) framework to ensure confidentiality while data exchange. The proposed framework combined symmetric key cryptography with attribute-based encryption to manage data access and enforce security on data. Attribute-based encryption encrypts the data according to the access policies defined by the user and the data requester must meet those conditions to decrypt the data [65]. The proposed SEIP framework allows users to request an attribute-based encryption private key by giving a set of verifiable credentials. The key will provide them with access policies and data will be decrypted once conditions are satisfied. However, the work mentions private key sharing, which is a major security risk that could lead to key leakage issues. In addition, this work did not support data updates.

The authors in [96] present a framework called UniProDaPI (Universal Proven Data & Process Interchange). The proposed framework allows authorized users to access the user's metadata stored on the sidechain, while actual data is stored on the blockchain. It enables users to trace their product (e.g parcel, medicine) through ID. However, this work did not provide any solution to prevent data from attacks.

The authors in [101], propose a decentralized framework to store IoT data. The proposed framework used DHT to store the IoT data while the address of the corresponding data is managed on the blockchain. To ensure privacy, they encrypted the data with the IoT device's public key before storing it. They used certificateless cryptography to authenticate the system. However, this solution does not consider transaction history support, spoofing, and eavesdropping attacks aspects.

The authors in [153], propose a blockchain-based framework that allows users to share their data with other users. The proposed framework used smart contract to manage data access control policies. Multi-chain as off-chain storage is used to store the users' private data. However, policies recorded on the smart contract are immutable.

In summary, existing data management and storage solutions are subject to security, privacy, immutability, pointer management, and scalability issues. Most existing solutions store data publicly which leads to data loss and privacy leakage or are not flexible with the encryption solutions they offer. Although, existing solutions are not scalable to manage a large amount of data. In addition, data mutability and data

privacy management are big challenges for blockchain. Thus, motivating research towards decentralizing data storage, data mutability, managing access to privacy-sensitive data, enforcing data security, and multiple types of encryption in a single solution is a challenging task.

2.3.2 Decentralized Identity Privacy

In this section, we explore the most relevant existing work on decentralized user's identity privacy management. In [88], the authors propose a blockchain-based distributed framework for anonymized trading of datasets. The proposed work used blockchain to store all transactions of anonymized data. The main limitation of this work is a third party that deals with privacy policy management.

Another work presented a blockchain-based dynamic identity management framework that also depends on a third party [17]. The proposed framework guarantees the confidentiality of the user's data and enables a user to control their identities in a public network.

The authors in [29] used a zero-knowledge mechanism and smart contract to ensure identity management in the blockchain. However, this work could not deal with malicious verifiers and thus other cryptographic mechanisms need to be combined with it to attain better security.

In [54] the authors discussed privacy issues in blockchain-based IoT. The proposed work used ring signature to ensure anonymity on the healthcare blockchain. The main limitation of this work is a denial of service and modification attacks, where a attacker can prevent a user to use the services and insert falsified transactions in a network.

The authors in [107] proposed an Ethereum blockchain-based uPort platform. A smart contract is used to manage transactions and enable a user to access their identity in case they are lost. However, transaction data stored on the smart contract is publicly visible which raises security and privacy issues.

In paper [8] the authors presented a blockchain-based self-sovereign identity management framework. The proposed framework uses a smart contract that allows user to manage their identities based on the attributes such as name, age, and profession. Service providers enable users to access their services based on the validated attributes. However, data stored on the smart contract is immutable.

Furthermore, the authors in [4, 99] proposed a self-sovereign identity authentication for users and applications. Blockchain technology is used to store authenticated data and to ensure data transparency and immutability. An Authcoin protocol (blockchain-based validation and authentication) is used to ensure identity security. This work generates a new key pair (public and private key) for each new user and maintains the binding between keys and the key's owner. An Authcoin protocol is implemented using Colored Petri Nets (CPNs) [81] to identify and eliminate design flaws, security, and privacy issues. However, this work did not allow updating data and transaction history support.

In paper [53] discussed decentralized identity management using uPort, Sovrin, and ShoCard. An uPort is a blockchain-based framework that provides identities for services such as email and bank. It is based on the smart contract that maps uPortID

with identity attributes. The registry component is used to store the hashes of the JSON attributes, while original data is stored on the IPFS. However, the registry is centralized, and data can be compromised. In addition, uPort does not authenticate the identities of the owner. A user on the network could not identify that initiated claim is valid.

On the other hand, Sovrin is a permissioned-based identity network that allows only trusted parties (e.g bank, university) to run the node. Sovrin has control over users' identity and allows users to decide about attributes they would like to share with other parties. However, it does not verify the party with whom data is shared.

ShoCard is a blockchain-based identity authentication framework that gives a trusted identity by protecting users' identity. The main limitation of ShoCard is a centralized design that comes due to its central server to exchange the encrypted identity information between a user and a relying party. Therefore, ShoCard stores data in encrypted form, but data could be linked with both ShoCardIDs and relying on parties, which raises privacy.

In summary, most of the existing work is based on the centralized third party to control and manage identity privacy, which raises privacy issues.

2.3.3 Semantic Approaches to Access Control

We explore the most relevant work on role hierarchy, the relationship between roles, access control permissions, and rules using ontology. Access control is used to restrict unauthorized access to the data. Several access control models have been presented to enforce security on data such as relation-based access control model [64], rule-based access control model [112], attribute-based access control model [135, 136], mandatory access control model [58,137], discretionary access control model [69,148], and role-based access control model [147, 149].

In the following, we discuss the access control models.

Relation-based Access Control Model:

The relation-based access control model allows access to the data based on the relation a user has. With Relation-based access control model, permissions are modeled depending on the relationship between subject and object. For instance, on the Facebook, a user's friends have permission to read the post. The authors in [64], present a relation-based access control model that defined permissions depending on the relation between subject (e.g user) and object (e.g data). The proposed solution used ontology to represent subject and object in the form of classes.

For instance, permission 'use' is modelled to define the relationship between subject and object, such as 'student use PC'. They defined access control rules such as all students must have access to at least one PC among all available PCs. Pellet reasoner is used to reason the developed ontology. However, it is complex to model permissions for each subject and object. In addition, this work did not use any query language to enable end users to access the domain knowledge.

In paper [180] authors use description logic with relation-based access control model to control data access. The description logic [39, 62]. is used to define access

control policies. The proposed work manages user's access to the resources based on the permission between them. They defined access control policies such as David has a permission to modify the data. However, description logic reasoning is not efficient as it could not detect all instance relationships represented in hierarchical form.

Rule-based Access Control Model:

The rule-based access control model manage user's access to the resources according to the pre-defined set of rules. It ensures that user can access only to the resources they are allowed. It compares the user's access request with the access rules to make access granted or denied decision. In paper [112] the authors extend a rule-based access control model with semantic Web. The proposed solution defines access control rules (such as insert, read, delete) to determine which user has permission to access which resource. SWRL (Semantic Web Rule Language) is used to write access control rules. Ontology is used to represent the resources. However, this solution did not use HTTP verbs to design the rules.

The authors in [100] discuss rule-based access control policies as rules in the ontology. Ontology is used to represent users and resources in the form of classes and used object properties between classes such as permit and deny. SWRL is used to write rules and grant access to the resources based on these rules. For example, a group leader defines a access control policy such as only PhD student are permitted to access published resources. However, this work did not use reasoner to reason the rules and ontology.

Attribute-based Access Control Model:

This model grant access to the data based on the attributes, instead of the subject and role [135, 136]. The aim of Attribute-based access control is to prevent data from an unauthorized user, those have not defined by organization. With Attribute-based access control, access decisions are done according to the attributes of the subject, resource, and environment. A subject is a user who makes the request to access the data. Users' attributes include user's identity such as ID, name, job title, organization name etc. Resource is an object or data that user wants to access. Resource attributes are data creation date, name, and type etc. Environmental attributes contain the access time and data location. However, management of access control policies based on the attributes are complex and time consuming.

In authors in paper [134] discuss the attribute-based access control model in the context of semantic Web. Ontology is used to map different attributes with attribute's conditions such as age > 18. The proposed work used XACML to write access control policies. For instance, a user with age attribute (age > 18) has read access to the data. However, this work did not define level of permissions. In addition, the authors define only one policy to access data based on the age and email attribute.

The authors in [42] proposed a domain-specific framework that extends the role-based access control model with an attribute-based access control model using OWL ontology. The proposed work defined roles to identify the user within the system. The OWL ontology expresses the components of the access control model such as

roles, resource, attribute, and permission. They represent the classification of access control roles and resources using OWL ontology.

For example, in the university domain, all subclasses of role Student will be classified as Role and will be allowed to use the privileges of Student. Privilege is comprised of actions and resources. A SPARQL query is used to query the ontology knowledge. However, they assigned the same permission to the role and classified roles. In addition, they did not define the level of permissions according to classified roles to access the data.

Role-based Access Control Model:

Other access control models such as Mandatory Access Control (MAC) and Discretionary Access Control (DAC) are used to control access to the database [58, 137]. In DAC, the data owner decides who can access the data [69, 148]. DAC is a access control model for operating systems and relational databases. It is easy to understand. However, it does not ensure full security on the data and difficult to maintain for large number of users. In addition, it is time consuming and complex to define same permissions individually for several users performing the same actions. MAC is based on the hierarchical approach to grant access to the resources [36].

In MAC, operating system controls the access to the resources. It is based on the strict rules such as 'write-up' and read-down', where user is only allowed to write to data in their upper level of classification and is allowed only to read data in the down level of classification. However, it does not allow users to alter the access control of resources and users must make request to access for each new data [146].

Therefore, the RBAC is a famous security method that assigns permissions to the roles and then roles are assigned to the users [147, 149]. RBAC model supports both DAC and MAC [46]. It ensures security by preventing data from unauthorized access. RBAC is comprised of four parameters: user, role, object, and permission. The role defines user's rights or identity to retrieve the resources based on the given permission. The object can be any resource that can be accessed by a role depending on their assigned permissions. A permission defines access to different levels of data within the same domain [82, 86].

The authors in [68] focus on a university ontology that defines the concept of classes, subclasses, and object property. In this work, the authors explain the class hierarchy by splitting classes into sub-classes, such as, the class 'course' has two subclasses 'GraduateCourse' and 'UnderGraduateCourse'. The relationship between two classes is represented using object property, for instance, Department has a head Chair. In this example, 'Department' and 'Chair' are two classes, and 'has head' is a relationship between them. However, this work did not consider roles and access control permissions to manage access to the data.

In [41] the authors develop a project management ontology that restricts access to the data. The ontology is used to represent roles and permissions hierarchy to access the data. For instance, the role 'project member' has permissions to 'read' and 'write' the 'document'. On the other hand, role 'visitor' has permission to just 'read' the 'document'. The proposed ontology is implemented using a protégé. However, this work did not explain role hierarchies and relationships between them.

Table 2.1: Summary of related work analysis

Ref no	**Challenge Addressed**	**Approach**	**Advantages**	**Limitations**
[77]	HATEOAS realization	HATEOAS with attribute-based access control model	Ensure authorized data access	Complexity; Time-consuming
[90]	Develop a client to support navigation	HATEOAS with open APIs	Offers links to perform transition	Identified transitions are incorrect; Privacy risk remains
[102]	HATEOAS client	HATEOAS with REST APIs	Offers links to perform transition; Stateless	Only XML format is acceptable; Communication overhead
[103]	Enable trust between supply chain users	Blockchain-based software connector framework	Decentralization; Enhance trust between stakeholders	Not scalable to manage large amount of data
[40]	Maintain owner's data traceability	Blockchain based U-share framework	Data owner has control over their data; Transparency	Private key leakage: Privacy risk remains; Performance; Blockchain scalability
[175]	Maintain product traceability	Smart contract	Completely decentralized; Transparent data	Data is publicly available; Immutable
[185]	Decentralized personal data management	Blockchain; Distributed hash table	Remove trusted third party; Ensure data privacy	Need to secure symmetric key from unauthorized access; Permissions are immutable
[7]	Decentralized data management	Permissioned blockchain; Distributed hash table	Decentralization; Data transparency; Remove trusted third party	Need security and privacy; Store data on a distributed hash table without encryption; Privacy-sensitive data is publicly available

Table 2.1: Continued

Ref. no	**Challenge Addressed**	**Approach**	**Advantages**	**Limitations**
[76]	Maintain product traceability	Blockchain; InterPlanetary File System	Ensure track of the data	Single point of failure
[182]	Decentralized data management	IOTA Tangle; Blockchain; InterPlanetary File System	Ensure security on data	Single point of failure; Immutability
[21]	Distributed data storage	Storj	Data owner has control over their data	Depend on centralized architecture to conclude storage and payment
[139]	Decentralized data storage	Solid pods; Blockchain	Ensure data confidentiality	Need to maintain trust; Scalability
[3]	Personal data storage	Mydex	Ensure security on data	High cost; Centralized
[165]	Decentralized data management	ONTOCHAIN; Smart contract; Ontology	Maintain trust; Decentralization	Require security schemes
[157]	Decentralized data management	ONTOCHAIN; Smart contract; Ontology	User can keep track of the data	Need to prevent data from unauthorized access; Immutable
[35]	Decentralized data management	Smart contract; Ontology	User can keep track of the data	Require security schemes; Immutable
[143]	Personal data management	GDPR; Semantic web	Ensure data privacy	Need to ensure data security;
[84]	Decentralized data management	Smart contract; Ontology; InterPlanetary File System	Data interoperability	Need to prevent data from unauthorized data access; Security risk remains; Immutable

Table 2.1: Continued

Ref. no	Challenge Addressed	Approach	Advantages	Limitations
[95, 104]	Maintain track of the data	Smart contract; GDPR	Ensure trust; Transparency	Need to ensure data security
[50, 132]	Decentralized data sharing	Symmetric key cryptography; Attribute-based access control	Ensure data confidentiality and integrity	Immutable
[83, 110]	Management of data sharing	Smart contract; InterPlanetary File Sytem System	Data interoperability	Data is publicly available; Scalability
[88]	Anonymized dataset	Blockchain	Decentralization; Data owner can trace their data	Need third party to manage privacy policy; Privacy risk remain
[17]	Ensure identity in bitcoin blockchain	Zero-knowledge proof	Dynamic update of identities	Need third party
[29]	Anonymity of user data over blockchain	Zero-knowledge protocol; Smart contract	Ensure data privacy; Remove third party	Require more security schemes; Unable to work when verifier is malicious
[54]	Secure data management	Ring signature; Smart contract; Cryptographic techniques	Reduce blockchain bandwidth and computational power	Scalability
[107]	Decentralized digital Identity management	uPort; Smart contract	Completely decentralized; Ensure data transparency	Need to ensure data security
[8]	Decentralized digital Identity management	Smart contract	Completely decentralized; Ensure data transparency	Immutability
[4, 99]	Decentralized digital Identity management	Blockchain	Ensure data transparency; Ensure trust	Immutability

Table 2.1: Continued

Ref. no	Challenge Addressed	Approach	Advantages	Limitations
[64]	Manage unauthorized data access	Relation-based access control model	Ensure data access control	Complex to model permissions; Unable to change policies at run time
[180]	Manage unauthorized data access	Description logic and relation-based access control model	Easy to understand; Ensure authorized data access	Not efficient reasoning; Unable to identify instances in hierarchies
[112]	Manage unauthorized data access	Rule-based access control model and semantic Web	Ensure authorized access to the data	Difficult to manage rules for each user
[134]	Management of attribute-based access control policy	Use XACML with attribute-based access control model	Easy to use policy editor	Limited to age and email attributes; Difficult to manage attributes
[42]	Static RBAC	Use ontology to represent classes and relations corresponding to RBAC	End user can query the data	Not defined level of permissions to access data
[60]	Data access control	Integrate Owl ontology with RBAC	Useful to share information; Reduce system design complexity	Fixed roles, resources and permissions; Limited to one domain; Not define complex permissions between users
[138]	Manage access to the resources	Combine role-based access control model with ontology	Addressed role hierarchies; Flexible to add sub-roles	Not defined level of permissions
[86]	Authorized access to the resources	Integrate RBAC with ontology	Addressed role hierarchies and permissions;	Not defined complex relationships between users
[68]	Organize unstructured data	OWL Ontology	Represent subclasses hierarchies	Need to define access control permissions

Table 2.1: Continued

Ref no	Challenge Addressed	Approach	Advantages	Limitations
[152]	Decentralized data management and access control	Blockchain; Distributed hash table; Fine-grained access control	Decentralization; Ensure data security	Privacy risk remains; Permissions are immutable
[63]	Decentralized data management	Blockchain; Ontology	Maintain trust	Security risk remains; Immutable
[72]	Decentralized data storage	Blockchain; Distributed hash table	Decentralization; Efficient storage	Data is publicly available; Need to ensure data security
[163]	Maintain product traceability	Blockchain; Radio-Frequency Identification	Ensure track of the data	Data is immutable; Scalability
[53]	Decentralized digital Identity management	Sovrin, Uport and ShoCard	Completely decentralized; Ensure data transparency; User can control data transactions	Need to manage cryptographic keys
[159]	Represents role concept and role hierarchy	Ontology-based framework	Represents role hierarchies	Relationship between roles is very simple; Not defined permissions to restrict unauthorized data access
[41]	Manage access to the resources	Ontology-based access control framework	Define the level of permissions; Handle access to the resources	Not defined role hierarchies
[167]	Manage unauthorized data access	Extend RBAC using Ontology	Represent role hierarchy	Did not manage complex permissions between individuals

The authors in [167] integrate the RBAC model with a role ontology to ensure security on the data. The proposed work represents the role hierarchy by converting an ontology into a tree structure (such as parent and child). They assigned permissions to the roles to access and read the data. The proposed ontology allows only authorized users to access the data based on their roles. However, this work did not define the permissions hierarchy according to the levels of roles. Additionally, this work did not manage complex permissions between individuals.

The authors in [86] describe a domain specific RBAC model with an ontology approach. The ontology is used to represent and manage the RBAC roles as a class having permissions given to them. Role hierarchies of the RBAC model can easily be managed and represented in the ontology as classes and subclasses. The proposed solution assigns a role to the user if the user id and password entered by the user are matched with pre-stored entries in the knowledge base. However, this solution is unable to handle complex permissions. In addition, this work did not define the relationships between users.

In [159] the authors propose an ontology-based framework that represents the role hierarchy and relationship between them. For instance, a 'teacher' is a role and has a hierarchy: a 'high school teacher' and 'elementary school teacher'. In the proposed framework, the 'is-a' relation is used between roles that belong to the same hierarchy, such as 'high school teacher' is-a 'teacher' represents the relationship between two roles. Role hierarchy and the relationship between them are represented using the Hozo tool. However, the proposed framework did not define access control permissions to manage data access. In addition, they use a very simple relationship between roles.

The authors in [60] proposed a framework that integrates ontology with XACML (Extensible Access Control Markup Language) policies to extend the existing RBAC model. XACML is used to manage the authorization policies called a set of rules. Rules are assigned to the roles to perform actions. Ontology is used to represent the role hierarchies of RBAC and XACML policies. The authors represent the role hierarchies of the university domain, for example, Dean role has a role hierarchy such as a full professor, Associate professor, and Assistant professor. Based on these roles, they define policies such as every professor, except assistant professors, can review a project. The proposed framework has advantages in terms of flexibility and easy to use. However, this solution did not define complex permissions between users.

In [138] the authors extend the role-based access control model using ontology. The proposed work used ontology to represent the role hierarchies and policies that restrict access to the resources according to the given permissions. The proposed ontology has an advantage in terms of flexibility to add new roles in the role hierarchy. However, the authors did not assign permissions to each role in the hierarchy.

The authors in [177] proposed an ontology-based RBAC that represents the concepts and relationships. The proposed ontology is defined classes based on the concept of RBAC such as users, roles, permissions, and sessions. Relationships between these classes are defined using an object property. For instance, property 'hasRole' and 'hasPermission' is used to define the relationship between classes user, role, and permission. However, this work assigned the same permission to the senior role and junior role.

The authors in [27] proposed the ontology for the wood supply chain. The proposed ontology represents the class hierarchy and object properties and defines the relationship between classes. For instance, a class 'size' is further categorized into two sub-classes such as 'height' and 'width' to show the class hierarchy. They define object property 'represent' between two classes such as 'size' and 'meter'. It shows a relationship between two classes such as size can represent in meter. Protégé is used to develop the ontology. This work used Racer reasoner to check the ontology consistency. However, the authors did not allow an end-user to query the data. In addition, they did not define permission to manage unauthorized access to the data.

In summary, existing access control models are complicated to manage permissions for each user, object, rules, and attributes, where the number of users is very high, and some users are not known in advance. In addition, it needs a lot of time and resources. If access rights for the users and data are required to be altered in the organization, then it is complex to make changes for each subject, object, and attributes in that organization. Therefore, RBAC model assigns permissions to the roles and then roles are assigned to the users and management of roles are easy as compared to management of each user's rights and attributes.

However, the complexity of the RBAC model remains to manage and represents non-hierarchical relationships between actors. Also, it is static in nature and could not adopt changes at run time. Therefore, ontology is useful to support both hierarchical and non-hierarchical relationships due to its graph representation. It is flexible to add roles, objects, and permissions at run time. Most existing solutions did not manage complex relationships between users. In addition, existing solutions did not handle complex permissions to access data. Thus, motivating research towards developing an access control framework that manages access to data using ontology.

2.3.4 REST and HATEOAS

In this section, we explore the existing literature on HATEOAS with REST APIs. The authors in [47] discuss the framework based on REST and HTTP requests to enable communication between users. The proposed framework uses Petri Net Markup Language (PNML) [25] as a formal model to describe the execution process of RESTful services. However, this work does not support the concept of HATEOAS, nor authorization aspects to restrict unauthorized access to the resources.

In [130] the authors extend the Business Process Modeling (BPM) using graphical syntax and semantics to support REST. The proposed solution presents a simple process to connect with external resources (e.g REST APIs). However, this work did not address hypermedia or HATEOAS. In addition, it could not identify resource identifiers dynamically.

The authors in [102] introduce the concept of HATEOAS. The proposed approach uses the wrapper between the client and API that generates hyperlinks. This wrapper processes the client request and sends it to the actual server. It receives the server response and sends back to the client. However, this wrapper might change the request and response content, which increase the security issues. However, the proposed wrapper is based on the manually created model that explains the states

of the wrapped application. Additionally, the proposed approach use XPath2 (XML Path Language) that point to values in the server response and only handle the services based on the XML messages.

In [77] the authors present a HATEOAS that enables only authorized clients to request the data. The RestACL (REST Access Control Language) is used to manage access to the data. The RestACL is based on the concept of an attribute-based access control model that restricts access based on the user's attributes e.g name. However, the management of access control policies according to the attributes is complex and time-consuming. In addition, the proposed approach is based on customized resources that work only with computer to computer and do not work with a Web browser.

The authors in [90] discuss the HATEOAS principle using API. The proposed approach uses a proxy application that returns hyperlinks in the client's response. The authors in [90] claim that the proposed approach shows high false positives, where it identifies incorrect transitions. However, the proposed solution supports only those APIs that contain only one path parameter e,g id. Additionally, unauthorized data access aspects are not investigated.

The authors in [127] discuss the HATEOAS based AProPro (Adapting Processes via Processes) framework. The proposed framework is based on an adaptation process that adapts to changes in other processes. However, the adaptation process could change the active node that does not provide the next possible actions. In addition, this work is not optimized because it shows high overhead while increasing the number of actions.

In summary, most of the existing solutions are subject to scalability, time overhead, and security issues. Some solutions use a wrapper between client and server which can modify the actual message and response content. Existing solutions are based on static link templates and do not allow the client to add and modify the templates.

2.3.5 Comparative Analysis

In this section, we present a features comparison of the proposed solution with the other existing solutions discussed in Section 2.3. The features that distinguish our work are decentralization, data privacy, security properties, multiple types of encryption, attacks prevention, data updates and transaction history support.

Table 2.2 provides a global overview of existing solutions with respect to these features. In the following, we analyze in detail each solution of Table 2.2, including their main advantages and limitations.

Decentralization shows if the solution stores or manages data without any trusted third party. Data privacy determines if it ensures the data owner's access control on the data. Security properties reflect to enforce security on data such as confidentiality, integrity, availability, and non-repudiation. Multiple types of encryption determine whether the solution provides different options between encryption mechanisms to encrypt the data. Attacks prevention shows if it prevents data from attacks such as linking, eavesdropping, and modification. Data updates determine if data owner can modify their data. Transaction history support shows if data owner can

access their update history.

The table shows that the mydex solution [3], ensures the privacy of the data and provides security properties (e.g confidentiality and integrity). However, centralized data storage and attacks risks are drawbacks because it is managed by a single authority.

The on-chain data management solution [165] is decentralized which ensures data privacy and supports data updates. However, this solution stores data without encrypting it, which is subject to security and attack risks.

DECODE (Decentralized Citizens Owned Data Ecosystem) as proposed in [35], enforces security on data such as confidentiality, integrity, and availability. It enables users to update the data in the decentralized platform. However, it does not protect against attacks such as Linking attack, modification attack, and eavesdropping attack.

In [143], ROPA is presented to manage personal data. This solution is decentralized and ensures privacy on data according to the GDPR rules. However, it does not provide security properties and attacks risk are also possible.

In [105], the authors proposed a BOWLER (Blockchain-Oriented Warehouse & Low-Code Engine and Reasoner) framework that ensures security property such as availability. However, it does not protect data from unauthorized access, thus making data privacy major concerns. In addition it does not support data updates and transaction history.

CARECHAIN (Supporting CARE through micro-insurance using blockchains) as proposed in [162], achieves both decentralization and data privacy. The proposed solution is unable to support security properties, attacks prevention, data updates, and transaction history.

In [63], the authors proposed a copyright solution to manage data and ensures data privacy in a decentralized platform. However, it does not enforce security properties (e.g confidentiality, integrity, and availability, and non-repudiation). In addition, multiple types of encryption, data updates, attacks prevention, and transaction history aspects are also not considered.

The authors in [43], proposed a data storage solution that ensures decentralization, data privacy and integrity. However, this solution stores data publicly, which leads to data security and attacks issues. In addition, it is unable to support data updates and transaction history due to the immutability feature of the blockchain-based smart contract.

In [94], the authors presented a decentralized data management solution that only ensures data availability security property. However, it stores data without encrypting it, which leads to security breach and attacks.

In [18], the authors discussed a solution to manage data on blockchain. It addressed data privacy and availability security property. However, it does not support data updates due to immutability property of blockchain. In addition, multiple types of encryption, attacks prevention and transaction history support aspects are also not considered.

The authors in [125] presented a solution that manages data storage and owner's control in a decentralized platform. This solution only achieves integrity and availability security properties. The proposed solution is unable to ensure data privacy

and prevent against attacks because data is not encrypted before storing it.

In [37], the authors proposed a self-sovereign identity management and data sharing solution which does not enforce security on data. This solution enables data owner to share their personal data which is subject to data privacy concerns. It does not enable users to update their data and maintain transactions history. In addition, attacks prevention aspect is also not addressed.

The authors in [7], presented a framework to manage data without a trusted third party. They ensure the integrity and availability of data in a decentralized framework. However, data is available to everyone on the network, which is subject to data privacy and attacks risks. In addition, they did not address encryption mechanisms, data updates, and transaction history support features.

In [101], the authors combined blockchain with DHT to store IoT data. They ensure data privacy and updates in a decentralized platform. However, they do not address security properties such as confidentiality, integrity, availability, and non-repudiation. In addition, the transaction history support aspect is also not considered.

The authors in [84], presented a decentralized ISLAND (Interlinked Semantically enriched Blockchain Data) framework. The proposed framework achieves security properties such as availability and integrity. However, privacy risks remain an issue because data is not prevented from unauthorized access. Another drawback of this framework is data update due to the permanent storage of data on the smart contract and IPFS.

In [99], the authors proposed a self-sovereign identity management solution that ensures identity privacy without any trusted third party. It addresses security properties such as confidentiality, integrity, and availability. However, attacks prevention, data updates, and transaction history support are not considered.

NFTWATCH framework, as presented in [80], manages digital tokens associated with data in a blockchain. It enables data owner to control their data and make queries in a decentralized platform. This solution is unable to ensure data security, privacy, data updates, and transaction history. In addition, this solution did not use any encryption technique to encrypt the data before storing it.

The authors in [157], presented a solution to manage semantic data in the blockchain. However, blockchain is managed by a centralized mainchain component which becomes a single point of failure. Another drawback of this solution is security, privacy, and attacks risks due to public accessibility of data. In addition, data update and transaction history aspects remain an issue.

In [21], the authors proposed a solution to store data without any trusted third party. The major disadvantage of this solution is data privacy, security, and attacks.

The authors in [104], presented a solution that maintains track of data in a decentralized platform. It prevents unauthorized access to the data which ensures data privacy. However, security properties attack prevention. data updates, transaction history, and queries to read data are not provided in this solution.

In [110], the authors presented a solution to manage data in a decentralized platform. It address security properties such as (confidentiality, integrity, availability, and non-repudiation) and allows authorized users to update the data. However, it does not support transaction history. Another disadvantage of this solution is public

accessibility of data which increases privacy risks.

The authors in [132] proposed a decentralized data sharing framework which ensures data privacy and security using symmetric key cryptography. However, it does not allow user to update their data and maintain transaction history.

In [96], the authors proposed a decentralized data management framework which enforce security on data (such as confidentiality and integrity). They ensures data privacy by limiting unauthorized access to the data. However, it does not allow to update data and maintain transaction history. Additionally, attacks prevention remains an issue.

The authors in [102], discussed HATEOAS with REST APIs solution which allows user to read the data. The major drawback of this solution is their centralized design which makes single point of failure issue and vulnerable to privacy, security and attacks risks. In addition, data updates and transaction history are not addressed.

Another solution [77], presented a HATEOAS solution which ensures data privacy using RestACL. However, this solution is centralized, thus making data security and attacks major concerns.

In [90], discussed HATEOAS client which is not decentralized. They did not investigate data privacy, security and attacks aspects. In addition, this solution does not allow HATEOAS client to update their data and maintain transaction history.

As we can see from the analysis presented above, existing solutions do not completely fulfill the features presented in Table 2.2. Our proposed solution is more reliable and secure due to the following reasons:

- The proposed solution is completely decentralized to store and manage the data without any centralized party.
- Our solution ensures data privacy by protecting data from unauthorized access. The data owner has control over their data and the data owner decides who may access their data.
- Our solution ensures the following security properties: confidentiality, integrity, availability, and non-repudiation.
- It is flexible to use different types of encryption to ensure data security.
- It protect data against attacks such as linking, eavesdropping, sybil, spoofing, and modification attacks.
- Our solution enables actors to modify their data and maintains their transaction history.

2.4 Discussion and Conclusion

In this chapter, we provided the key concepts of the main research areas that are relevant to this dissertation. We also described the existing literature dealing with the research challenges identified in Chapter 1. Based on our literature review, we

identified the research gaps in available knowledge. Moreover, we compared the features of our proposed solution with existing solutions and showed the position of this dissertation. Based on our previous analysis, these features are decentralization, data privacy, security properties, multiple types of encryption, attacks prevention, data updates, and transaction history support.

In summary, based on the literature review, we observed that existing solutions are suffering from the following issues:

- Some existing solutions are based on centralized storage design, thus making single point of failure major concerns [3, 76, 182].
- Most existing data storage and management solutions store data publicly in a decentralized ledger. However, public availability of data in such ledgers does not take data privacy requirements into account [72, 84, 175]. In addition, some solutions are not scalable to handle large amount of data since the data is replicated on each node of the ledger [110, 163].
- Some existing solutions provide security and privacy to store and manage the data. However, they did not offer flexible encryption design to encrypt the data. Additionally, they did not provide any solution to update data and transaction history support [3, 99].
- Most solutions are suffering from lack of data access control in a decentralized ledger [83, 84, 157]. In addition, some solutions did not address complex relationships and permissions between actors [60, 86, 159]. Most of the solutions did not design access control rules and permissions using HTTP verbs such as POST, GET, PUT, and DELETE [41, 112].
- Some solutions address only availability and integrity security properties. However, they did not to ensure confidentiality and non-repudiation to enforce security on data [7, 84, 132].
- Some solutions use a proxy to process the client's request and send it to the actual server. Then it receives the actual server's response and sends it back to the client. However, it increases the chance of modification in the actual request and response content [102].

The goal of this thesis is to provide a solution that overcome the limitations discussed above. In the next chapter, we detail our first contribution dealing with data management using REST APIs without any trusted third party.

Table 2.2: Features comparison of our proposed work with existing solutions

Ref.no/year	**Decentralization**	**Data privacy**	**Security properties**	**Multiple types of encryption**	**Attacks prevention**	**Data updates**	**Transaction history support**
[3], 2018	No	Yes	Confidentiality; Integrity	No	No	No	No
[165], 2021	Yes	Yes	Confidentiality; Integrity; Availability; Non-repudiation	No	No	Yes	Yes
[35], 2017	Yes	Yes	Confidentiality; Integrity; Availability	No	No	Yes	No
[143], 2021	Yes	Yes	No	No	No	Yes	No
[105], 2020	Yes	No	Availability	No	No	No	No
[162], 2020	Yes	Yes	No	No	No	No	No
[63], 2019	Yes	Yes	No	No	No	No	No
[43], 2020	Yes	Yes	Integrity	No	No	No	No
[94], 2020	Yes	No	Availability	No	No	No	No
[18], 2020	Yes	Yes	Integrity	No	No	No	No
[125], 2020	Yes	No	Availability; Integrity	No	No	No	No
[37], 2019	Yes	No	No	No	No	No	No
[7], 2017	Yes	No	Integrity; Availability	No	No	No	No
[101], 2018	Yes	Yes	No	No	No	Yes	No

Table 2.2: Continued

Ref.no/year	**Decentralization**	**Data privacy**	**Security properties**	**Multiple types of encryption**	**Attacks prevention**	**Data updates**	**Transaction history support**
[140], 2021	Yes	Yes	Integrity	No	No	No	No
[84], 2021	Yes	No	Availability; Integrity	No	No	No	No
[99], 2017	Yes	Yes	Confidentiality; Integrity; Availability	Yes	No	No	No
[80], 2020	Yes	No	No	No	No	No	No
[157], 2022	Yes	No	No	No	No	Yes	Yes
[21], 2021	Yes	No	No	No	No	No	No
[104] , 2021	Yes	Yes	No	No	No	No	No
[110], 2021	Yes	No	Confidentiality; Integrity; Availability; Non-repudiation	No	Yes	Yes	No
[132], 2017	Yes	Yes	Confidentiality; Integrity	Yes	No	No	No
[96], 2020	Yes	Yes	Confidentiality; Integrity	No	No	No	No
[102], 2011	No	No	No	No	No	No	No
[77], 2018	No	Yes	No	No	No	No	No
[90], 2020	No	No	No	No	No	No	No
Our proposed solution	Yes	Yes	Confidentiality; Integrity; Availability; Non-repudiation	Yes	Yes	Yes	Yes

Chapter 3

Decentralized Web Framework for Data Management

The Results of this chapter are published in the following articles:

- Aslam, S., Mrissa, M.: A RESTful Privacy-Aware and Mutable Decentralized Ledger. *European Conference on Advances in Databases and Information Systems (pp. 193-204). Springer, Cham, 2021*
- Aslam, S., Bukovszki, B., Mrissa, M.: Decentralized Data Management Privacy-aware Framework for Positive Energy Districts. *Energies, 14(21), 7018, 2021*
- Aslam, S., Mrissa, M.: Privacy-aware Distributed Ledger for Product Traceability in Supply Chain Environments. *Conference of SWST International Society of Wood Science and Technology, 2020*

3.1 Introduction

As explained in the previous chapters, the need for a decentralized framework is crucial to overcome the limitations of cloud-based approaches. In this chapter, we propose a Web-based framework that eliminates the need for a trusted third party to manage data. To do so, we combine blockchain technology with Distributed Hash Table (DHT), access control ontology, and multiple encryption mechanisms. Those contributions, detailed in the following chapters of this dissertation, are supported by a set of RESTful APIs that we designed to support our decentralized solution and ensure interoperability over the Web. Each peer of the framework exposes the same set of APIs, so that they are easy to maintain and enable smooth communication over HTTP between peers. The proposed RESTful APIs are fully compliant with the REST architectural style, they enable loosely-coupled interactions between network peers using HTTP calls, appropriate usage of the HTTP verbs, and decentralized management of the data in the wood supply chain (WSC).

This chapter is structured as follows. First, we provide an overview of our framework. Then, we explain the actor registration algorithm using our REST APIs. After

that, we detail each actor's interaction with the framework to manage their data according to our motivating scenario. We explain in detail how our APIs support those interactions. After that, we detail framework components including the access control ontology component, blockchain component, DHT component, and encryption manager component. Then, we present the implementation and discussion of the proposed solution. Finally, we summarize this chapter.

3.2 Framework Overview

Our framework enables authorized actors to write, read, update, delete data and interact with other actors via HTTP calls. Figure 3.1 depicts the framework overview and its components organized around a `main` program. In our framework, actors are the nodes or peers of the framework and they are running the `main` program and they call the `registry_server` component to receive the list of connected peers and connect with each other through their APIs.

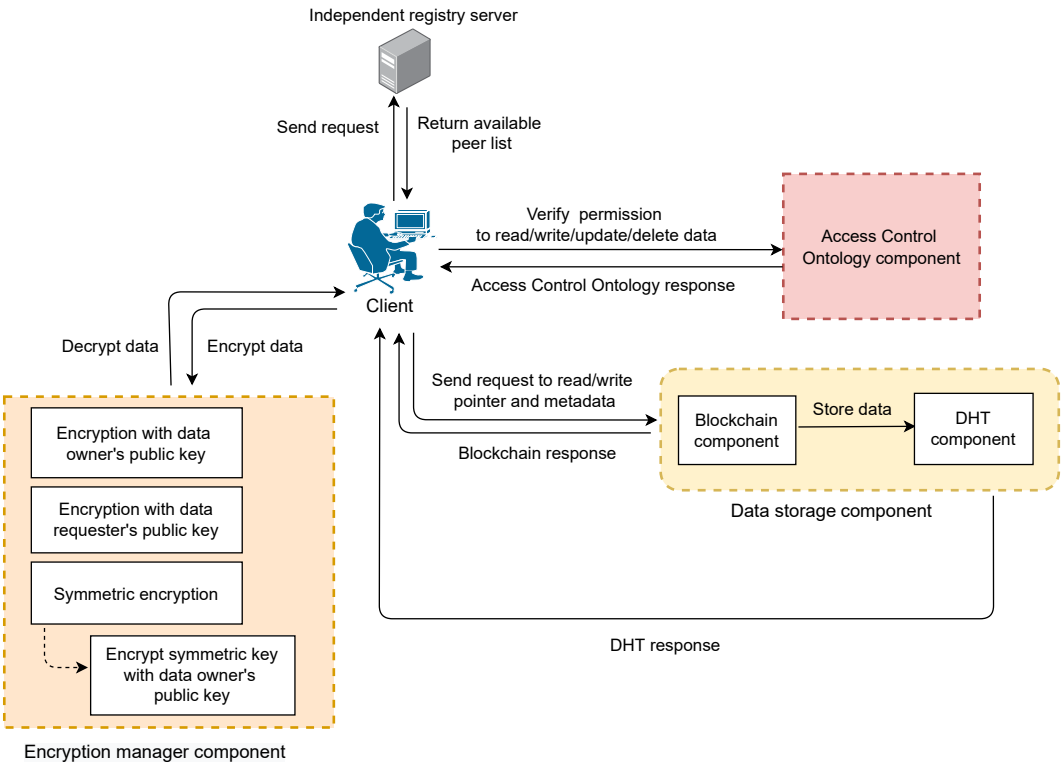

Figure 3.1: Overview of a peer architecture.

Let us consider our running motivating use case: a forest manager actor logs in our framework to some details about the trees cut on this day. The forest manager program will call the `/peers` resource with the `'POST'` method of the registry server to add its URL (Uniform Resource Locator) and public key to the list of available peers. Then, it will receive the list of connected peers by calling `/peers` resource with method `'GET'`. After that, it will call the `/chain` resource using the method

'GET' to receive or copy the last 40 transactions of blockchain from other connected peers (please note that, if an actor wants to see more blocks then here is a possible option to do it using HATEOAS links. To do so, we can provide the first block of the blockchain and then links to the other blocks. This way an actor can access the blocks of the blockchain using links until it gets the required data).

Upon request, the `main` component will call the `access control ontology` component to authenticate the permissions of the current actor, for example, a current actor such as a forest manager actor is authorized to write, read, update, and delete data or not. The proposed `access control ontology` component is responsible to manage relationships between actors and handle complex permissions for data access. In our `access control ontology` component, we define actors' rules and permissions to manage access to the data.

After verifying the permission of the actor, our `encryption manager` component allows the authorized actor to select between different types of encryption methods to store and read the data. It is also responsible to create a public and private key, or symmetric key of the actors. To store the data, the `main` component will call the `encryption manager` component to encrypt the entered data with the current actor's public key or symmetric key depending on the encryption method selected by the actor. Then, this encrypted data will be stored on the off-chain (key, value) storage known as the `DHT` component, while the corresponding DHT key (hash of the data) and metadata will be sent to the `blockchain` component. An authorized actor is allowed to create, read, update and delete their data using the DHT key stored on the `blockchain` component.

Accordingly, an actor will send a `'POST'` request to the `/chain` resource to create a new block. Our framework enables authorized actor to update their data on the chain. To update the data, it will make a `'PUT'` request to the `/chain/<id>` resource. Accordingly, a request to (`/chain/<id>`, method `'DELETE'`), will delete the data. To read the data, an actor will call the resource `/chain/<id>` using `'GET'` method. A request to the resource `/public_key` with method `'GET'` will return the public key of the actor. Figure 3.2 depicts the swagger user interface that allows authorized actors to use the APIs discussed above.

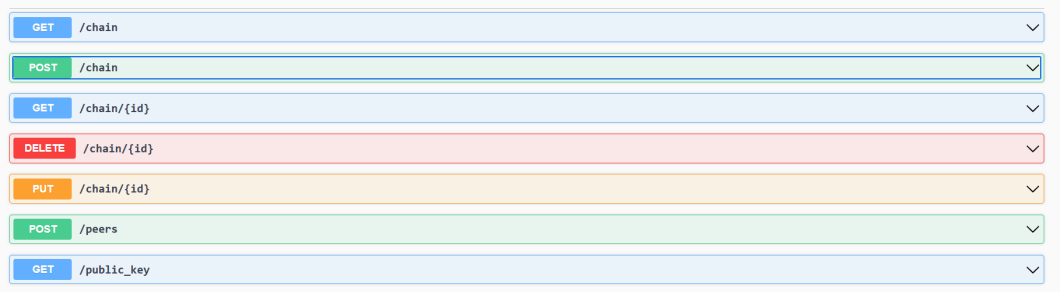

Figure 3.2: Overview of the proposed APIs using Swagger.

3.2.1 API and Algorithm for Actor Registration

Algorithm 1 presents the actor registration process using our designed APIs. It allows actors to connect to the proposed framework and receives the information of available peers.

Lines 1-3 represent that, the actor uses the `/peers` resource with `'GET'` method to retrieve the list of available peers list (`PL`). The new actor calls the `/peers` resource with `'POST'` method to insert its endpoint `PE` or address to the list of available peers and successfully connects with the registry server (lines 4 and 10). Then it iterates peers in the peer list and sends a request to other available peers to acknowledge the connected peer (`/peers` resource, `'POST'` method) in lines 6-8. Then, lines 11-17 represent if it's already in the peer list then disconnect it from the list using the `/peers` resource with `'DELETE'` method. The request will send to other available peers to acknowledge the disconnected peer in line 14.

Algorithm 1 Actor registration algorithm

Input: ca: current actor
Output: 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)
▷ PL: peer list (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
```

3.2.2 Execution Flow

In our framework, the actors participate to perform different actions on the data depending on their permissions and roles. Figure 3.3 represents the overview of each actor's actions (such as GET, POST, PUT and DELETE) on the data in the framework. In Section 3.2, we develop a detailed discussion of the internal operation of a peer and explain to it interacts with the framework components using RESTful APIs and HATEOAS links.

In the following, we develop a detailed execution flow of our framework that illustrates the interaction between each actor and the framework using RESTful APIs. Our solution maintains the id's references to ensure traceability. It allows actors to verify the origin of the product in the chain.

Figure 3.4 shows the execution flow between the forest manager actor and the framework. We assume that every actor is already registered on the framework. A forest manager actor makes a `'POST'` request to the `/chain` resource to write log data in the framework. The code sample below illustrates what the data that describes a log may contain:

```
{
    "id": "RFID_number",
    "resource": "log",
    "woodtype": "oak",
    "datetime": "2022-03-16, T-19:20:30.45+01:00",
    "location":
    {
        "lat": "38,3951",
        "long": "-77,0364"
    }
}
```

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 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, a forest manager 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 successful response (HTTP code 200), our framework returns the representation of the log data.

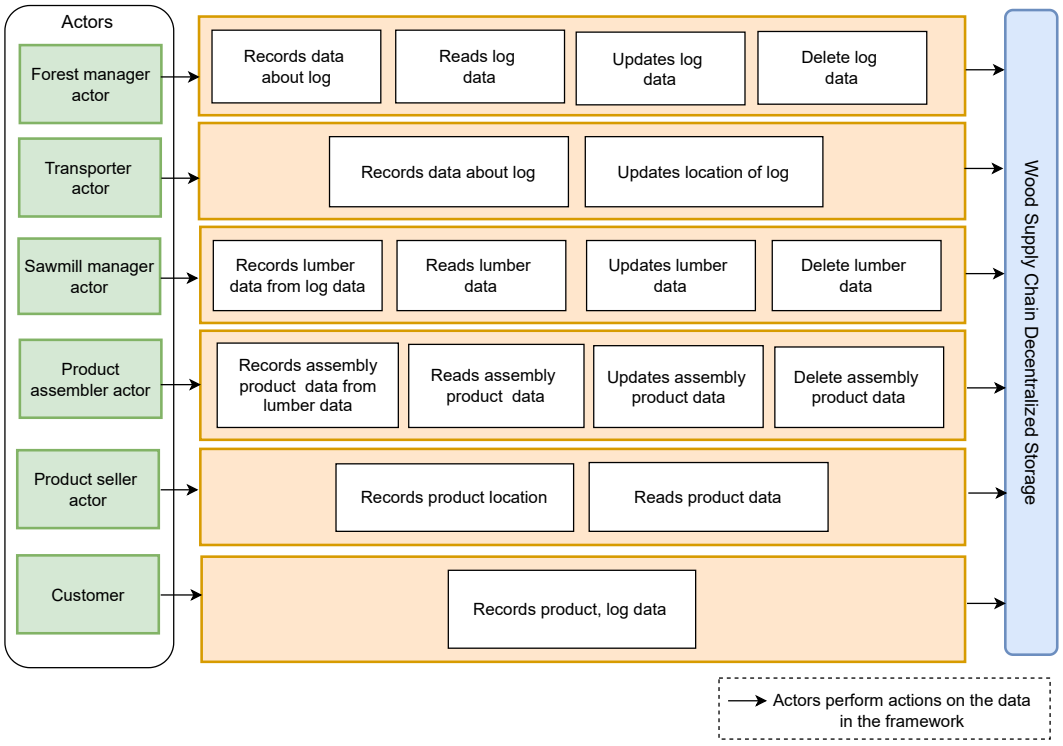

Figure 3.3: High-level representation of actors actions on the data.

In case a forest manager actor wants to update their data, then they use the PUT link that makes a `'PUT'` request to the (`/chain/<id>` resource). To update the data, our access control rules verify the actor's permission, for instance, if a current actor such as a forest manager is authorized to update the data or not. We develop the details of access control rules in Chapter 5. It will then writes new data against the same id. Then, a new metadata structure creates on the blockchain that contains the new DHT key of this updated data and the previous pointer of the old version of the data (more details develop in Chapter 4). In the successful case, it updates the log data.

Similarly, to delete the data, a forest manager actor may follow the DELETE link (`/chain/<id>` resource, method `'DELETE'`). Our framework allows the authorized actor to delete the specific data based on the id. After verifying the permission of the forest manager actor, it will delete the data. In this case, a new metadata structure creates on the blockchain that has a new DHT key with a NULL value. In the successful response, it receives log data deleted.

Figure 3.5 shows the execution flow between the transporter actor and the framework. Transporter actor will call the `/chain/<id>` resource with method `'GET'` to retrieve the representation of the log data. In response, the framework returns the representation of the log data and links to perform further actions on the data. Then, the transporter actor follows the link to modify the location of the log (`/chain/<id>` resource, method `'PUT'`).

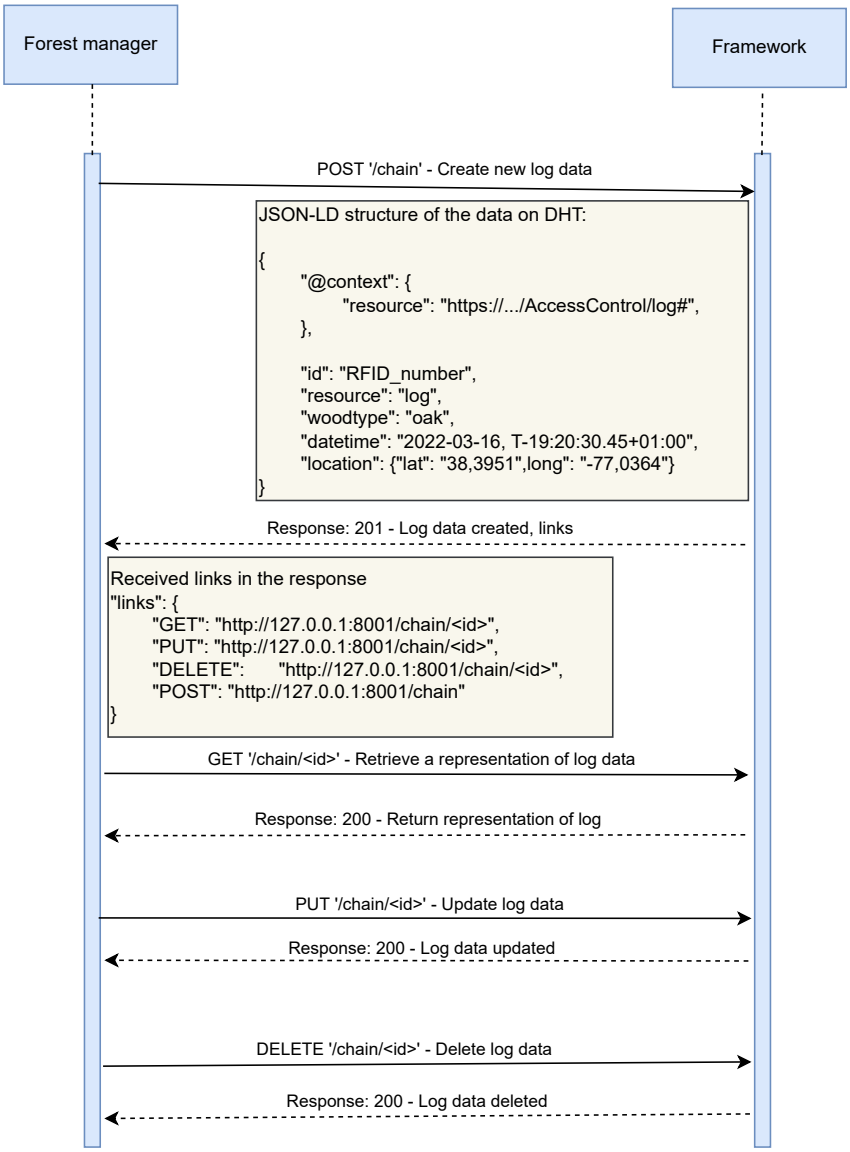

Figure 3.4: Execution flow of forest manager actor in the framework.

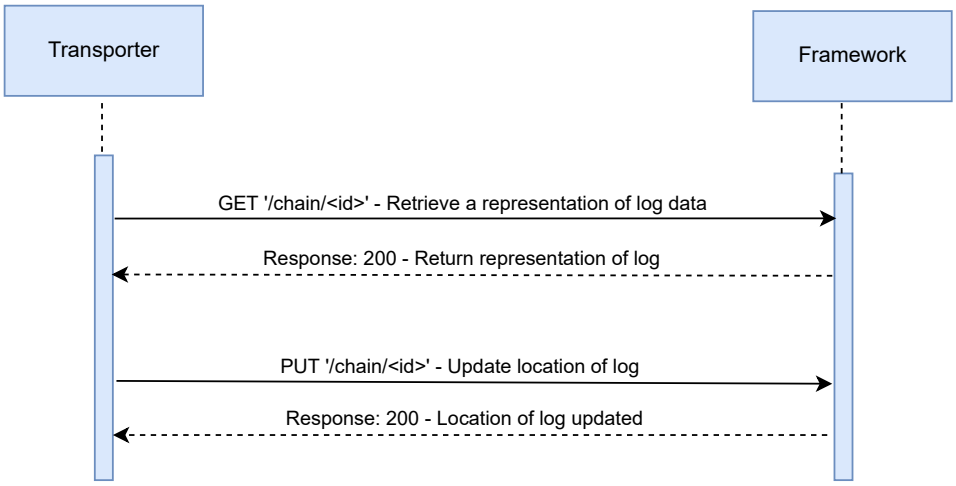

Figure 3.5: Execution flow of transporter actor in the framework.

Figure 3.6 depicts the interaction of the sawmill manager with the framework. A sawmill actor makes a `'POST'` request to the `/chain` resource to write lumber data in the framework as presented below:

```
{
    "id": "RFID_number",
    "resource": "lumber",
    "datetime": "2022-05-18, T-17:10:35.45+01:00",
    "location":
    {
        "lat": "33,3242",
        "long": "-56,0413"
    },
    "log":
    {
          "id": "RFID_number"
    }
}
```

As shown in this data, lumber has an reference id such as RFID_number of the log that was cut before. Therefore, it enables sawmill manager to verify the origin of the log. In the successful case, sawmill manager will receive the links of this lumber data and DHT key will be stored in the metadata.

The sawmill manager can use these links to perform further actions on the data such as read, update, and delete the lumber data.

The execution flow between the product assembler actor and the framework is presented in Figure 3.7. As we can see from Figure 3.7, an actor will call the `/chain` resource (`'POST'` method) to write assembly product data in the framework, as shown below:

```
{
    "id": "RFID_number",
    "resource": "product",
    "datetime": "2022-07-22, T-15:13:13.45+01:00",
    "location":
    {
        "lat": "23,1242",
        "long": "-43,1245"
    },
    "lumber":
    {
        "id": "RFID_number"
    }
}
```

The product assembler actor has the unique data id such as RFID_number of the product. It also has a reference of lumber id (such as RFID_number) that is used to build the furniture. This lumber id has the reference of the log (such as RFID_number). In the successful case, it receives the links in the response, and the DHT key of this data will be sent to the metadata. To read the data, the product assembler actor would make a `'GET'` request to the `/chain/<id>` resource. In the response, the framework returns the representation of product assembly. Similarly, product assembler actor can update and delete their data using links.

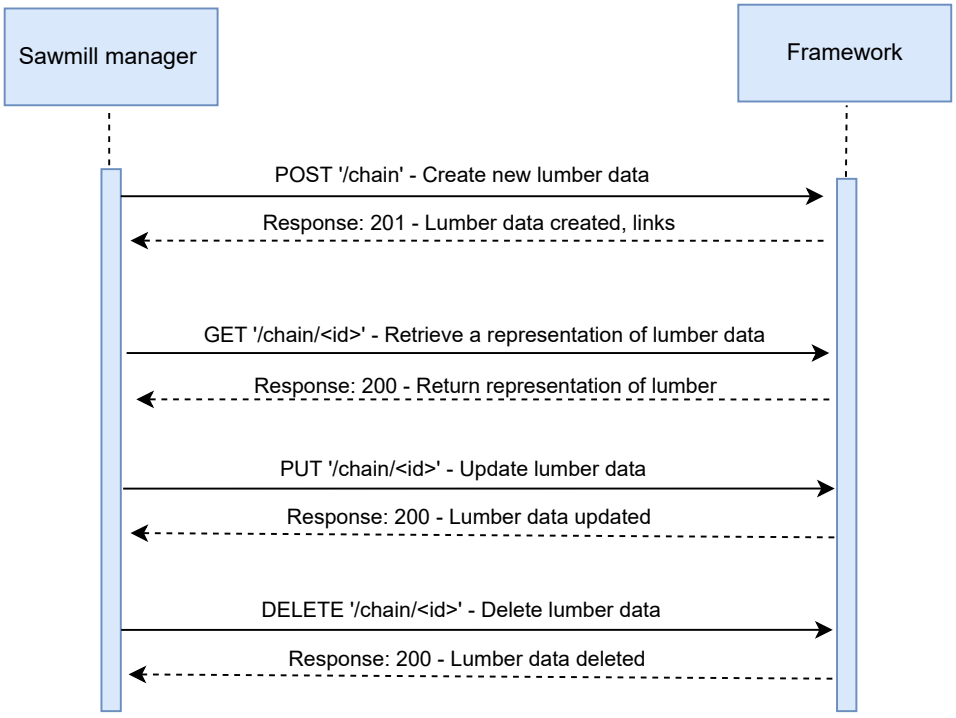

Figure 3.6: Execution flow of sawmill actor in the framework.

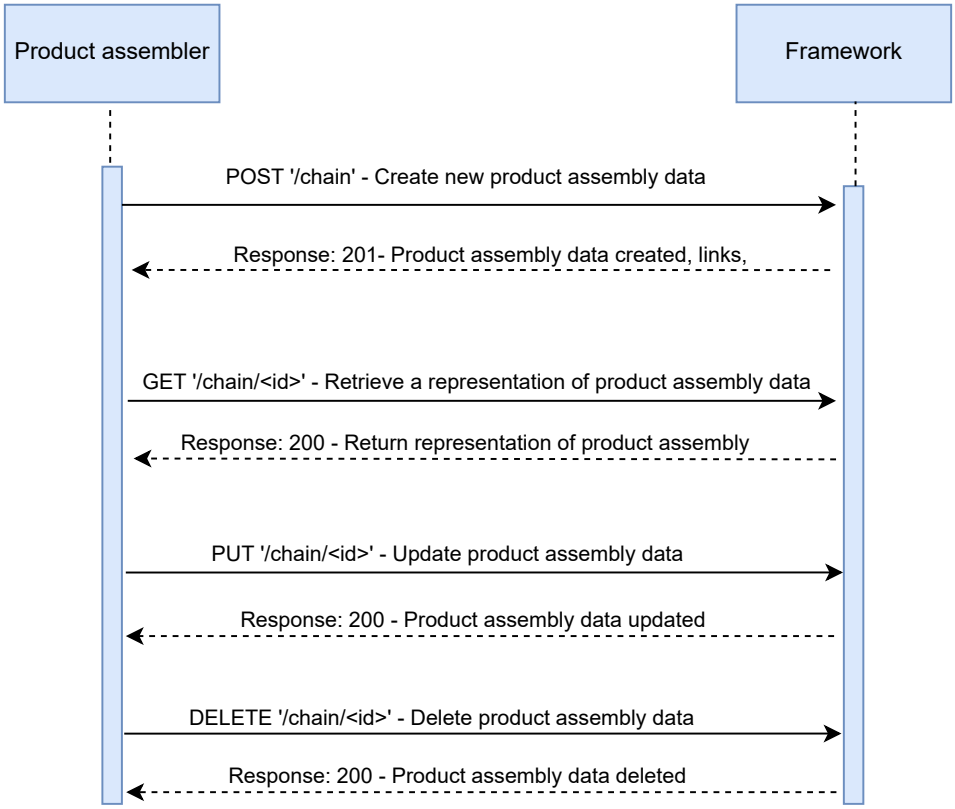

Figure 3.7: Execution flow of product assembler actor in the framework.

Figure 3.8 shows the execution flow between a product seller actor and the framework. The product seller actor calls the `/chain` resource with method `'POST'` to write the product data in the framework. In the response, it receives the links that allow them to perform read action on the data. To do so, it use the GET link that return the representation of the product data(`/chain/<id>` resource, method `'PUT'`).

Accordingly, customer will call the `/chain/<id>` resource with method `'GET'` to retrieve the representation of product data and log as presented in Figure 3.9. In a successful case, the framework returns the representation of the product and log data. This way, customer can use product id to trace the product origin by using reference ids. In the following subsections, we discuss the components of our framework.

3.3 Framework Components

In the following, we discuss the components of our framework that support privacy-aware decentralized data storage and management, multi-level data access, and data security.

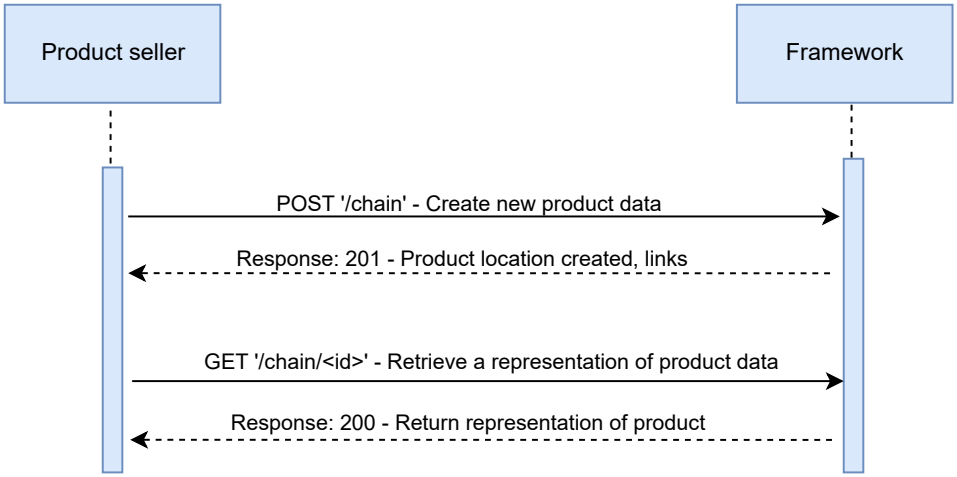

Figure 3.8: Execution flow of product seller actor in the framework.

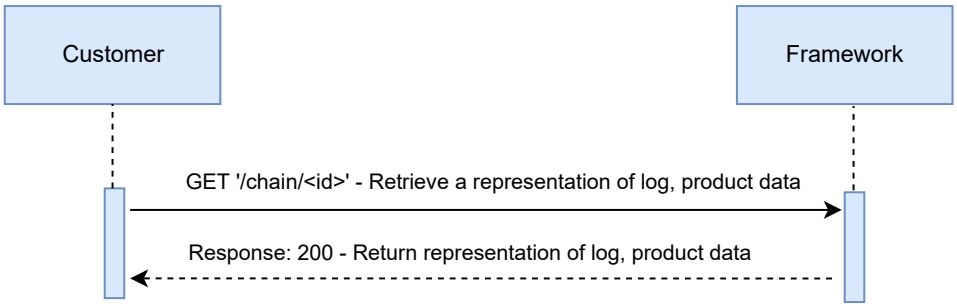

Figure 3.9: Execution flow of customer in the framework.

3.3.1 Access Control Ontology Component

In our access control ontology component, we extend the Role-based Access Control (RBAC) model with an ontology to ensure data privacy by preventing unauthorized access to the data in the decentralized storage framework. The RBAC model contains the following four parameters: user, role, resource, and object. The RBAC users are actors within the application. A role is the application's function that enables access to the resources according to the allocated permissions. A Permission is an authorization to access data within the same application [24].

For our WSC use case, we presented the users, roles, resources, and permissions in the following.

In our framework, we define the RBAC users according to our WSC scenario.

- **Users:** RBAC users are the actors that we define according to our WSC scenario. Therefore, users are as follows: Alice, Bob, David, John and Eric.
- **Roles:** We define the roles that assign to the actors to perform different operations on the data according to their assigned permissions. We define the

following roles: forest manager, transporter, sawmill manager, product assembler, product seller, and customer. Each actor has different permission to access the data according to their role. For example, with RBAC, we define that the product assembler actor is a data owner and has permission to perform write, read, update, and delete actions on their data, while the product seller actor is a business partner of the product assembler actor and can only read their data without updating and deleting it.

- **Resources:** The framework enables actors to access a resource according to their roles and permissions. Each data unit stored at a specific location on the DHT is a resource and has a blockchain hash key pointing to it.

- **Rules and Permissions:** Rules give restriction criteria to gain access to resources. Rules provide permissions to restrict actions to the data. Our CheckPermission (actor, role, verb) function called by the main component is responsible to verify that if the "current actor" is allowed to perform write, read, update, and delete actions on the data or resource according to their role and HTTP verb permissions.

  We develop detail discussion of our access control ontology component including rules and permissions in Chapter 5.

3.3.2 Blockchain Component

In our framework, we use a blockchain component to manage metadata and the DHT key (a hash pointer refers to the data in the DHT) of the encrypted data. Our blockchain component is comprised of block transactions, consensus mechanism, and metadata structure. We develop a detailed discussion of the blockchain component including the proposed metadata structure in Chapter4. In the following, we discuss the block transactions and consensus mechanism.

- **Block and Transactions:** Each block of the blockchain includes a block header, consensus signature, hash of the previous block, validated metadata, and data hash key. Each block includes a unique hash value, that guarantees the integrity of the entire blockchain including the first block known as the genesis block to the last block [126]. Our framework allows actors to take the copy of the blockchain's transactions by calling the `/chain` resource with method `'GET'` if any other actor will be available on the network, if not then, the genesis block will be generated on the blockchain.

  In the blockchain, each block may have many transactions [126]. Each new transaction is broadcast across nodes of the network to verify it. Miners used a proof of work consensus mechanism to validate the transaction and then the verified transaction is added to the block of the blockchain. After storing metadata and DHT key in the block, our framework enables data owners to read or access their data by calling (`/chain/<id>, method 'GET'`). Then, the data owner can perform different operations such as read, update, and delete on their stored data.

- **Consensus Mechanism:** It plays an important role in our blockchain component, where peers agree on the same copy of the data in the network. It guarantees that all nodes of the blockchain have the same copy of the data. Furthermore, it prevents the attacker nodes to make any changes in the data on the network. Our blockchain component uses a proof of work mechanism for transactions validation and creating a new block to the blockchain. To do so, proof of work needs miners to solve hard calculations that must be accepted by other miners on the network. Once miners validate the block transactions, they receive a reward such as a new coin. After completing the block validation process, a validated block inserts into the chain. The proof of work mechanism has an advantage in terms of preventing malicious nodes to compromise more than 51% hashing power of the blockchain. In addition, it is easy and quick to verify the proof.

3.3.3 DHT Component

In the framework, we use off-chain (key, value) storage called the DHT component to store the encrypted data of the actors. We use the Kademlia library to implement the DHT component of our framework. Authorized actors can write, read, update, and delete their data associated with the DHT key. Actors' data are replicated across the nodes on the network to eliminate the risk of data loss and a single point of failure. Our solution records the date and time of each new transaction to keep the track of the data. In Chapter 4 we discuss the process to write data on the DHT.

3.3.4 Encryption Manager Component

Our framework design is flexible to choose multiple types of encryption mechanisms to enforce security on the data. In the proposed solution, the encryption component enables authorized actors to choose between multiple types of encryption methods such as symmetric encryption or asymmetric encryption to write and read data in a decentralized storage framework.

Symmetric encryption is useful where a large amount of data requires to be stored and is needed to be shared with many actors. Symmetric encryption mechanism contains one key to encrypt and decrypt the data. If an actor as a data owner selects the symmetric encryption then data will be encrypted with a symmetric key. Our encryption component is responsible to encrypt this symmetric key with the data owner's public key to ensure the security of the symmetric key. Both encrypted key and encrypted data will be stored on the DHT component. If an actor such as transporter requests to read the forest manager's data, then the data owner such as the forest manager would decrypt this key and then will be encrypted this key with the data requester's public key. This way only the data requester can access it to decrypt and read the data.

On the other hand, asymmetric encryption is useful for a small amount of data that are not required to share with other actors. Asymmetric encryption mechanism is based on two keys known as a public and private key. A public key is available to everyone for data encryption while the private key is only known to the owner of the key for data decryption.

Let us consider our running example, if an actor as a forest manager selects the asymmetric encryption method, then forest manager actor's data will be encrypted with their public key. Later, the only forest manager actor can decrypt or access their encrypted data using the corresponding private key. Asymmetric encryption mechanism makes sure that only the forest manager can access their data. In our case, asymmetric encryption is required to identify actors, and symmetric encryption is used to share the data.

3.4 Implementation and Discussion

We used Python 3 to implement the REST APIs. We used JSON library[1] to receive the response message. We used a Python-based Web framework such as Flask[2] to manage HTTP requests and REST APIs. We used Flask due to the following reasons:

- It is simple and easy to learn and use. Its documentation helps us to use the application.
- It is an open framework and easily accessible under an open-source license.

We evaluated the proposed REST APIs on 64-bit Microsoft Windows Operating System, with 16GB of RAM, Intel core i-7 processor system with a clock cycle of 1.80 GHz.

In the following, we provided the comparative analysis of our APIs with the Hyperledger APIs [3]. For comparison, we choose Hyperledger because it is widely adaptable solution to the industry.

The Hyperledger framework relies on JSON_RPC (Remote Procedure Call) which is a different protocol. JSON_RPC does not allow the usage of a generic client because a software client must know the server's code or method names, and parameters to communicate with it. If some modification has occurred on the server's side then the client must know about them and update accordingly, making update management a tedious task, especially when the number of clients is large. In contrast, we used REST APIs that promote uniform interfaces, thus enabling a generic client which are easy to manage because a client does not need to know about method names and parameters. In addition, we used HTTP verbs and status codes according to the REST principle.

The Hyperledger framework do not follow HTTP verbs completely as it does not use PUT and DELETE HTTP verbs. In contrast to this work, we used all HTTP verbs (such as POST, GET, PUT, DELETE) to follow the principle of REST.

3.5 Chapter Summary

In this chapter, we give a global overview of our framework and its components, together with the Web APIs that support peer and data management. We designed

[1] `https://docs.python.org/3/library/json.html`
[2] `https://flask.palletsprojects.com/en/2.0.x/`
[3] `https://github.com/hyperledger/fabric/blob/v0.6/docs/source/API/CoreAPI.rst/`

simple RESTful APIs to allow authorized actors to write, read, update and delete data in a decentralized fashion. We provided a comparative analysis of our APIs with existing Hyperledger framework APIs and showed where our design choices differ. As our solution provides for data security and privacy through multi-level data access control, our framework integrates the blockchain with DHT, ontology-based access control, and different types of encryption mechanisms. Therefore, we structured it into a set of components that interact with each other upon request. We described the functionality of each component of our framework. We presented actors' interactions with the framework using RESTful APIs, and we showed how our solution allows actors to communicate with other actors through HTTP calls and realize the tasks defined in our use case. We discussed the implementation details of the proposed solution, that we kept as generic as possible for better re-usability in other scenarios and application domains.

Chapter 4

Decentralized Mutable Data Storage

The Results of this chapter are published in the following articles:

- Aslam, S., Mrissa, M.: Mutable and Privacy-aware Decentralized Ledger for Data Management in Wood Supply Chain Environments. *InnoRenew CoE International Conference (IRIC2021), 2021*
- Aslam, S., Bukovszki, B., Mrissa, M.: Decentralized Data Management Privacy-aware Framework for Positive Energy Districts. *Energies, 14(21), 7018, 2021*
- Mrissa, M., Tošić, A., Hrovatin, N., Aslam, S., Dávid, B., Hajdu, L., Krész, M., Brodnik, A., Kavsek, B., : Privacy-aware and Secure Decentralized Air Quality Monitoring. *Applied Sciences, 12(4), 2147, 2022*

One paper has been submitted and is currently under review:

- Aslam, S., Mrissa, M.: A Framework for Privacy-aware and Secure Decentralized Data Storage. *Computer Science and Information Systems (ComSIS), June 2022*

4.1 Introduction

Our motivating scenario discussed in Chapter 1 highlights the need for a decentralized solution to manage data at each point of the WSC. Indeed, the WSC actors need to store and update the characteristics of data describing the wood as it is being processed, transported, and assembled into a final product. For example, the location of each item changes from one step to another. Over time, a large quantity of data items related to the produced goods must be stored, their security and integrity must be guaranteed, and updates need to be realized when needed. On the one hand, typical solution for decentralized data storage provide some guarantee in terms of security and reliability, and on the other hand, blockchain provides the necessary support to build decentralized trust, but is not designed to handle large amounts

of data nor updates, which makes a combination of both technologies an interesting choice to meet the requirements of the WSC.

In this chapter, we explain how we combine blockchain and Distributed Hash Table (DHT) into a secure decentralized data storage solution that features flexible encryption, meaning that different encryption methods can be chosen at run-time to ensure data protection. Our solution relies on the RESTful APIs that support our framework discussed in Chapter 3. It ensures actors' trust by storing metadata on the blockchain, as an immutable record of the data operations. We propose a metadata structure that extends existing work [7] by providing DHT key, previous pointer, data id (RFID_number), data owner's id, date and time of each piece of data. The elements that compose our metadata structure allow the blockchain to keep record of all the operations performed on the data storage, through a system of pointers that actually refer to the index of the data in the DHT. In order to exploit this metadata structure, we developed algorithms that support data updates and traceability. Upon data update, a new transaction is created on the blockchain, that contains a reference to the previous block describing that piece of data. The WSC actors need to trace all id's that are used to compose the certain product. To do so, we rely on a traceability algorithm that allows the actors of our scenario to trace the data in the chain and verify product origin.

In the following sections, we discuss the details of the proposed metadata structure. Then, we describe an algorithm for the data write on secure decentralized storage. After that, we discuss the management of data operations including data update, read, and delete on a decentralized ledger. We discuss the detail of the proposed traceability algorithm. Then, we provide implementation details with security analysis and performance evaluation of our proposed solution. A conclusion summarizes the results obtained in this chapter.

4.2 Metadata Structure

In order to maintain the trust of the WSC actors, our framework needs to record the date and time of each action on the data, so that actors are able to keep track of the data. We propose a metadata extension inspired by the authors in [7], to manage access restrictions on the data. Therefore, our solution encrypts the actor's data through encryption mechanisms (illustrated in Algorithm 2), and sent this encrypted data to the DHT. We store metadata and the DHT key of this encrypted data on the blockchain. Our metadata structure on blockchain contains DHT key, previous pointer, data owner's id, date, time, and RFID_number as depicted in Figure 4.1.

4.3 Algorithm for the Data Write Operation

In this section, we present the workflow that our system follows when a write operation is requested from the API. Let us consider our running example: an actor has a role "forest manager" and issues an HTTP POST request to the API to store data about available wood, as presented below:

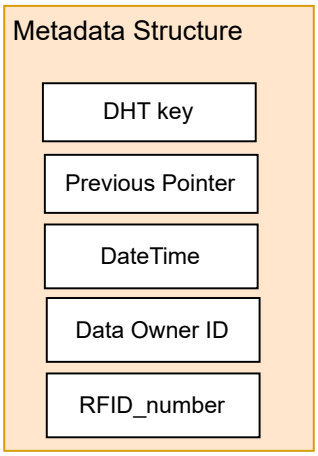

Figure 4.1: Metadata structure on the blockchain

```
{
    "id": "RFID_number",
    "datetime": "2022-01-15, T-13:15:20.45+01:00",
    "woodtype": "maple",
    "location":
    {
        "lat": "38,3951",
        "long": "-77,0364"
    }
}
```

Algorithm 2 shows how our system handles the data write operation. Lines 1-2 check if the current actor is authorized to store their data or not according to their role and HTTP verbs permissions (more details about that part in Chapter 5). This algorithm allows the authorized actor to choose between multiple types of encryption methods (`em`) to encrypt their data before storing it on the decentralized platform to enforce security on the data, as shown in lines 3 to 9. The authorized actor has an option to choose asymmetric to encrypt the data if it is not required to share the data with many other actors. Asymmetric encryption involves two keys called public and private keys. Lines 3-4 show if an actor chooses an asymmetric encryption method then data will be encrypted using the public key of the data owner. Later, an authorized actor can use their private key to decrypt the data.

The authorized actor also has a choice to select symmetric to encrypt the data, if he wants to share their data with other actors. Symmetric encryption uses a single key for data encryption and decryption. We use the Fernet[1] library to generate the symmetric key. If the data owner selects symmetric `em`, then data will be encrypted with the symmetric key as represented in line 6. Then, line 7 encrypts this symmetric key (`sk`) with the data owner's public key (`PK`) to protect the key from unauthorized usage and to make sure that only the data owner can access or read this key. Upon

[1] `https://cryptography.io/en/latest/fernet/`

data read request, the data owner will encrypt this symmetric key with the requester's public key to allow the authorized actor to read the data. Line 8 shows the encrypted data and encrypted symmetric key. A DHT key (`dk`) will be generated for this encrypted data (`ed`) on line 10. Our framework stores this `ed` including encrypted key (`ek`) on DHT (line 11). Then, on line 12, the function `FindLastTransaction` returns previous pointer (`pp`) if it exists otherwise it returns 0. Line 13 stores the `dk` and metadata on the blockchain. The metadata contains `pp`, datetime, data owner id (`DOID`), and data id (`rfid_number`). We develop the detail of previous pointer in section 4.4.

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: boolean value: if true we use asymmetric and if false then it is symmetric

▷ PK: public key of data owner (constant)
▷ DOID: id of the data owner (constant)
▷ ed: variable to store the encrypted data, encrypted symmetric key
▷ sk: symmetric key (variable)
▷ encrypd: variable to store the encrypted data using sk
▷ ek: encrypted 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
```

4.4 Management of Data Operations

The proposed metadata structure connects the different values attached to a specific piece of data to maintain its history. Let us explain using our running example, if an actor as a forest manager writes log data such as:

```
{
    "id": "RFID_number",
    "resource": "log",
    "woodtype": "maple",
    "datetime": "2022-02-11, T-13:15:20.45+01:00",
    "location":
    {
        "lat": "38,3951",
        "long": "-77,0364"
    }
}
```

Then, the new transaction that contains a DHT key and data id such as RFID_number of this data will be stored in the metadata (illustrated in Algorithm 2). Later the data owner (such as the forest manager actor) can perform different operations (such as update, read and delete) on their data for the specific RFID_number. Our solution allows only authorized actors to perform these operations on the data using HATEOAS links discussed in Chapter 6. We manage actors' write, read, update, and delete actions on data through a proposed access control ontology discussed in Chapter 5. Later, an authorized actor wants to update some part of the data, then he will write a new data value against the same RFID_number such as:

```
{
    "id": "RFID_number",
    "resource": "log",
    "woodtype": "oak",
    "datetime": "2022-12-16, T-16:11:10.45+01:00",
    "location":
    {
        "lat": "38,3951",
        "long": "-77,0364"
    }
}
```

In this case, new metadata will be created on the blockchain that contains a new DHT key of this updated version of the data and the previous pointer that points to the previous version of the data that is stored on the DHT. Our metadata structure also records the new datetime of the updated version of the data. In case of data update, the DHT key of the previous version of the transaction becomes the previous pointer which is stored in the new version of the transaction in the metadata. Later, if the forest manager actor wants to access his previous version of the data then he

will use the `FindLastTransaction (did)` function that returns the latest version of the data against this `did` (data id as RFID_number) including DHT key of new data and previous pointer of the updated data. This way an actor can access their update history.

Upon data read, our solution enables the authorized actor to decrypt and access their data in a decentralized platform. If data is encrypted using the data owner's public key then a data owner can decrypt this encrypted data using his private key. In case, if data is encrypted using a symmetric key then the authorized actor will first decrypt the symmetric key with their private key and then this decrypted symmetric key will use to access the data which is stored on the DHT.

Similarly, the delete data operation works as an update where a new DHT key generates with a NULL value. If an authorized actor wants to delete their data against a specific RFID_number, then a new transaction creates on the blockchain that contains a new metadata structure. This metadata contains a new DHT key that points to the Null value on the DHT.

4.5 Traceability Algorithm

In this section, we explain the process to trace the data in the WSC as presented in Algorithm 3. Let us develop an example, a customer buys a product such as a table and he wants to trace where this product comes from. Then, he will use the product id as a data id (such as RFID_number) to trace their origin. The proposed algorithm allows actors to identify the origin of the final product based on the data id's references.

In Algorithm 3, line 1 initialize item (`I`) to empty set. In line 2, the `did` is an RFID_number of the item that goes through the wood supply chain, and data (such as location) of this item changes against the same `did`. Therefore, on the blockchain, we can have many transactions against this `did`. Whenever the location of the item changes then metadata of this `did` is stored on the blockchain, and data is sent to the DHT. Our `FindLastTransaction` function finds the last transaction from this `did`, which is a RFID_number (line 2). For example, if we have `did` of the log then it returns the last transaction of this log. Now, we can access the metadata from this transaction `t` and can request to access the data from the DHT. In the metadata, we have a DHT key that points to the data stored on the DHT.

After that, it checks if the current data requester has permission to read the data or not according to their role and HTTP verbs permission `'GET'` on line 3. We define access control rules that limit unauthorized access to the data stored on the DHT (more details in Chapter 5). In line 4, the function `GetReferences` takes the `t` and extract the `did` of the items. Then, it finds the previous references of this `did`. For example, if the input `did` is product id then it gets the previous references such as RFID_number of the lumbers.

Line 5 checks the list of references (`I`) is not empty. Then, it iterates items in the item list e.g it checks lumbers in the list (line 7). Line 8 add items (e.g lumbers references) in the output list (`O`). The `Traceability` function takes `i` such as lumber as input and call recursively to find log and return them in output list (`O`) in line 9.

Then, line 11 returns the final output (O). In case the list is empty then it means we are applying traceability of the log and the log does not have any previous reference so then it returns nothing (line 14).

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
▷ I: items list (variable)

```
1:  I ≠ ∅                                              ▷ Initialize I to empty set
2:  t ← FindLastTransaction(did)          ▷ Find the last transaction from this RFID
3:  if CheckPermission (actor, role, v) then   ▷ Verify permission of the requester
4:      I ← GetReferences (t)                          ▷ Get the RFID of the items
5:      if I ≠ ∅ then                                          ▷ If there are items
6:          O ← ∅                                              ▷ Initialize output
7:          for each i ∈ 𝓘 do                                      ▷ For each item
8:              O.append(i)                          ▷ Add it to the output list
9:              O.append(Traceability(i))              ▷ Find item recursively
10:         end for
11:         return O                                    ▷ Return the final output
12:     end if
13: end if
14: return ∅          ▷ If the list was empty then return nothing, end of the recursion
```

4.6 Results and Implementation

In the following, we discuss the results of our decentralized data storage and update solution. Section 4.6.1 presents the experimental setup and implementation details. In section 4.6.3 we discuss the qualitative security analysis and quantitative performance evaluation.

4.6.1 Experimental Setup and Implementation

We used Python 3 to perform all experimental processes because it is a scalable and dynamic language. We used an open-source blockchain library[2] to implement the blockchain component. We used the blockchain library to achieve consensus on a distributed network, generate new transactions and blocks, and mine them through a consensus mechanism (proof-of-work). We implemented a DHT component using a Kademlia library[3]. The DHT is used to input and retrieves data link with a hash key on the peer-to-peer network. We used the cryptography RSA library to create encryption/decryption keys and sign/verify signatures.

[2] https://github.com/satwikkansal/python_blockchain_app/tree/ibm_blockchain_post
[3] https://github.com/bmuller/kademlia

We evaluated our framework components on a Windows 10 operating system with 16 GB of RAM. The Central Processing Unit (CPU) architecture used in the 64-bit operating system was x64 Core i7 processor system with a clock cycle of 1.80 GHz.

In the following, we discuss the security analysis of our proposed solution, which is based on security properties and attacks. Moreover, in the performance evaluation, we computed time cost and scalability.

4.6.2 Security Analysis

In our solution, to protect the data from unauthorized access, we offer multiple types of encryption to enforce security on the data. Our solution allows data owners to own and control their data in a decentralized platform. Before sharing data with other actors, we encrypt the data with the requester's public key to ensure data protection from an malicious actor who might try to read or access the data. Our framework encrypts the data before storing it on the DHT. Even if an unauthorized actor obtains access to the DHT nodes, then only cipher-texts are visible to them and they could not gain any information about the data. Furthermore, we used blockchain and DHT due to their decentralized and distributed design. This would eliminate the risk of the single point failure issue, and ensure data replication and availability.

Our framework design addressed the following main security properties:

- To achieve confidentiality, we use asymmetric and symmetric encryption.
- Our framework encrypts the data using the data owner's public key to achieve integrity such that the data can only be decrypted or updated using the corresponding private key.
- Our framework achieves availability using the access control ontology.
- To achieve non-repudiation, we store metadata in the chain and maintain data traceability. Our proposed metadata structure contains the date and time of data storage, due to this the data owner could not deny the data's existence on the chain.

We evaluated the security of our solution under the following attacks:

Modification attack: A modification attack happens when a malicious actor tries to change the data content. Our solution design enables data owners to encrypt the data with their public key and manage the corresponding DHT key on the blockchain. The proposed metadata design keep the track of data entry date and time to identify the modification in the data. A malicious actor cannot update the data because data can only be decrypted using a data owner's private key that is only known to the data owner.

Spoofing attack: A spoofing attack occurs when an attacker uses the ID of someone else and might try to access the data. In our solution, an attacker could not spoof the ID of other actors because they could not spoof its private key. In our solution, each actor has their secrete private key that will not be shared with anyone.

Linking attack: In a linking attack, a malicious actor tries to link different transactions or data with the corresponding public key. To prevent this attack, our framework design uses different types of encryption mechanisms such as data owner's public key, symmetric key, and requester's public key. Our proposed encryption component is responsible to generate public, private, and symmetric keys on run time according to the actor's encryption method selection.

To protect the symmetric key from unauthorized access, we encrypt the symmetric key with the data owner's public key and store it on the DHT. Later, only the authorized actor can use this symmetric key to access their data. This way a malicious actor will not be able to link different transactions to the same public key, because our framework encrypts the data through different encryption mechanisms and public keys.

Eavesdropping attack: An eavesdropping attack occurs when a malicious actor listens to privacy-sensitive information on the network. To prevent this attack, our framework encrypts privacy-sensitive data using the public key of the requester upon data read request. Later, only authorized actors are allowed to access and read the data using the corresponding private key.

Sybil attack: An attack occurs when an attacker attacks data redundancy. Malicious nodes act as many nodes by generating multiple identities in the peer-to-peer network. To protect against this attack, our framework assigns only one identity such as data owner id and public key to each actor. Each time when the data is stored, then the data owner id will be recorded in the metadata to identify the owner of the data.

In case an actor connects and is authorized to perform actions and then becomes malicious later and disconnected. In this case, the data of the disconnected actor or node will not be available anymore. In our framework, actors need to be connected or online because data is encrypted with their public key and only they can decrypt it and allow other actors to read it upon request. This problem is not currently addressed as the purpose of the DHT is to not replicate the data on all nodes, the tolerance to a 51% (or more) attack is still a problem in general as in this case. This opens the door for future work to design a solution to mitigate this issue.

4.6.3 Performance Evaluation

We obtained results based on the evaluation parameters, such as time consumption and scalability. We evaluated the time consumption of our solution based on the following parameters, such as: check permission of an actor, data encryption, and decryption using the asymmetric or symmetric method, DHT access, and blockchain access. We computed the time cost while performing data write, data update, data read, data delete and traceability.

Figure 4.2 and 4.3 outline the time cost for both asymmetric encryption and symmetric encryption. We compared the results of using asymmetric encryption and symmetric encryption.

As we can see from the results, for write data operation, blockchain access time for symmetric encryption is less than the blockchain access time for asymmetric encryption. The DHT access time to write data for asymmetric encryption is less than

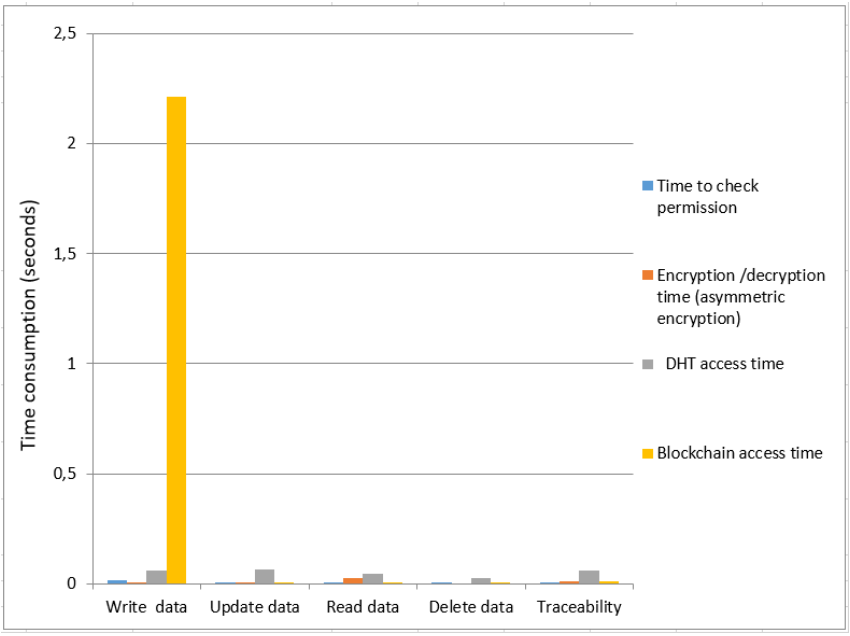

Figure 4.2: Time overhead using asymmetric encryption

the DHT access time of symmetric encryption. Total time to check permission and encryption/decryption does not show much difference for both symmetric encryption and symmetric encryption while performing a data write operation.

For update data, the overall DHT access time for symmetric encryption is slightly higher than the total DHT access time for asymmetric encryption. The total time to check permission, encryption/decryption, and blockchain access time is not much affected for both figures 4.2 and 4.3.

For data read and delete, the total time to check permission, encryption/decryption, blockchain access time, and DHT access time does not show much difference for both symmetric encryption and asymmetric encryption.

For data traceability, the DHT access time for asymmetric encryption is slightly higher than the DHT access time for symmetric encryption. Total time to check permission, encryption/decryption, and blockchain access time is not much affected for both asymmetric and symmetric encryption.

Moreover, we tested the scalability of our solution with an increasing number of actors such as 1, 50, 89, and 110 and get a reasonable performance with 110 actors. We have designed our framework in the context of Positive Energy Districts (PED), where we model a different number of actors depending on the energy community, hence explaining the chosen numbers of actors 50, 89 and 110 as they relate to the proportion of actors for PEDs. Details of these numbers and the associated research are given in the published paper.

We tested our prototype 100 times for each operation including write data, update data, read data, delete data, and traceability, and then calculated the average time, Standard Deviation (SD), min, and max values in seconds for accurate results of operations. Detailed results statistics are summarized in Table 4.1. We computed

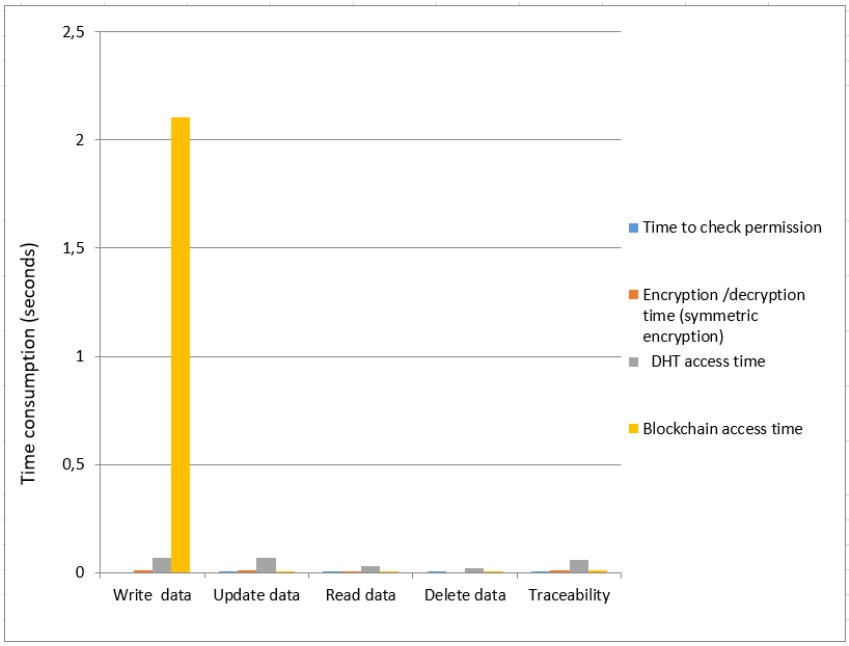

Figure 4.3: Time overhead using symmetric encryption

the average time consumption of our solution with an increasing number of actors as presented in Figure 4.4. As we can observe from Figure 4.4 and Table 4.1, in the case of 1 actor, write data has an average of 0,5712 seconds and update data takes an average time of 0,0024 seconds. Data read gives a SD of 0,0113 seconds and delete data gives a SD of 0,0012 seconds. The traceability data operation provides an min value of 0,0112 seconds and max value of 0,0312 seconds.

For the case of 50 actors, the write operation takes an average of 0,6068 seconds which is slightly less than the average time to write data with 89 actors and 110 actors. The update operation provides an average time of 0,0052 seconds which is slightly higher than the average time to updates time with 1 actor. The delete operation gives an average time of 0,0013 seconds which is slightly less as compared to 1 actor. The traceability algorithm provides an average time of 0,0345 seconds which is less than the average time for 89 and 110 actors.

In the case of 89 actors, the write operation takes an average of 0,6305 seconds which is slightly higher than the average time to write data for the case of 1 actor and 50 actors. The update operation has an SD of 0,0046 seconds and delete data provides an SD of 0,0021 seconds. The read operation gives an average time of 0,0623 seconds which is higher as compared to case of 1 and 50 number of actors. The delete operation provides min value of 0,0023 seconds which is slightly close to the min value for 110 actors. The traceability data operation has a min value of 0,0346 seconds and the data read provides a min value of 0,0572 seconds.

Similarly, with the number of 110 actors, the write operation takes an average time of 0,6653 seconds and the read operation has an average of 0,0644 seconds. The update gives a min value of 0,0055 seconds which is close to the min value for the number of 89 actors. The average time to update operation does not show much

Table 4.1: Detailed results under different number of actors

Number of Actors	Data Operations	Average Time	St Deviation	Min	Max
1	Write data	0,5712	0,4321	0,4635	0,6564
	Update data	0,0024	0,0021	0,0022	0,0064
	Read data	0,0254	0,0113	0,0124	0,0456
	Delete data	0,0014	0,0012	0,0013	0,0234
	Traceability	0,0201	0,0101	0,0112	0,0312
50	Write data	0,6068	0,5427	0,5846	0,7653
	Update data	0,0052	0,0024	0,0044	0,0097
	Read data	0,0471	0,0312	0,0336	0,0556
	Delete data	0,0013	0,0010	0,0012	0,0326
	Traceability	0,0345	0,0232	0,0246	0,0472
89	Write data	0,6305	0,5342	0,5601	0,8541
	Update data	0,0057	0,0046	0,0053	0,0071
	Read data	0,0623	0,0531	0,0572	0,0843
	Delete data	0,0024	0,0021	0,0023	0,0032
	Traceability	0,0436	0,0334	0,0346	0,0521
110	Write data	0,6653	0,5451	0,5542	0,7792
	Update data	0,0057	0,0053	0,0055	0,0072
	Read data	0,0644	0,0531	0,0552	0,0712
	Delete data	0,0032	0,0022	0,0024	0,0043
	Traceability	0,0478	0,0401	0,0421	0,0645

difference as compared to the average time to update operation for 50 actors. The read operation provides a max value of 0,0712 seconds which is less as compared to the max value of read data for the number of 89 actors. The delete operation has a min value of 0,0024 seconds which is close to the min value for 89 actors. The traceability algorithm gives an average time of 0,0478 seconds which is higher than the average time for 89 actors.

We can see from the experimental results that our solution is scalable to handle many actors at the same time. The results demonstrate that, with an increasing number of actors, each data operation (such as write, update, read, delete) gives an average time less than 1 second. Therefore, we can conclude that our solution gives a reasonable time overhead.

4.7 Chapter Summary

In this chapter, we detailed our decentralized data storage and update solution. We proposed a data write algorithm that combines blockchain with DHT to allow for secure data storage. Our algorithm allows an authorized actor to store the data using RESTful APIs. It enables an actor to choose between multiple types of encryption to secure their data in a decentralized ledger. Our proposed metadata structure ensures trust between actors by recording metadata on the blockchain. To update the data, our pointer management system maintains the history of the values that are stored in the DHT for a single piece of data. The proposed traceability algorithm allows actors to keep track of their data according to our WSC scenario. We illustrated the use of the proposed algorithms to show their applicability with our WSC example.

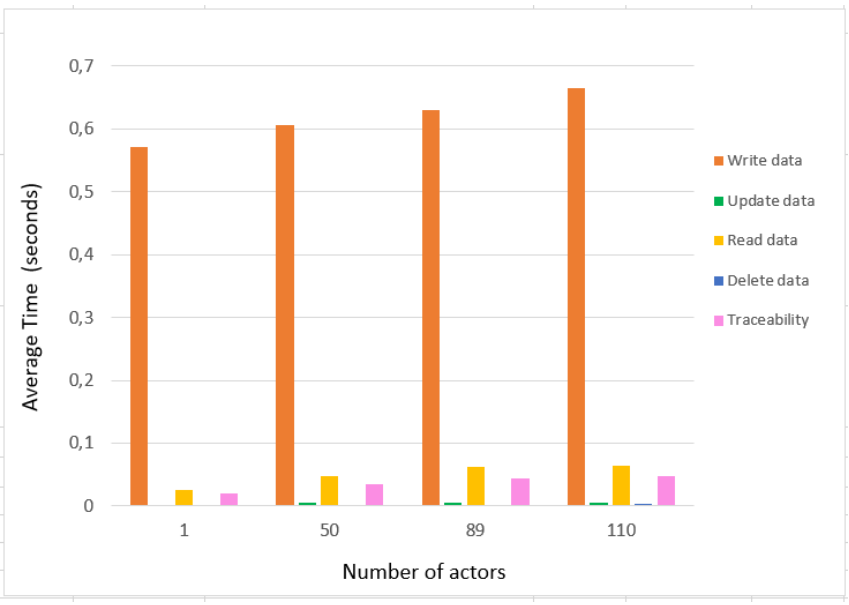

Figure 4.4: Average time consumption under different number of actors

We detailed the security analysis of our solution and evaluated its performance in terms of time cost and scalability. The results show that the proposed solution is scalable to manage many actors at the same time and provides an average time cost less than 1 second while performing data operations. Therefore, our solution achieves an acceptable time overhead.

Chapter 5

Semantic Role-based Access Control

The Results of this chapter are published in the following articles:

- Aslam, S., Bukovszki, B., Mrissa, M.: Decentralized Data Management Privacy-aware Framework for Positive Energy Districts. *Energies, 14(21), 7018, 2021*
- Aslam, S., Bukovszki, B., Mrissa, M.: Multi-level Data Access Control in Positive Energy Districts. *Sustainability in Energy and Buildings 2021 (pp. 553-565). Springer, Singapore, 2021*

5.1 Introduction

As blockchain stores data publicly by design, the privacy requirements of data need to be handled with an additional component that integrates with blockchain in a way that it preserves it operation, and at the same time in a way that guarantees fine-grained privacy management, meaning that it should be possible to control what data is made available to whom and with what rights.

Typically, the Role-based Access Control (RBAC) model is useful to define and enforce permissions to access data. However, RBAC is static and unable to adopt changes such as adding new roles, objects, and permissions at run time. In addition, the RBAC model does not handle complex relationships between actors that go beyond a hierarchical representation of roles, to enable individual relationship between actors. The ideal solution should manage unauthorized access to the data not only according to the roles but also according to the individuals. Our contribution builds on the literature review provided in Section 2.3.3 of Chapter 2 where we highlight the limitations of existing solutions. In a summary, most existing solutions do not consider the non-hierarchical relationships between actors. In addition, most of the solutions do not focus on unauthorized data access control in a decentralized ledger. Some other solutions do not define rules for each role and also at the individual level in a single solution. Most of the solutions do not use Web APIs to access the data and do not design rules and permissions using HTTP verbs.

In this chapter, we propose an access control ontology to address these shortcomings. The proposed access control ontology is exploited with our access control component based on the framework discussed in Chapter 3. We integrate an ontology-driven approach with RBAC to support both hierarchical and non-hierarchical relationships. As explained in Chapter 3, our decentralized solution allows authorized actors to access data through Web APIs. Therefore, we structure our access control rules according to the HTTP verbs GET, POST, PUT, and DELETE. We define access control rules to determine if a specific 'role' or 'individual' has permission to access the 'resource' and perform specific 'actions' on it or not. Our solution ensures data privacy by restricting unauthorized access to the data.

We use a simple subset of OWL-DL (Description Logic) because it supports reasoning and allows us to model classes, instances, and properties so that they can support RBAC with non-hierarchical relationships between both classes and instances. The combination of our access control rules and a subset of OWL-DL are necessary to enhance the OWL's expressivity to express actor relationships as well as to handle data access in our decentralized platform [138]. They are sufficient to ensure computational efficiency and we do not need more expressive language such as OWL Full because it does not support full reasoning. Also, it does not guarantee to complete all computations in a limited time [9]. In addition, OWL full allows using the same class as an individual, which increases the complexity.

As an illustrative motivating example, we refer to the wood supply chain (WSC) scenario presented in Chapter 1. We design our solution according to our WSC actors, roles, and resources.

In the following sections, we provide the detail of our access control ontology component including ontology classes, instances, relationships through object properties, semantic description of non-hierarchical relationships, and access control rules. After that, we discuss the implementation and evaluation of our solution, before concluding the chapter with a summary of the results obtained.

5.2 Access Control Ontology Component

In the following, we discuss the detail of our access control ontology component that manages relationships between actors and handles complex permissions for data access. This component includes an ontology that describes the actors, resources, etc., a set of rules and a reasoner that enable inferring permissions from given facts in the ontology.

5.2.1 Ontology Classes and Instances

Figure 5.1 depicts the class hierarchy of our access control ontology. In our work, we define the instances (e.g individuals) of the OWL classes according to our WSC scenario, as follows:

- The 'Thing' class is also known as a superclass, It represents the root of the classes hierarchy that include the sub classes, namely 'Actor', 'Role', 'Resource', and 'SpatialThing'.

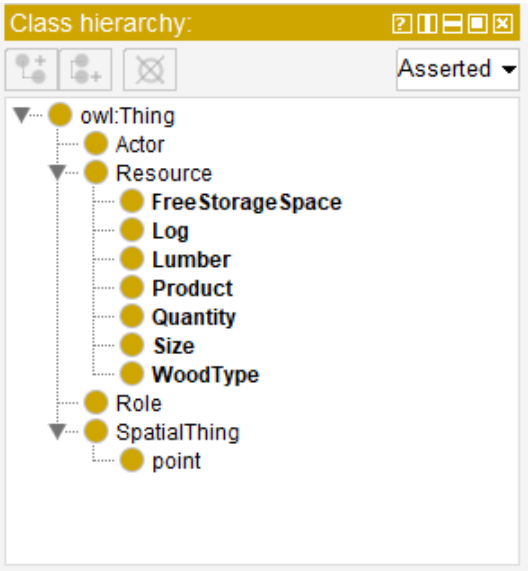

Figure 5.1: Classes hierarchy of the ontology

- The 'Actor' class defines the number of actors in the proposed framework. For the sake of simplicity we defined instances of the 'Actor' class with usual names such as Alice, Bob, David, Eric, and John. In our scenario, actors are actually company representatives who participate in the WSC, they connect to each other through our solution and can have one or several roles in the WSC.

- The 'Role' class defines to which actor and permission should be assigned. We define the instances of the 'Role' class as follows: ForestManager, Transporter, SawmillManager, ProductAssembler, ProductSeller, and Customer. Our WSC actors can perform different actions on the data based on their roles and permissions.

- The 'Resource' class represents the data schema parameters such as log, lumber, free storage space, product, quantity, wood type, and size. The actual data values of these parameters are stored on the DHT component. Authorized actors can access these resources according to their assigned role and permission.

- The 'SpatialThing' class has a further subclass such as 'Point' to define a geographical location. Instances of the 'Point' class have 'Latitude' and 'Longitude' attributes to represent the geographical location of the resources that we manipulate in the WSC. To do so, we reused the geo ontology from the W3C semantic Web interest group[1]. Let us explain through an example, the actor has a role transporter who picks up a log in Izola and delivers it to Koper. We assume that wooden logs, lumber, and products are monitored and traced from the start until the end of the WSC with RFID chips that are integrated into the trees and then into the intermediary and final products.

[1]https://www.w3.org/2003/01/geo/

5.2.2 Relations using Object Properties

We use object properties to define the relationship and semantics between the classes, so that they apply to their instances as well. As depicted in Figure 5.2, we use the HTTP verbs to define the permissions of the 'Role' such as hasPostPermission, hasGetPermission, hasPutPermission, hasDeletePermission. In the following, we define the domain and range that link ObjectProperty to the classes:

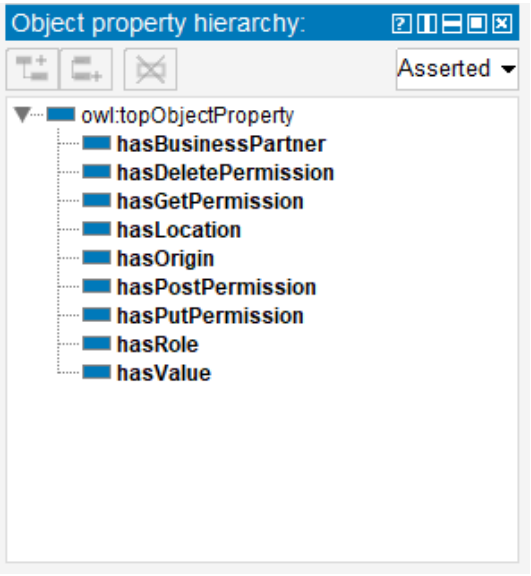

Figure 5.2: The ontology object properties

```
<owl:ObjectProperty rdf:about = hasGetPermission">
    <rdfs:domain rdf:resource = Role"/>
    <rdfs:range rdf:resource = Resource"/>
</owl:ObjectProperty>
```

The ObjectProperty hasGetPermission; its domain is Role and range is Resource, such as Role hasGetPermission to the Resource. In our solution, we assign an RBAC 'Role' to the 'Actor' through hasRole ObjectProperty that allows them to perform actions on the Resource based on their role. In the following, we define the domain and range that connects the property to the class, such as the Actor hasRole a Role. For instance, Alice 'hasRole' ForestManager.

```
<owl:ObjectProperty rdf:about = hasRole">
    <rdfs:domain rdf:resource = Actor"/>
    <rdfs:range rdf:resource = Role"/>
</owl:ObjectProperty>
```

Our WSC actors can request to access or read the data of other actors based on their business partner relationships. Therefore, we define the relationship between actors using object property hasBusinessPartner such as:

```
<owl:ObjectProperty rdf:about=hasBusinessPartner">
    <rdfs:domain rdf:resource=Actor"/>
    <rdfs:range rdf:resource=Actor"/>
</owl:ObjectProperty>
```

The object property hasBusinessPartner; its domain and range is actor, which shows the relationship between two instances that belong to the same class 'Actor' For instance, Bob hasBusinessPartner relationship with Alice. In this example, Alice and Bob are actors in our WSC scenario.

5.2.3 Semantic Description of Non-Hierarchical Relationships

Typical RBAC does not support non-hierarchical relationships because RBAC roles follow a tree structure. Therefore, we add semantics to RBAC to support both hierarchical and non-hierarchical relationships between classes as explained below. Hierarchical relationships enable classes to link to their parent class as shown in Figure 5.1. In OWL, the root class is typically named the 'Thing' class and is the superclass of all other classes in ontology. Non-hierarchical relationships allow classes or individuals to link each other through properties, and do not necessarily follow the class hierarchy. In our solution, we design 'Actor', 'Role', and 'Resource' classes based on the concept of RBAC and define non-hierarchical relationships between them based on the `ObjectProperty` property.

In the following, we show some examples of non-hierarchical relationships between actors using OWL objectProperty.

- We define the business partner relationship between Alice as a forest manager and Bob as a transporter via object property hasBusinessPartner. Therefore, the forest manager actor and transporter actor have a non-hierarchical relationship because the forest manager actor has a 'GET' permission to access the resource of the transporter actor, which is basically outside of their hierarchy.
- Eric as a product assembler is linked to John as a sawmill manager via object property hasBusinessPartner. Therefore, the product assembler actor and sawmill manager actor have a non-hierarchical relationship because the product assembler actor has a 'GET' permission to access the resource of the sawmill manager actor.
- We define the business partner relation between Eric as a product assembler and David as a product seller. Therefore, the product assembler actor and product seller actor have a non-hierarchical relationship, because the product assembler actor has a 'GET' to the resources of the product seller actor that is basically outside of their hierarchy.

5.2.4 Access Control Rules

RBAC is based on the roles and allows actors to access the data according to their assigned roles. However, it does not allow to access data based on the individuals. There is a need to define the rules at the individual level that allow actors to access

the data according to their relationship with other actors without depending on their role.

Therefore, we write the access control rules in SWRL (Semantic Web Rule Language) language that gives permission to perform actions on the resources (such as data) depending on both roles or individuals. In our work, we define the SWRL rules using HTTP verbs such as GET, POST, PUT, and DELETE to manage authorized access to the data.

In our decentralized framework, data owner maintains their OWL file including these access control rules and this file will not be disclosed to others to prevent security risks. Our access control ontology component verifies if the current role or actor has permission to perform read, write, update, and delete actions on the resource. We achieve access control by increasing the expressivity of OWL through the SWRL inference process and HermiT. SWRL infers new knowledge such as ObjectProperty while executing the rules. To do so, our designed ontology and SWRL rules are transferred to the reasoner. Then, we run the reasoner to reason rules and inferred knowledge, which is sent back to, and enrich the OWL.

In the following, we categorized rules into two levels: class level rules and individual level rules.

- **Class level access control rules:** We define the class level rules based on the roles that are involved in our motivating scenario. In the following, we define the rules to control access to the data.

  - **Rule 1:** A forest manager has permission to make a 'POST' request to their resource wood type.

    ```
    Role (?ForestManager) ∧
    hasPostPermission (ForestManager, ?w) ∧
    WoodType (?w)
    → hasPostPermission (ForestManager, ?w)
    ```

  - **Rule 2:** A transporter has permission to make a 'PUT' request to their resource location.

    ```
    Role (Transporter) ∧
    hasLocation (Transporter, ?p)   ∧
    Point (?p)
    → hasPutPermission (Transporter, ?p)
    ```

  - **Rule 3:** A sawmill manager has permission to make a 'POST' request to their resource lumber.

    ```
    Role (?SawmillManager) ∧
    hasPostPermission (SawmillManager, ?l)   ∧
    Lumber (?l)
    → hasPostPermission (SawmillManager, ?l)
    ```

  - **Rule 4:** A product seller is allowed to make a 'DELETE' request to their resources product.

```
Role (ProductSeller) ∧
hasDeletePermission (ProductSeller, ?r)  ∧
Product (?r)
→ hasDeletePermission (ProductSeller, ?r)
```

– **Rule 5:** A customer has 'GET' request permission to verify the origin of product.

```
Role (Customer) ∧
hasOrigin (?k, ?p) ∧
point (?p) ∧
Product(?k)
→ hasGetPermission (Customer, ?k)
```

- **Individual level access control rules:** In the following, we define the rules between individuals based on their business partner relationship to restrict unauthorized access to the data.

  – **Rule 6:** If Bob 'hasBusinessPartner' relationship with Alice then Bob is allowed to make a 'GET' request to the resource wood type of Alice.

```
Actor (Bob) ∧ hasBusinessPartner (Bob, Alice) ∧
Actor(Alice) ∧ WoodType(?t) ∧
hasGetPermission (Alice, ?t)
→  hasGetPermission (Bob, ?t)
```

  – **Rule 7:** If Eric 'hasBusinessPartner' relationship with John then Eric can make a 'GET' request to the resource lumber of John.

```
Actor (Eric) ∧ hasBusinessPartner (Eric, John) ∧
Actor(John) ∧ Quantity(?d) ∧ lumber(?l) ∧
hasValue (?l, ?d) ∧ hasGetPermission (John, ?l)
→   hasGetPermission (Eric, ?l)
```

  – **Rule 8:** If Eric 'hasBusinessPartner' relationship with David then Eric has a 'GET' request permission to the resource product of David.

```
Actor (Eric) ∧ hasBusinessPartner (Eric, David) ∧
Actor(David) ∧ Product (?r) ∧
hasGetPermission (David, ?r)
→  hasGetPermission (Eric, ?r)
```

The rules discussed above infer the knowledge to identify who has permission and which type of permission to access the resource. In Rule 1, left side is based on the OWL classes such as Role and WoodType, and permission is defined using Object-Property hasPostPermission. We execute Rule 1 and it infers that ForestManager hasPostPermission to the resource 'w' which is wood type as mentioned on the right side. The rest of the rules show the same results. SWRL rules have an advantage in terms of human-readable syntax. Our solution allows authorized actors to add more rules, update, and delete existing rules.

5.3 Implementation and Evaluation

In the following, we detail the implementation and evaluation of our access control ontology solution. We discuss the performance evaluation of the proposed solution in terms of check permission based on the access control rules in Chapter 4.

5.3.1 Implementation

We used Python 3 and Protégé 5.5.0 to implement our access control ontology solution. We model classes, class hierarchy, object properties, instances, and SWRL rules in Protege. We used owlready2[2] library to load OWL ontology in Python. We describe the relationships between actors (who is the partner of whom) and rules to control authorized access to the privacy-sensitive data. We evaluated the proposed solution on a PC with a Core i7 CPU at 1.80 GHz, 16 GB RAM, and Windows 10.

5.3.2 Evaluation

We used the HermiT reasoner to ensure the consistency of our ontology. It also verifies the classification of ontology such as classes and subclasses is correct. During the process of developing our ontology, we continuously used a reasoner to avoid any errors. It highlights the unsatisfiable classes with red color to show the errors. Once the reasoning process is completed, the identified errors in the whole ontology including hierarchy are listed.

In our work, we compared the HermiT reasoning time with other reasoners such as Pellet, Drools, and RACER. Figure 5.3 depicts the average time cost comparison while using different reasoners. As we can see from the figure, the HermiT reasoner takes 163 milliseconds to verify the consistency of our ontology. In the case of the Pellet reasoner, it gives an average time of 214 milliseconds which is higher than the average time of the HermiT reasoner and less than the Drools average time. For the Drools reasoner, it takes an average time of 227 milliseconds which is higher than the average time of both HermiT and Pellet. Similarly, RACER provides an average time of 239 milliseconds which is slightly higher than the Drools average time.

The results demonstrate that HermiT reasoning time is faster as compared to other reasoners. Our experimental results show that all reasoners take an average time of less than 1 second which is acceptable for the end-user.

5.4 Conclusion

In this chapter, we discuss our access control ontology solution. Our solution combines the RBAC with OWL ontology and SWRL rules to manage unauthorized access to the data. The main aim of using the semantic web approach is to support complex permissions and relationships between actors. Our solution is flexible to add more resources, actors, roles, and rules. For instance, a new actor can be easily added as an instance of the actor class without affecting the existing classes, instances, resources, and properties. We discuss the implementation details and evaluate the overhead of

[2]https://owlready2.readthedocs.io/en/latest/install.html

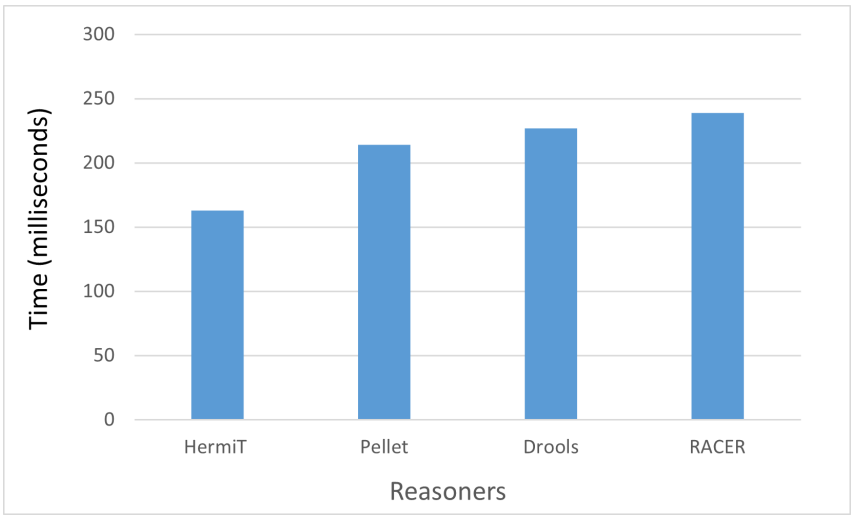

Figure 5.3: Average time comparison between different reasoners

our solution. Our implementation shows that the reasoning times necessary to grant or deny a permission are negligible, under one second for our sample scenario, with different reasoners, and our results allow to recommend a specific reasoner.

Chapter 6

HATEOAS Client with REST APIs

6.1 Introduction

While Representational State Transfer (REST) is an architectural style that plays an important role to design scalable and distributed systems, Hypermedia as the Engine of Application State (HATEOAS) is the constraint that probably receives the least attention. HATEOAS, as its name indicates, and as described in Section 2.3.4 of Chapter 2 promotes using hyperlinks to drive the interactions between a (generic) client and an application.

However, the concrete implementation of the HATEOAS principle is challenging. Typically, the inclusion of hyperlinks within HTTP response messages is the responsibility of the developer of the RESTful service. Such a process is manual and tedious as the links must be added to responses according to specific rules, which require the developer to have a clear overview of the processes the application follows, or participates to. As a result, most Web-based APIs do not implement the HATEOAS constraint, rather limiting their compliance to REST to the other constraints [124,141]. Rather than providing links to responses, developers of Web-based APIs provide API documentation for developers to read (e.g open API) [73]. This situation shows the need to facilitate the implementation of HATEOAS into Web APIs with a new solution.

In this chapter, we propose a simple proxy-based solution that enables HATEOAS navigation. We develop a generic HATEOAS client that supports any REST API. We design a proxy that acts as a middleware between the client and the APIs. Our proxy embeds hyperlinks into the APIs response to provide clients with possible transitions to further application states. It enables clients to interact with REST APIs through the server's responses. It does not require the client to know any prior information to use it except the URL to go to the next state. The client is able to follow URIs to make the transition without building the URIs by itself. We propose the REST API that enables a client to add, modify, and delete templates and links in the proxy. As an illustrative example, we consider our wood supply chain use case

discussed in Chapter 1.

This chapter is structured as follows. First, we discuss the detail of the proposed HATEOAS client that navigates through different hyperlinks. Then, we discuss our proxy as a resource solution that processes the responses and extends them by injecting hyperlinks. After that, we detail the proposed proxy template management solution that allows for updating the proxy with templates and links. Then, we detail the results and performance evaluation of the proposed solution. Finally, we summarize the results obtained in this chapter.

6.2 HATEOAS Client

The proposed HATEOAS client sends a request to the actual server endpoint and receives the response message. Then, it sends the response message to the proxy endpoint and receives the extended response with links. The client can use these links to make GET, PUT, and DELETE requests. Our proposed solution allows the client to keep original requests and response messages and prevents modification in the request message content, whereas the response message is simply extended with relevant HATEOAS links.

First, the HATEOAS client (such as the forest manager client) calls the `/chain` resource with method `'POST'` actual server endpoint to create the data and also provide the encryption type (e.g symmetric or asymmetric). The data contains the following parameters:

```
{
    "id": "RFID_number",
    "resource": "log",
    "woodtype": "maple",
    "datetime": "2022-05-20, T-17:20:33.45+01:00",
    "location":
    {
        "lat": "32,3351",
        "long": "-63,0561"
    }
}
```

In case of successful response (HTTP code 201), it receives the data id of the created resource. Then, client calls the proxy endpoint with the `'GET'` method, passing the previous response message (including the data id that is useful for the proxy to generate the links) as a parameter, and receives the extended version of the response, that includes HATEOAS links. The client can use these links to perform further actions such as data read, update, and delete the specific data.

After that, the forest manager client may use the GET link that calls `/chain/<id>` to get the data of that specific id. In the successful response, it receives the representation of the data. The proxy is called again to get an extended response with additional links. To update the data forest manager client uses the PUT link that calls the `/chain/<id>` resource and provides the data and encryption type to the

actual server to update their data. It also has an option to delete their data using the DELETE link, To do so, it calls `/chain/<id>` to delete their data for the specific id. All responses involve an additional call to the proxy and extended responses are provided with HATEOAS links.

6.3 Proxy as a Resource

In this section, we explain the proxy that guides the client regarding the possible transitions after processing that particular response. We design a proxy as a resource, a RESTful service, that can be utilized by any client and is configurable through its Web API. The idea to develop the concept of "proxy as a resource" comes from the fact that most RESTful APIs implement the REST architectural style but are missing the part related to HATEOAS. Implementing the HATEOAS part as a proxy allows to progressively build a knowledge base of templates that helps recognize specific (parts of) data schema contained in a given response message and generate adequate links to extend the message with. Therefore, when our proxy receives a response message forwarded from a client, it generates hyperlinks and inserts them into the response, and forwards that response back to the HATEOAS client. The client sends the actual response message (such as data id) to the proxy (`/chain/<id>` resource, `'POST'` method). To create the links, the proxy must know the content of the response. The proxy creates the following links such as GET, PUT and DELETE for the specific data id. In the links, the client also has an option to make a new POST request to add new data.

```
links = [
    {
        "href": "http://localhost:8888/chain/<id>",
        "type": "GET"
    },
    {
        "href": "http://localhost:8888/chain/<id>",
        "type": "PUT"
    },
    {
        "href": "http://localhost:8888/chain/<id>",
        "type": "DELETE"
    },
    {
        "href": "http://localhost:8888/chain",
        "type": "POST"
    }
]
```

In these links, the href represents the URL of the resource and the type is an HTTP verb that indicates what the client can do using the specific link. The proxy forwards an extended response message with links to the client.

We automate the management of proxy links and responses according to the data. To do so, we design link templates where the first part is the actual proxy endpoint that is known and the second part is data id that can be changed according to the response message. We provide the unique id of the data in the link template to point to the relevant data. Our proxy stores link templates and generate different types of links (such as GET, PUT, DELETE) for specific data id. We use headers that contain the content type (such as JSON data format) in HTTP requests and responses. This proxy has an advantage in terms of that we do not need to change actual APIs and HATEOAS client for navigational support.

6.4 Proxy Template Management

We propose a proxy that manages links. These links might need to be changed over time that is hard to add them manually in the code. It is also a time-consuming process. Therefore, we propose a REST APIs to manage these links and the client does not have to go into code to do it. The proposed REST APIs allows clients to add new templates, add new links, and update and delete existing templates and links at run time. The proposed APIs enables the client to update the proxy with templates and links.

Let us consider our running example, forest manager client cuts logs and stores data in the framework. In the response, the proposed proxy returns links with log id. The forest manager actor can use these links to perform GET, PUT, and DELETE on log data. Later, if the forest manager client wants to add a new template with only two links such as GET and PUT. Then, it will send a `POST` request to the `/templates` resource of proxy to create a new template. In the successful response (HTTP code 201), it returns the links including template id in the response. To update the template, it will make a `PUT` request to the `/templates/<id>` resource. The proposed API will update the template for specific template id on the proxy.

Later, if the forest manager client wants to read the all templates that they created or updated, then it will send a `GET` request to the `/templates` resource. In the response, he will receive the all existing templates. Accordingly, to read the specific template based on template id, an actor will call the `/templates/<id>` resource with method `GET`. A request to the `/templates/<id>` resource with method `DELETE` will delete the template for the specific template id.

Figure 6.1 shows the swagger user interface of the proposed APIs discussed above.

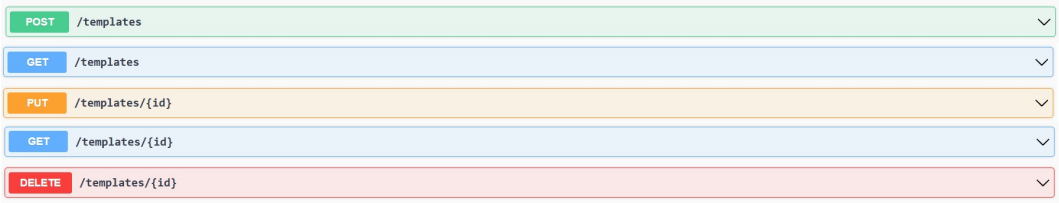

Figure 6.1: Swagger interface of the proposed proxy APIs.

6.5 Results and Evaluation

In the following, we discuss the results and performance evaluation of our proposed HATEOAS client with REST APIs solution. In Section 6.5.1, we describe the experimental setup and implementation details. The quantitative performance evaluation of our proposed solution is detailed in Section 6.5.2.

6.5.1 Experimental Setup and Implementation

We used Python 3 to implement the proposed solution. We used the JSON library[1] to receive and represent information in human-readable form. We used Flask[2], which is a Python-based Web framework that enabled us to manage HTTP requests and RESTful APIs.

We conducted our experiments on 64-bit Windows operating systems with 16 GB of RAM, x64 Core i7 processor system clock cycle at 1.80 GHz.

6.5.2 Evaluation

To highlight the advantages of our solution, we provide a comparative analysis of the non-HATEOAS client with our HATEOAS client solution.

The limitation of creating a non-HATEOAS client solution that would not offer links is that the client requires to know about the services in advance. It also requires having a knowledge which requests are permitted and how these requests are created. The second issue is to design a client that discovers all resources, the developer needs to do a lot of work to implement it [79]. Third, they need to build all URIs by themselves. These issues might increase the chance of errors during the development phase.

In contrast, the proposed HATEOAS client that uses hyperlinks from the proxy response only requires knowing the link type and href of the possible links. These link types are added to the hyperlinks. The proposed client requires to follow the URI to go to the next stage without building the URI by themselves. Our proposed solution enables HATEOAS for services and the client does not need to enable them itself.

We calculated the time cost for POST, GET, PUT, DELETE, and links. We summarized the detailed results statistics of our proposed solution in Table 6.1. Figure 6.2 depicts the overall average time needed to request REST commands and creates links for an increasing number of clients.

We tested our solution 100 times for each operation including requests and create links and then calculated the various statistics such as average time, Standard Deviation (SD), min, and max values in seconds. For the experiments, we observed the time that is required to answer REST requests and create links.

As we can see from the results, in the case of 1 client, the POST request gives an average time of 0,0423 seconds, and an overhead of 5% which is used to create the links. The GET request has an SD of 0,0343 seconds and the PUT provides an SD

[1] `https://docs.python.org/3/library/json.html`
[2] `https://flask.palletsprojects.com/en/2.0.x/`

Table 6.1: Timing statistics to answer REST requests and create links

No.clients	Requests	Avg Time	Overhead	St Deviation	Min	Max
1	POST	0,0423	5%	0,0301	0,0354	0,0675
	GET	0,0417	5%	0,0343	0,0371	0,0551
	PUT	0,0352	6%	0,0234	0,0314	0,0597
	DELETE	0,0254	8%	0,0201	0,0233	0,0474
	Links	0,0023	-	0,0011	0,0021	0,0046
40	POST	0,0532	4%	0,0504	0,0523	0,0703
	GET	0,0424	5%	0,0311	0,0347	0,0543
	PUT	0,0436	5%	0,0325	0,0352	0,0613
	DELETE	0,0341	6%	0,0302	0,0324	0,0452
	Links	0,0024	-	0,0021	0,0022	0,0047
80	POST	0,0573	4%	0,0432	0,0553	0,0795
	GET	0,0456	5%	0,0403	0,0431	0,0561
	PUT	0,0479	4%	0,0421	0,0443	0,0594
	DELETE	0,0335	6%	0,0313	0,0322	0,0431
	Links	0,0024	-	0,0021	0,0023	0,0046

of 0,0234 seconds. The GET request provides an overhead of 5% which is similar to the POST request. The DELETE request gives an average time of 0,0254 seconds and has a min value of 0,0233 seconds.

For the case of 40 clients, the POST request time gives an overhead of 4% which is slightly less than the case of 1 client. The GET request time has an average of 0,0424 seconds which is close to the average GET time of 1 client. The overhead of GET requests is 5% which is similar to the case of 1 and 80 clients. The PUT gives an SD of 0,0325 seconds and has a max value of 0,0613 seconds which is slightly higher than the case of 1 client. The DELETE request has a min value of 0,0324 seconds and creates links that give a min value of 0,0021 seconds.

Similarly, In the case of 80 clients, the POST overhead is 4% which is similar to the POST overhead of 40 clients. The POST request has an SD of 0,0432 seconds which is slightly less than the SD of 40 clients. The GET gives a max value of 0,0561 seconds which is close to the max value of GET for 1 client. The average request time of PUT does not show much difference for both cases 40 and 80 clients. The DELETE provides a max value of 0,0431 seconds which is less than the max value for clients 1 and 40. The create links average time is 0,0024 seconds which is similar to the create links average time of 40 clients.

The results demonstrate that our proposed solution gives a reasonable time overhead. The average time to answer REST requests and create links is not affected much by increasing the number of clients.

6.6 Chapter Summary

In this chapter, we design a solution that interacts with the REST APIs, processes the response message, and offers relevant links to the client, so the client would

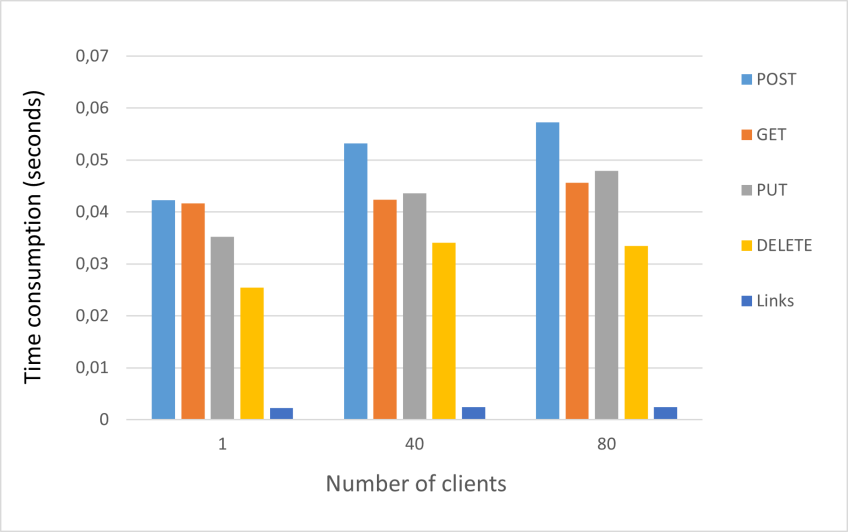

Figure 6.2: Overview of overall time needed to answer REST requests and create links on different numbers of clients

perform further operations on the data such as POST, GET, PUT, and DELETE. It allows the client to navigate through data using RESTful APIs. It is complex to do it manually because it is a lot of work on the implementation side. To do so, a developer needs to know responses from every server. It increases the chance of errors during the development phase. Our solution allows a client to add new templates and links and also to update and delete the existing templates using our designed APIs.

We also discussed the results and evaluation of our proposed solution. The performance of our solution is evaluated in terms of time cost. The proposed solution is flexible to deal with an increasing number of clients, as the results demonstrate an acceptable overhead, in average 5% of additional processing time is required to extend the HTTP responses with HATEOAS links.

Chapter 7

Conclusion and Future Work

In this chapter, we first summarize the main contributions of this dissertation, our Web framework, data storage and update solution, semantic access control and Web client. Then, we provide directions for future work, mainly related to keys management, HATEOAS, rule management and blockchain.

7.1 General Conclusion

In this Ph.D. thesis, we propose a decentralized solution for secure and privacy-aware data storage, multi-level ontology-driven access control, and support for data mutability without any third party. We detail, in each chapter of this dissertation, the following contributions:

- **A Decentralized Web Framework:** In chapter 3, we propose a decentralized Web framework that allows data owners to control and manage their data without any trusted third party. We design the RESTful APIs that demonstrate our solution applicability on the Web, with all the benefits that this architectural style brings. We provide the comparative analysis of our APIs with existing Hyperledger's APIs.

- **A Decentralized Mutable Data Storage:** In chapter 4, we propose a decentralized solution that allows authorized actors to write, read, delete, update their data and access the history of previous versions. Our solution stores metadata and DHT keys (hash pointer of the data) on the blockchain, while encrypted data is managed on the DHT, which enables the storage of large data contents. We propose a pointer system that allows data owners to access their update history. To ensure data security, we propose an encryption design that allows actors to choose between different types of encryption mechanisms to store and read data in a decentralized framework. We design a metadata structure that keeps an immutable record of data operations on the blockchain to ensure actors' trust. In addition, we propose a traceability algorithm that enables product traceability in a decentralized ledger. The design of the proposed solution ensures security properties such as confidentiality, integrity, availability,

and non-repudiation. It protects data against linking, eavesdropping, spoofing, and modification attacks. We evaluate the performance of our solution in terms of scalability with a proof-of-concept prototype, by implementing time measurements with different numbers of actors. The experimental results show that our proposed solution handle a large number of actors and achieved acceptable time overhead.

- **Semantic Role-based Access Control:** In chapter 5, we combine the Role-Based Access Control (RBAC) model with an OWL ontology to control access to the data through RESTful APIs. We define access control rules using HTTP verbs that allow authorized actors to perform POST, GET, PUT, and DELETE on data. The proposed solution manages complex permissions to access data and manage relationships between actors. We model OWL classes (such as roles, actors, and resources) and individuals according to our wood supply chain motivating scenario. We discuss implementation details and provide an average time comparison of our solution using different reasoners.

- **Web Client and HATEOAS proxy:** In chapter 6, we develop a proxy that enables HATEOAS navigation. The proposed proxy processes the API's response and offers relevant links to the client with the help of REST APIs. It manages the updates of links and responses according to the data. We design a HATEOAS client to navigate through data using REST APIs. The proposed HATEOAS client is generic and supports any Web-based REST APIs. We also propose a proxy template management solution that enables a client to update the proxy with templates and links using our designed REST APIs. We provide implementation details and evaluate its performance in terms of time overhead.

7.2 Future Work

In the following, we provide the directions for future research including key management, proposed solution applicability to large scale complex scenarios, ontology management REST API, semantic-based natural language rules, and semantic HATEOAS proxy.

- **Key Management:** Today, most security solutions, such as for example ring signature, depend on asymmetric cryptography. Therefore, public and private keys are required for each involved actor. Key management is an issue for large scale solutions where large number of keys are involved and need to be managed. It is also important to manage the private key that should not be disclosed. According to the literature, key management issue is not new, this problem is constant and an efficient solution is still required as the number of required keys grows over time and with the increasing number of applications requiring them.

- **Solution Applicability to Large Scale Complex Scenarios:** In this dissertation, we tested our solution according to wood supply chain, with a limited

number of actors and a rather low diversity of data. A direction for future research is to explore the applicability the proposed solution with different real life scenarios, increase the number of actors with diverse data, and improve its performance for large-scale networks.

- **Ontology Management REST API:** In this thesis, we used Protégé to model our ontology. It is difficult for non-experts to design an ontology using Protégé without having expertise in it. This opens the door for future work to design a REST API combined with user-friendly Web interfaces to manage ontologies at run time. The REST API should enable actors to add and update classes, instances, and object properties in the ontology without learning to use a complex tool such as Protégé. A generic user-friendly interface for ontology edition that connects to a RESTful API to enable domain-specific edits would require a configuration driven by domain knowledge. We believe that it would be an interesting contribution to facilitate the use of ontologies in different domains with such a tool.

- **Semantic-based Natural Language Rules:** Semantic Web Rule Language (SWRL) allows actors to perform actions on data according to constraints or conditions. It has more expressive power to define rules which means complex rules can be written in OWL. To write rules in SWRL, actors need to understand and follow the syntax properly. One future direction is to design a natural language rule interface that allows actors to write rules in human language. This interface should be integrated with OWL and natural language rules should be converted to SWRL at the back end.

- **Semantic HATEOAS Proxy:** The HATEOAS proxy developed in this thesis could be improved so it would be able to understand the context of the data it receives. To do so, semantic annotations to the data should be added, as well as contextual annotations, for the proxy to improve the selection of best applicable templates for each request. Another possibility is to give an overview of previous requests, so that the HATEOAS proxy can apply its algorithm over a view of a partial set of already executed calls, to facilitate the selection of the next actions to suggest to the client. This evolution would connect our work on this aspect to the research work related to fragments of business processes.

Bibliography

[1] Oluwashina Joseph Ajayi, Joseph Rafferty, Jose Santos, Matias Garcia-Constantino, and Zhan Cui. Beca: A blockchain-based edge computing architecture for internet of things systems. *IoT*, 2(4):610–632, 2021.

[2] Hamda Al Breiki, Lamees Al Qassem, Khaled Salah, Muhammad Habib Ur Rehman, and Davor Sevtinovic. Decentralized access control for iot data using blockchain and trusted oracles. In *2019 IEEE International Conference on Industrial Internet (ICII)*, pages 248–257. IEEE, 2019.

[3] Marco Alessi, Alessio Camillo, Enza Giangreco, Marco Matera, Stefano Pino, and Davide Storelli. Make users own their data: A decentralized personal data store prototype based on Ethereum and IPFS. In *3rd International Conference on Smart and Sustainable Technologies (SpliTech)*, pages 1–7. IEEE, 2018.

[4] Norta Alexander, Udokwu Chibuzor, and Leiding Benjamin. Mfssia: Multi-factor self-sovereign identity authentication. `https://ontochain.ngi.eu/content/mfssia`, 2017. Accessed: 2021-12-05.

[5] Abdulelah A Algosaibi, Saleh Albahli, Samer F Khasawneh, and Austin Melton. Web evolution-the shift from information publishing to reasoning. *International Journal of Artificial Intelligence and Applications (IJAIA)*, 8(6):11–28, 2017.

[6] Rawaa Ahmed Ali and S StatPearls Nagalli. Internet. *British Muslims in Number. Muslim Council of Britain*, 2015.

[7] Saqib Ali, Guojun Wang, Bebo White, and Roger Leslie Cottrell. A blockchain-based decentralized data storage and access framework for pinger. In *2018 17th IEEE International Conference on Trust, Security and Privacy in Computing and Communications/12th IEEE International Conference on Big Data Science and Engineering (TrustCom/BigDataSE)*, pages 1303–1308. IEEE, 2018.

[8] Jamila Alsayed Kassem, Sarwar Sayeed, Hector Marco-Gisbert, Zeeshan Pervez, and Keshav Dahal. Dns-idm: A blockchain identity management system to secure personal data sharing in a network. *Applied Sciences*, 9(15):2953, 2019.

[9] Grigoris Antoniou and Frank van Harmelen. Web ontology language: Owl. In *Handbook on ontologies*, pages 67–92. Springer, 2004.

[10] Sidra Aslam, Viktor Bukovszki, and Michael Mrissa. Decentralized data management privacy-aware framework for positive energy districts. *Energies*, 14(21):7018, 2021.

[11] Sidra Aslam, Viktor Bukovszki, and Michael Mrissa. Multi-level data access control in positive energy districts. In *Sustainability in Energy and Buildings*, pages 553–565. Springer, 2021.

[12] Sidra Aslam and Michael Mrissa. Privacy-aware distributed ledger for product traceability in supply chain environments. In *63rd SWST International Convention*, pages 3–10, 2020.

[13] Sidra Aslam and Michael Mrissa. Mutable and privacy-aware decentralized ledger for data management in wood supply chain environments. In *InnoRenew CoE International Conference 2021 (IRIC2021)*, 2021.

[14] Sidra Aslam and Michael Mrissa. A restful privacy-aware and mutable decentralized ledger. In *European Conference on Advances in Databases and Information Systems*, pages 193–204. Springer, 2021.

[15] Sidra Aslam and Michael Mrissa. A framework for privacy-aware and secure decentralized data storage. *Computer Science and Information Systems*, 2022.

[16] Sidra Aslam, Aleksandar Tošić, and Michael Mrissa. Secure and privacy-aware blockchain design: Requirements, challenges and solutions. *Journal of Cybersecurity and Privacy*, 1(1):164–194, 2021.

[17] Daniel Augot, Hervé Chabanne, Olivier Clémot, and William George. Transforming face-to-face identity proofing into anonymous digital identity using the bitcoin blockchain. In *2017 15th Annual Conference on Privacy, Security and Trust (PST)*, pages 25–2509. IEEE, 2017.

[18] Rakić B and Kotlar M. Dkg: Decentralised and scalable knowledge graph economy tools supporting the trusted, traceable and transparent ontological knowledge on blockchain. `https://ontochain.ngi.eu/content/dkg`, 2020.

[19] Sujata Banerjee, Sujoy Basu, Shishir Garg, Sukesh Garg, Sung-Ju Lee, Pramila Mullan, and Puneet Sharma. Scalable grid service discovery based on uddi. In *Proceedings of the 3rd international workshop on Middleware for grid computing*, pages 1–6, 2005.

[20] Dizza Beimel and Mor Peleg. Using OWL and SWRL to represent and reason with situation-based access control policies. *Data & Knowledge Engineering*, 70(6):596–615, 2011.

[21] Giampaolo Bella, Domenico Cantone, Cristiano Longo, Marianna Nicolosi Asmundo, and Daniele Francesco Santamaria. Semantic representation as a key enabler for blockchain-based commerce. In *International Conference on the Economics of Grids, Clouds, Systems, and Services*, pages 191–198. Springer, 2021.

[22] Nazanin Zahed Benisi, Mehdi Aminian, and Bahman Javadi. Blockchain-based decentralized storage networks: A survey. *Journal of Network and Computer Applications*, 162:102656, 2020.

[23] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic Web. *Scientific american*, 284(5):34–43, 2001.

[24] Elisa Bertino. RBAC models - concepts and trends. *Computers & Security*, 22(6):511–514, 2003.

[25] Jonathan Billington, Søren Christensen, Kees van Hee, Ekkart Kindler, Olaf Kummer, Laure Petrucci, Reinier Post, Christian Stehno, and Michael Weber. The petri net markup language: Concepts, technology, and tools. In *International Conference on Application and Theory of Petri Nets*, pages 483–505. Springer, 2003.

[26] Alex Biryukov, Dmitry Khovratovich, and Ivan Pustogarov. De anonymisation of clients in bitcoin p2p network. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 15–29, 2014.

[27] Jaqueline Blake and Wayne Pease. Development of an ontology to improve supply chain management (scm) in the australian timber industry. In *Semantic Web Technologies and E-Business: Toward the Integrated Virtual Organization and Business Process Automation*, pages 360–383. IGI Global, 2007.

[28] Marcelo Blois, Maurício Escobar, and Ricardo Choren. Using agents and ontologies for application development on the semantic Web. *Journal of the Brazilian Computer Society*, 13(2):35–44, 2007.

[29] Yogita Borse, Anushka Chawathe, Deepti Patole, and Purnima Ahirao. Anonymity: A secure identity management using smart contracts. In *Proceedings of International Conference on Sustainable Computing in Science, Technology and Management (SUSCOM), Amity University Rajasthan, Jaipur-India*, 2019.

[30] Lexi Brent, Anton Jurisevic, Michael Kong, Eric Liu, Francois Gauthier, Vincent Gramoli, Ralph Holz, and Bernhard Scholz. Vandal: A scalable security analysis framework for smart contracts. *arXiv preprint arXiv:1809.03981*, 2018.

[31] N Brindha, S Deepa, and S Balamurugan. Security protocol for multimedia streaming. *Design and Analysis of Security Protocol for Communication*, pages 155–170, 2020.

[32] Clemens Brunner, Ulrich Gallersdörfer, Fabian Knirsch, Dominik Engel, and Florian Matthes. Did and vc: Untangling decentralized identifiers and verifiable credentials for the Web of trust. In *2020 the 3rd International Conference on Blockchain Technology and Applications*, pages 61–66, 2020.

[33] Kyle Burgess and Joe Colangelo. The promise of bitcoin and the blockchain. *Consumers' Research*, 2015.

[34] Rodrigo N Calheiros, Rajiv Ranjan, César AF De Rose, and Rajkumar Buyya. Cloudsim: A novel framework for modeling and simulation of cloud computing infrastructures and services. *arXiv preprint arXiv:0903.2525*, 2009.

[35] Antonio Calleja López, Arnau Monterde Mateo, and Xabier E Barandiaran. Framework for democratic governance of distributed architectures: Decentralised citizens owned data ecosystem. Technical report, The DECODE consortium, 2017.

[36] Roy H Campbell and M Dennis Mickunas. Building a dynamic interoperable security architecture for active networks. Technical report, Illinois univ champaign grants and contracts administration, 2002.

[37] Roelofs Caspar and Tanner Jack. Gimly id: an ssi application suite. `https://ontochain.ngi.eu/content/gimly-id`, 2019. Accessed: 2021-12-15.

[38] Stefano Ceri, Georg Gottlob, Letizia Tanca, et al. What you always wanted to know about datalog(and never dared to ask). *IEEE transactions on knowledge and data engineering*, 1(1):146–166, 1989.

[39] Jung Hwa Chae and Nematollaah Shiri. Formalization of RBAC policy with object class hierarchy. In *International Conference on Information Security Practice and Experience*, pages 162–176. Springer, 2007.

[40] Antorweep Chakravorty and Chunming Rong. Ushare: user controlled social media based on blockchain. In *Proceedings of the 11th international conference on ubiquitous information management and communication*, pages 1–6, 2017.

[41] Mohammad MR Chowdhury, Josef Noll, and Juan Miguel Gomez. Enabling access control and privacy through ontology. In *2007 Innovations in Information Technologies (IIT)*, pages 168–172. IEEE, 2007.

[42] Lorenzo Cirio, Isabel F Cruz, and Roberto Tamassia. A role and attribute based access control system using semantic Web technologies. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 1256–1266. Springer, 2007.

[43] Kumar G Conti M, Brighente A and Saha R. Dart: A distributed-oracles framework for privacy-preserving data traceability. `https://ontochain.ngi.eu/content/dart`, 2020. Accessed: 2021-12-13.

[44] Michael Crosby, Pradan Pattanayak, Sanjeev Verma, Vignesh Kalyanaraman, et al. Blockchain technology: Beyond bitcoin. *Applied Innovation*, 2(6-10):71, 2016.

[45] Francisco Curbera, Matthew Duftler, Rania Khalaf, William Nagy, Nirmal Mukhi, and Sanjiva Weerawarana. Unraveling the Web services Web: an introduction to soap, wsdl, and uddi. *IEEE Internet computing*, 6(2):86–93, 2002.

[46] Sammy de Figueiredo, Akash Madhusudan, Vincent Reniers, Svetla Nikova, and Bart Preneel. Exploring the storj network: a security analysis. In *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, pages 257–264, 2021.

[47] Gero Decker, Alexander Lüders, Hagen Overdick, Kai Schlichting, and Mathias Weske. RESTful petri net execution. In *International Workshop on Web Services and Formal Methods*, pages 73–87. Springer, 2008.

[48] Kiran Devaram and Daniel Andresen. Soap optimization via parameterized client-side caching. In *Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2003)*, pages 785–790. Citeseer, 2003.

[49] Tien Tuan Anh Dinh, Ji Wang, Gang Chen, Rui Liu, Beng Chin Ooi, and Kian-Lee Tan. Blockbench: A framework for analyzing private blockchains. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1085–1100, 2017.

[50] Rotondi Domenico, Saltarella Marco, Azzara Pietro, and Riccobene Michelangelo. Seip: Service for encrypted information provider. `https://ontochain.ngi.eu/content/seip`, 2020. Accessed: 2021-12-08.

[51] John Domingue, Allan Third, and Manoharan Ramachandran. The fair trade framework for assessing decentralised data solutions. In *Companion Proceedings of The 2019 World Wide Web Conference*, pages 866–882, 2019.

[52] Jos Dumortier and Niels Vandezande. Critical observations on the proposed regulation for electronic identification and trust services for electronic transactions in the internal market. *ICRI research paper*, 9, 2012.

[53] Paul Dunphy and Fabien AP Petitcolas. A first look at identity management schemes on the blockchain. *IEEE security & privacy*, 16(4):20–29, 2018.

[54] Ashutosh Dhar Dwivedi, Gautam Srivastava, Shalini Dhar, and Rajani Singh. A decentralized privacy-preserving healthcare blockchain for iot. *Sensors*, 19(2):326, 2019.

[55] Jesse Eisses, Laurens Verspeek, Chris Dawe, and Sjoerd Dijkstra. Effect network: Decentralized network for artificial intelligence. `http://effect.ai/download/effectwhitepaper.pdf`, 2018. Accessed: 2020-02-01.

[56] Ariel Ekblaw, Asaph Azaria, John D Halamka, and Andrew Lippman. A case study for blockchain in healthcare:"medrec" prototype for electronic health records and medical research data. In *Proceedings of IEEE open & big data conference*, volume 13, page 13, 2016.

[57] Xing Fan, Baoning Niu, and Zhenliang Liu. Scalable blockchain storage systems: research progress and models. *Computing*, 104(6):1497–1524, 2022.

[58] Yanfang Fan, Zhen Han, Jiqiang Liu, and Yong Zhao. A mandatory access control model with enhanced flexibility. In *2009 international conference on multimedia information networking and security*, volume 1, pages 120–124. IEEE, 2009.

[59] Tiago M Fernández-Caramés and Paula Fraga-Lamas. A review on the use of blockchain for the internet of things. *Ieee Access*, 6:32979–33001, 2018.

[60] Rodolfo Ferrini and Elisa Bertino. Supporting RBAC with XACML+OWL. In *Proceedings of the 14th ACM symposium on Access control models and technologies*, pages 145–154, 2009.

[61] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. University of California, Irvine, 2000.

[62] Tim Finin, Anupam Joshi, Lalana Kagal, Jianwei Niu, Ravi Sandhu, William Winsborough, and Bhavani Thuraisingham. ROWLBAC: representing role based access control in owl. In *Proceedings of the 13th ACM symposium on Access control models and technologies*, pages 73–82, 2008.

[63] Roberto García and Rosa Gil. Social media copyright management using semantic Web and blockchain. In *Proceedings of the 21st International Conference on Information Integration and Web-based Applications & Services*, pages 339–343, 2019.

[64] Fausto Giunchiglia, Rui Zhang, and Bruno Crispo. Relbac: Relation based access control. In *2008 Fourth International Conference on Semantics, Knowledge and Grid*, pages 3–11. IEEE, 2008.

[65] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 89–98, 2006.

[66] G Guy Zyskind, O Oz Nathan, and AS Alex'Sandy'Pentland. Decentralizing privacy: using blockchain to protect personal data. In *Proceedings of the Security and Privacy Workshops (SPW). San Jose, USA: IEEE*, 2015.

[67] Muhammad Asif Habib, Nasir Mahmood, Muhammad Shahid, Muhammad Umar Aftab, Uzair Ahmad, and C Muhammad Nadeem Faisal. Permission based implementation of dynamic separation of duty (DSD) in role based access control (RBAC). In *2014 8th international conference on signal processing and communication systems (ICSPCS)*, pages 1–10. IEEE, 2014.

[68] Karim Hadjar. University ontology: A case study at ahlia university. In *Semantic Web*, pages 173–183. Springer, 2016.

[69] BQ Hai and Z Ying. Study on the access control model in information security. In *Quad-regional Radio Science and Wireless Technology Conference*, pages 830–834, 2011.

[70] Festim Halili and Erenis Ramadani. Web services: a comparison of soap and rest services. *Modern Applied Science*, 12(3):175, 2018.

[71] Timo Hanke, Mahnush Movahedi, and Dominic Williams. Dfinity technology overview series, consensus system. *arXiv preprint arXiv:1805.04548*, 2018.

[72] Yahya Hassanzadeh-Nazarabadi, Alptekin Küpçü, and Öznur Özkasap. Lightchain: A dht-based blockchain for resource constrained environments. *arXiv preprint arXiv:1904.00375*, 2019.

[73] Florian Haupt, Frank Leymann, and Karolina Vukojevic-Haupt. API governance support through the structural analysis of REST APIs. *Computer Science-Research and Development*, 33(3):291–303, 2018.

[74] Yiming Hei, Yizhong Liu, Dawei Li, Jianwei Liu, and Qianhong Wu. Themis: An accountable blockchain-based p2p cloud storage scheme. *Peer-to-Peer Networking and Applications*, 14(1):225–239, 2021.

[75] Willem-Jan van den Heuvel, Damian A. Tamburri, Damiano D'Amici, Fabiano Izzo, and Sandra Potten. Chainops for smart contract-based distributed applications. In *International Symposium on Business Modeling and Software Design*, pages 374–383. Springer, 2021.

[76] Haihui Huang, Xiuxiu Zhou, and Jun Liu. Food supply chain traceability scheme based on blockchain and epc technology. In *International Conference on Smart Blockchain*, pages 32–42. Springer, 2019.

[77] Marc Hüffmeyer, Florian Haupt, Frank Leymann, and Ulf Schreier. Authorization-aware hateoas. In *CLOSER*, pages 78–89, 2018.

[78] Janne Häkli, Kaarle Jaakkola, Pekka Pursula, Miika Huusko, and Kaj Nummila. Uhf rfid based tracking of logs in the forest industry. In *2010 IEEE International Conference on RFID (IEEE RFID 2010)*, pages 245–251, 2010.

[79] Daniel Jacobson, Greg Brail, and Dan Woods. *APIs: A strategy guide.* " O'Reilly Media, Inc.", 2012.

[80] Vila Jean and Broustail Alain. Nftwatch - an rdf based ontology. `https://ontochain.ngi.eu/content/nftwatch`, 2020. Accessed: 2021-12-05.

[81] Kurt Jensen, Lars Michael Kristensen, and Lisa Wells. Coloured petri nets and cpn tools for modelling and validation of concurrent systems. *International Journal on Software Tools for Technology Transfer*, 9(3):213–254, 2007.

[82] Qiu Jiong and Ma Chen-hua. Detecting and resolving constraint conflicts in role-based access control. In *2011 International Conference on Electrical and Control Engineering*, pages 5845–5848. IEEE, 2011.

[83] Arshad Junaid and Azad Ajmal. Reputable:a provenance-aware decentralized reputation system for blockchain-based ecosystems. `https://ontochain.ngi.eu/selected-projects`, 2020. Accessed: 2021-12-05.

[84] Alexandros Kalafatelis, Konstantinos Panagos, Anastasios E Giannopoulos, Sotirios T Spantideas, Nikolaos C Kapsalis, Marios Touloupou, Evgenia Kapassa, Leonidas Katelaris, Panayiotis Christodoulou, Klitos Christodoulou, et al. Island: An interlinked semantically-enriched blockchain data framework. In *International Conference on the Economics of Grids, Clouds, Systems, and Services*, pages 207–214. Springer, 2021.

[85] Ujval J Kapasi, Scott Rixner, William J Dally, Brucek Khailany, Jung Ho Ahn, Peter Mattson, and John D Owens. Programmable stream processors. *Computer*, 36(8):54–62, 2003.

[86] Avita Katal, Pranjal Gupta, Mohammad Wazid, RH Goudar, Abhishek Mittal, Sakshi Panwar, and Sanjay Joshi. Authentication and authorization: Domain specific role based access control using ontology. In *2013 7th International Conference on Intelligent Systems and Control (ISCO)*, pages 439–444. IEEE, 2013.

[87] Nattawat Khamphakdee, Nunnapus Benjamas, and Saiyan Saiyod. Performance evaluation of big data technology on designing big network traffic data analysis system. In *2016 Joint 8th International Conference on soft computing and Intelligent Systems (SCIS) and 17th International Symposium on Advanced Intelligent Systems (ISIS)*, pages 454–459. IEEE, 2016.

[88] Shinsaku Kiyomoto, Mohammad Shahriar Rahman, and Anirban Basu. On blockchain-based anonymized dataset distribution platform. In *2017 IEEE 15th International Conference on Software Engineering Research, Management and Applications (SERA)*, pages 85–92. IEEE, 2017.

[89] H Kohad, S Kumar, and A Ambhaikar. Scalability issues of blockchain technology. *Int. J. Eng. Adv. Technol. IJEAT*, 9(3), 2020.

[90] Sebastian Kotstein and Christian Decker. Navigational support for non HATEOAS-compliant Web-based APIs. In *Symposium and Summer School on Service-Oriented Computing*, pages 169–188. Springer, 2020.

[91] Kouji Kozaki, Eiichi Sunagawa, Yoshinobu Kitamura, and Riichiro Mizoguchi. Fundamental consideration of role concepts for ontology evaluation. In *EON@ WWW*, 2006.

[92] M Vinod Kumar and Sriman Narayana N Ch Iyengar. A framework for blockchain technology in rice supply chain management. *Advanced Science and Technology Letters*, 146:125–130, 2017.

[93] K Anitha Kumari, R Padmashani, R Varsha, and Vasu Upadhayay. Securing internet of medical things (iomt) using private blockchain network. In *Principles of Internet of Things (IoT) Ecosystem: Insight Paradigm*, pages 305–326. Springer, 2020.

[94] Roffia L, Gigli L, and Aguzzi C Marconi A. Desmo-ld: Decentralized smart oracles for trusted linked data. `https://ontochain.ngi.eu/content/desmo-ld`, 2020. Accessed: 2021-12-14.

[95] Joss Langford, A Poikola, Wil Janssen, Viivi Lähteenoja, and Marlies Rikken. Understanding mydata operators. `https://mydata.org/wp-content/uploads/sites/5/2020/04/Understanding-Mydata-Operators-pages.pdf`, 2020. Accessed: 2022-05-09.

[96] Henocque Laurent and Risterucci Gabriel. Uniprodapi: Universal proven data and process interchange. `https://ontochain.ngi.eu/content/uniprodapi`, 2020. Accessed: 2021-12-11.

[97] Tam Le and Matt W Mutka. Capchain: A privacy preserving access control framework based on blockchain for pervasive environments. In *2018 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 57–64. IEEE, 2018.

[98] Michel Legault. A practitioner's view on distributed storage systems: Overview, challenges and potential solutions. *Technology Innovation Management Review*, 11(6):32–41, 2021.

[99] Benjamin Leiding and Alex Norta. Mapping requirements specifications into a formalized blockchain-enabled authentication protocol for secured personal identity assurance. In *International Conference on Future Data and Security Engineering*, pages 181–196. Springer, 2017.

[100] Huiying Li, Xiang Zhang, Honghan Wu, and Yuzhong Qu. Design and application of rule based access control policies. In *Proc of the Semantic Web and Policy Workshop*, pages 34–41. Citeseer, 2005.

[101] Ruinian Li, Tianyi Song, Bo Mei, Hong Li, Xiuzhen Cheng, and Limin Sun. Blockchain for large-scale internet of things data storage and protection. *IEEE Transactions on Services Computing*, 12(5):762–771, 2018.

[102] Olga Liskin, Leif Singer, and Kurt Schneider. Teaching old services new tricks: adding hateoas support as an afterthought. In *Proceedings of the Second International Workshop on RESTful Design*, pages 3–10, 2011.

[103] Francesco Longo, Letizia Nicoletti, Antonio Padovano, Gianfranco d'Atri, and Marco Forte. Blockchain-enabled supply chain: An experimental study. *Computers & Industrial Engineering*, 136:57–69, 2019.

[104] Lundin Lotta, Chandran Lal, and Padayatti George. Nftwatch - an rdf based ontology. `https://ontochain.ngi.eu/content/nftwatch`, 2020. Accessed: 2021-12-07.

[105] Damian A Lucio, Fabiano D'Amici, Damiano, and Sandra Potten. Bowler: Blockchain oriented warehouse and low code engine and reasoner. `https://ontochain.ngi.eu/content/bowler`, 2020. Accessed: 2021-12-06.

[106] Ketil Lund, Anders Eggen, Dinko Hadzic, Trude Hafsoe, and Frank T Johnsen. Using web services to realize service oriented architecture in military communication networks. *IEEE communications magazine*, 45(10):47–53, 2007.

[107] Christian Lundkvist, Rouven Heck, Joel Torstensson, Zac Mitton, and Michael Sena. Uport: A platform for self-sovereign identity. `https://whitepaper.uport.me/uPort_whitepaper_DRAFT20170221.pdf`, 2017. Accessed: 2022-05-09.

[108] Emil C Lupu, Damian A Marriott, Morris S Sloman, and Nicholas Yialelis. A policy based role framework for access control. In *Proceedings of the first ACM Workshop on Role-based access control*, pages 11–es, 1996.

[109] Nikolaos Lykousas, Vasilios Koutsokostas, Fran Casino, and Constantinos Patsakis. The cynicism of modern cybercrime: Automating the analysis of surface Web marketplaces. *arXiv preprint arXiv:2105.11805*, 2021.

[110] Mohammad Madine, Khaled Salah, Raja Jayaraman, Yousof Al-Hammadi, Junaid Arshad, and Ibrar Yaqoob. Application-level interoperability for blockchain networks. *TechRxiv, IEEE*, 2021.

[111] M Mercedes Martínez-González, María Luisa Alvite-Díez, Pompeu Casanovas, Núria Casellas, David Sanz, Amador Aparicio, Inma Gutiérrez, et al. Ontoropa deliverable 2. proposed design specification and approach. Technical report, ONTOCHAIN, 2021.

[112] Amirreza Masoumzadeh and James Joshi. Ontology-based access control for social network systems. *International Journal of Information Privacy, Security and Integrity*, 1(1):59–78, 2011.

[113] Petar Maymounkov and David Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *International Workshop on Peer-to-Peer Systems*, pages 53–65. Springer, 2002.

[114] Sonia Mehla and Sarika Jain. Rule languages for the semantic Web. In *Emerging Technologies in Data Mining and Information Security*, pages 825–834. Springer, 2019.

[115] Ralph C Merkle. Protocols for public key cryptosystems. In *1980 IEEE Symposium on Security and Privacy*, pages 122–122. IEEE, 1980.

[116] Alexander Mikroyannidis, Allan Third, and John Domingue. A case study on the decentralisation of lifelong learning using blockchain technology. *Journal of Interactive Media in Education*, 2020(1):1–10, 2020.

[117] Malte Moser. Anonymity of bitcoin transactions. In *Münster Bitcoin Conference (MBC), Münster, Germany*, July 2013.

[118] Michael Mrissa, Aleksandar Tošić, Niki Hrovatin, Sidra Aslam, Balázs Dávid, László Hajdu, Miklós Krész, Andrej Brodnik, and Branko Kavšek. Privacy-aware and secure decentralized air quality monitoring. *Applied Sciences*, 12(4):2147, 2022.

[119] Snehal Mumbaikar, Puja Padiya, et al. Web services based on soap and rest principles. *International Journal of Scientific and Research Publications*, 3(5):1–4, 2013.

[120] San Murugesan. Understanding web 2.0. *IT professional*, 9(4):34–41, 2007.

[121] Satoshi Nakamoto and A Bitcoin. A peer-to-peer electronic cash system. `https://bitcoin.org/bitcoin.pdf`. Accessed: 2020-01-07.

[122] Arvind Narayanan, Joseph Bonneau, Edward Felten, Andrew Miller, and Steven Goldfeder. *Bitcoin and cryptocurrency technologies: a comprehensive introduction*. Princeton University Press, 2016.

[123] Denis Nasonov, Alexander A Visheratin, and Alexander Boukhanovsky. Blockchain-based transaction integrity in distributed big data marketplace. In *International Conference on Computational Science*, pages 569–577. Springer, 2018.

[124] Andy Neumann, Nuno Laranjeiro, and Jorge Bernardino. An analysis of public REST Web service APIs. *IEEE Transactions on Services Computing*, 2018.

[125] Isler Nikolaos, Devris and Xenakis Christos. Dw-marking - data watermarking: The missing link to on or off-chain implementation of distributed data marketplaces. `https://ontochain.ngi.eu/content/dw-marking`, 2020. Accessed: 2021-12-13.

[126] Michael Nofer, Peter Gomber, Oliver Hinz, and Dirk Schiereck. Blockchain. *Business & Information Systems Engineering*, 59(3):183–187, 2017.

[127] Roy Oberhauser. A hypermedia middleware for process enactment and adaptation. *International Journal of Software Engineering and Its Applications (IJSEIA)*, 10(6):53–68, 2016.

[128] Robert Olsson. Applying REST principles on local client-side APIs, 2014.

[129] Mourad Ouzzani and Athman Bouguettaya. Efficient access to web services. *IEEE Internet Computing*, 8(2):34–44, 2004.

[130] Cesare Pautasso. Bpmn for REST. In *International Workshop on Business Process Modeling Notation*, pages 74–87. Springer, 2011.

[131] Alex Pazaitis, Primavera De Filippi, and Vasilis Kostakis. Blockchain and value systems in the sharing economy: The illustrative case of backfeed. *Technological Forecasting and Social Change*, 125:105–115, 2017.

[132] Salvador Pérez, José L Hernández-Ramos, Diego Pedone, Domenico Rotondi, Leonardo Straniero, and Antonio F Skarmeta. A digital envelope approach using attribute-based encryption for secure data exchange in iot scenarios. In *2017 Global Internet of Things Summit (GIoTS)*, pages 1–6. IEEE, 2017.

[133] Julien Polge, Jérémy Robert, and Yves Le Traon. Permissioned blockchain frameworks in the industry: A comparison. *ICT Express*, 2020.

[134] Torsten Priebe, Wolfgang Dobmeier, and Nora Kamprath. Supporting attribute-based access control with ontologies. In *First International conference on availability, reliability and security (ARES'06)*, pages 8–pp. IEEE, 2006.

[135] Torsten Priebe, Wolfgang Dobmeier, Björn Muschall, and Günther Pernul. Abac–ein referenzmodell für attributbasierte zugriffskontrolle. *Sicherheit 2005, Sicherheit–Schutz und Zuverlässigkeit*, 2005.

[136] Torsten Priebe, Björn Muschall, Wolfgang Dobmeier, and Günther Pernul. A flexible security system for enterprise and e-government portals. In *International Conference on Database and Expert Systems Applications*, pages 884–893. Springer, 2004.

[137] Lili Qiu, Yin Zhang, Feng Wang, Mi Kyung, and Han Ratul Mahajan. *Trusted Computer System Evaluation Criteria*. Dod Computer Security Center, December 1985.

[138] Satyajeet Raje, Chowdary Davuluri, Michael Freitas, Rajiv Ramnath, and Jay Ramanathan. Using semantic Web technologies for RBAC in project-oriented environments. In *2012 IEEE 36th Annual Computer Software and Applications Conference*, pages 521–530. IEEE, 2012.

[139] Manoharan Ramachandran, Niaz Chowdhury, Allan Third, John Domingue, Kevin Quick, and Michelle Bachler. Towards complete decentralised verification of data with confidentiality: different ways to connect solid pods and blockchain. In *Companion Proceedings of the Web Conference 2020*, pages 645–649, 2020.

[140] Johnson Rebecca, Schaeffner Martin, and Reuter Michael. Hibi: Human identity blockchain initiative. `https://ontochain.ngi.eu/content/hibi`, 2021. Accessed: 2021-12-16.

[141] Dominik Renzel, Patrick Schlebusch, and Ralf Klamma. Today's top "RESTful" services and why they are not RESTful. In *International Conference on Web Information Systems Engineering*, pages 354–367. Springer, 2012.

[142] Jaideep Roy and Anupama Ramanujan. Understanding web services. *IT professional*, 3(6):69–73, 2001.

[143] Paul RYANab, Harshvardhan Pandit, and Rob Brennan. Building a data processing activities catalog: Representing heterogeneous compliance-related information for gdpr using dcat-ap and dpv. In *Further with Knowledge Graphs*, pages 169–182. IOS Press, 2021.

[144] Motaz K Saad, Ramzi Abed, and Hatem M Hamad. Performance evaluation of restful web services for mobile devices. *International Arab Journal of e-Technology*, 2010.

[145] Sara Saberi, Mahtab Kouhizadeh, Joseph Sarkis, and Lejia Shen. Blockchain technology and its relationships to sustainable supply chain management. *International Journal of Production Research*, 57(7):2117–2135, 2019.

[146] Pierangela Samarati and Sushil Jajodia. Data security. *Wiley Encyclopedia of Electrical and Electronics Engineering. John Wiley & Sons*, 1999.

[147] Ravi Sandhu, David Ferraiolo, Richard Kuhn, et al. The nist model for role-based access control: towards a unified standard. In *ACM workshop on Role-based access control*, volume 10, 2000.

[148] Ravi Sandhu and Pierangela Samarati. Authentication, access control, and audit. *ACM Computing Surveys (CSUR)*, 28(1):241–243, 1996.

[149] Ravi S Sandhu. Role-based access control. In *Advances in computers*, volume 46, pages 237–286. Elsevier, 1998.

[150] Andreas Schaad, Pascal Spadone, and Helmut Weichsel. A case study of separation of duty properties in the context of the austrian" elaw" process. In *Proceedings of the 2005 ACM symposium on Applied computing*, pages 1328–1332, 2005.

[151] Pascal Schiessle. Datalog-an overview and outlook on a decade-old technology. In *Proceedings of the 2020 OMI Seminars (PROMIS 2020)*, volume 1, pages 14–1. Universität Ulm, 2021.

[152] Hossein Shafagh, Lukas Burkhalter, Anwar Hithnawi, and Simon Duquennoy. Towards blockchain-based auditable storage and sharing of iot data. In *Proceedings of the 2017 on Cloud Computing Security Workshop*, pages 45–50, 2017.

[153] Ajay Kumar Shrestha, Julita Vassileva, and Ralph Deters. A blockchain platform for user data sharing ensuring user control and incentives. *Frontiers in Blockchain*, 3:48, 2020.

[154] Richard T Simon and Mary Ellen Zurko. Separation of duty in role-based environments. In *Proceedings 10th Computer Security Foundations Workshop*, pages 183–194. IEEE, 1997.

[155] Rajeev Sobti and Ganesan Geetha. Cryptographic hash functions: a review. *International Journal of Computer Science Issues (IJCSI)*, 9(2):461, 2012.

[156] Jihong Song and Shaopeng Wang. The pastry algorithm based on dht. *Comput. Inf. Sci.*, 2(4):153–157, 2009.

[157] Mirek Sopek, Dominik Tomaszuk, Szymon Głąb, Filip Turoboś, Ivo Zieliński, Dominik Kuziński, Ryszard Olejnik, Piotr Luniewski, and Przemysław Grądzki. Technological foundations of ontological ecosystems on the 3rd generation blockchains. *IEEE Access*, 2022.

[158] Ion Stoica, Robert Morris, David Liben-Nowell, David R Karger, M Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on networking*, 11(1):17–32, 2003.

[159] Eiichi Sunagawa, Kouji Kozaki, Yoshinobu Kitamura, and Riichiro Mizoguchi. A framework for organizing role concepts in ontology development tool: Hozo. *Boella et al.[8]*, pages 136–143, 2005.

[160] Nick Szabo. The idea of smart contracts. *Nick Szabo's papers and concise tutorials*, 6(1), 1997.

[161] Mohammad Mustafa Taye. Understanding semantic Web and ontologies: Theory and applications. *arXiv preprint arXiv:1006.4567*, 2010.

[162] Javier Prieto Tejedor, Sara Rodríguez González, María Aránzazu Moyeno Barredo, and Adrián Heredia Jimeno. Carechain: Supporting care through microinsurances using blockchain. `https://ontochain.ngi.eu/content/carechain`, 2020. Accessed: 2021-12-10.

[163] Feng Tian. An agri-food supply chain traceability system for china based on rfid & blockchain technology. In *2016 13th international conference on service systems and service management (ICSSSM)*, pages 1–6. IEEE, 2016.

[164] Juris Tihomirovs and Jānis Grabis. Comparison of soap and rest based web services using software evaluation metrics. *Information technology and management science*, 19(1):92–97, 2016.

[165] Dominik Tomaszuk, Dominik Kuziński, Mirek Sopek, and Bogusław Swiecicki. A distributed graph data storage in ethereum ecosystem. In *International Conference on the Economics of Grids, Clouds, Systems, and Services*, pages 223–231. Springer, 2021.

[166] Kentaroh Toyoda, P Takis Mathiopoulos, Iwao Sasase, and Tomoaki Ohtsuki. A novel blockchain-based product ownership management system (poms) for anti-counterfeits in the post supply chain. *IEEE access*, 5:17465–17477, 2017.

[167] Wei-Tek Tsai and Qihong Shao. Role-based access-control using reference ontology in clouds. In *2011 Tenth International Symposium on Autonomous Decentralized Systems*, pages 121–128. IEEE, 2011.

[168] Ioakeim Tzoulis and Zaharoula Andreopoulou. Emerging traceability technologies as a tool for quality wood trade. *Procedia Technology*, 8:606–611, 2013.

[169] Sarah Underwood. Blockchain beyond bitcoin. *Communications of the ACM*, 59(11):15–17, 2016.

[170] Steve Vinoski. Putting the" web" into web services. web services interaction models. 2. *IEEE Internet Computing*, 6(4):90–92, 2002.

[171] William Voorsluys, James Broberg, Rajkumar Buyya, et al. Introduction to cloud computing. *Cloud computing: Principles and paradigms*, pages 1–44, 2011.

[172] Shangping Wang, Yinglong Zhang, and Yaling Zhang. A blockchain-based framework for data sharing with fine-grained access control in decentralized storage systems. *Ieee Access*, 6:38437–38450, 2018.

[173] Sebastian Weber and Jörg Rech. An overview and differentiation of the evolutionary steps of the Web xy movement: the Web before and beyond 2.0. *Handbook of Research on Web 2.0, 3.0, and X. 0: Technologies, Business, and Social Applications*, pages 12–39, 2010.

[174] Kevin Werbach. *The blockchain and the new architecture of trust*. Mit Press, 2018.

[175] Martin Westerkamp, Friedhelm Victor, and Axel Küpper. Blockchain-based supply chain traceability: Token recipes model manufacturing processes. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 1595–1602. IEEE, 2018.

[176] Shawn Wilkinson, Tome Boshevski, Josh Brandoff, and Vitalik Buterin. Storj a peer-to-peer cloud storage network. Technical report, storj.io, 2014.

[177] Di Wu, Xiyuan Chen, Jian Lin, and Miaoliang Zhu. Ontology-based RBAC specification for interoperation in distributed environment. In *Asian Semantic Web Conference*, pages 179–190. Springer, 2006.

[178] Lei Xu, Nolan Shah, Lin Chen, Nour Diallo, Zhimin Gao, Yang Lu, and Weidong Shi. Enabling the sharing economy: Privacy respecting contract based on public blockchain. In *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts*, pages 15–21, 2017.

[179] Wenli Yang, Saurabh Garg, Ali Raza, David Herbert, and Byeong Kang. Blockchain: trends and future. In *Pacific Rim Knowledge Acquisition Workshop*, pages 201–210. Springer, 2018.

[180] Rui Zhang, Alessandro Artale, Fausto Giunchiglia, and Bruno Crispo. Using description logics in relation based access control. In *International Workshop on Description Logics, Oxford, UK*, July 2009.

[181] Ben Y Zhao, Ling Huang, Jeremy Stribling, Sean C Rhea, Anthony D Joseph, and John D Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on selected areas in communications*, 22(1):41–53, 2004.

[182] Xiaochen Zheng, Jinzhi Lu, Shengjing Sun, and Dimitris Kiritsis. Decentralized industrial iot data management based on blockchain and ipfs. In *IFIP International Conference on Advances in Production Management Systems*, pages 222–229. Springer, 2020.

[183] Zibin Zheng, Shaoan Xie, Hong-Ning Dai, Xiangping Chen, and Huaimin Wang. Blockchain challenges and opportunities: A survey. *International Journal of Web and Grid Services*, 14(4):352–375, 2018.

[184] Michael Zur Muehlen, Jeffrey V Nickerson, and Keith D Swenson. Developing web services choreography standards—the case of rest vs. soap. *Decision support systems*, 40(1):9–29, 2005.

[185] Guy Zyskind, Oz Nathan, et al. Decentralizing privacy: Using blockchain to protect personal data. In *2015 IEEE Security and Privacy Workshops*, pages 180–184. IEEE, 2015.

[186] Guy Zyskind, Oz Nathan, and Alex Pentland. Enigma: Decentralized computation platform with guaranteed privacy. *arXiv preprint arXiv:1506.03471*, 2015.

Index

Povzetek v slovenskem jeziku

Zasebnost in Varnost Podatkov za Decentralizirani Splet Stvari

7.3 Uvod

V zadnjih desetletjih sta internet in nato še splet zagotavljala zanesljiv sklad protokolov za podporo različnim aplikacijam, od shranjevanja podatkov do pretakanja in predvajanja v živo. Internet deluje kot globalno omrežje, ki omogoča, da naprave komunicirajo med seboj po vsem svetu, na podlagi internetnega protokola (IP[1]), ki zagotavlja identifikatorje za naprave, protokol za nadzor prenosa (TCP[2]) omogoči zanesljivo komunikacijo. Splet je zbirka spletnih strani, ki so med seboj povezane s hiperbesedilnimi povezavami in so dostopne prek interneta. Vsaka stran v spletu ima edinstven naslov, imenovan Enotni Identifikator Vira (URI[3]). Protokol za prenos hiperbesedila (HTTP[4]) je odgovoren za komunikacijo med spletnim odjemalcem (npr. brskalnikom) in spletnim strežnikom s pomočjo HTTP indeksHTTP sporočil (zahteva in odgovor) [5]. Prva implementacija spleta je temeljila na statičnih straneh [5] in omogočila le branje.

Nato je bil uveden dinamični splet, ki omogoča odjemalcu interakcijo s spletnimi stranmi. Hiter razvoj spleta je podprla tudi industrija zaradi potrebe po medpodjetni interakciji preko spletnih vmesnikov. Po tem trendu je razvoj sklada protokolov spletnih storitev omogočil programskim odjemalcem klic oddaljenih aplikacij s pomočjo spleta, s popolno abstrakcijo osnovnih operacijskih sistemov, protokolov in programskih jezikov. Sklad protokolov spletnih storitev je sestavljen iz protokola SOAP (ang. Simple Object Access Protocol)[5], WSDL (ang. Web Service Description Language)[6] in UDDI (ang. Universal Description, Discovery, and Integration)[7].

Vendar SOAP ni bil široko sprejet zaradi težav z zasnovo. Zaradi tega se je razvoj spletnih storitev razvil na osnovi arhitekturnega principa REST, ki pravilno izkorišča splet in HTTP protokol. Medtem ko so se je splet začel široko uporabljati, se je pojavila potreba po obsežni infrastrukturi, za ponujanje in nemoten dostop do spletnih

[1] https://datatracker.ietf.org/doc/html/rfc791
[2] https://www.ietf.org/rfc/rfc793.txt/
[3] https://datatracker.ietf.org/doc/html/rfc3986
[4] https://datatracker.ietf.org/doc/html/rfc2616
[5] https://www.w3.org/TR/soap/
[6] https://www.w3.org/TR/wsdl/
[7] http://www.uddi.org/pubs/uddi_v3.htm

storitev. To je spodbudilo razvoj računalništva v oblaku [34, 171]. Vendar je centralizirana zasnova računalništva v oblaku podvržena varnostnim pomanjkljivostim, kot je problem kritične točke odpovedi [172]. Zato se je pojavila potreba po decentraliziranem pristopu k shranjevanju in upravljanju podatkov, da bi uporabnikom ponudili enake storitve, kot jih ponuja oblak, brez varnostnih težav. V tem kontekstu se decentralizirani pristopi približajo shranjevanju in upravljanju podatkov ter uporabnikom ponudijo enake storitve, kot jih ponuja oblak, brez varnostnih težav. Zlasti v zadnjih nekaj letih je blockchain kot tehnologija razpršene evidence pridobila veliko pozornosti zaradi svoje decentralizirane in pregledne narave. Vendar kljub prednostim, ima tehnologija blockchain tudi številne omejitve. V tem diplomskem delu raziskujemo omejitve povezane z nespremenljivostjo podatkov, zasebnost in varnost.

7.4 Raziskovalni Prispevki in Zaključki

V nadaljevanju povzemamo glavne prispevke in rezultate te disertacije, spletno ogrodje, rešitev za shranjevanje podatkov, semantični nadzor dostopa in spletni odjemalec.

7.4.1 Decentralizirano Spletno Ogrodje:

V poglavju 3 smo predlagali decentralizirano spletno ogrodje, ki lastnikom podatkov omogoča nadzor in upravljanje podatkov brez kakršne koli vmesne osebe. Da bi to naredili, smo združili tehnologijo veriženja blokov z porazdeljeno zgoščevalno tabelo, več vrstami mehanizmov šifriranja, nadzorom dostopa do podatkov in podpisom v enem samem ogrodju, da bi izboljšali zasebnost posameznika, izboljšali varnost in zagotovili spremenljivost. Zasnovali smo API-je RESTful, ki prikazujejo uporabnost naše rešitve v spletu z vsemi prednostmi, ki jih prinaša ta arhitekturni slog. Zagotovili smo primerjalno analizo naših API-jev z obstoječimi API-ji tehnologije Hyperledger.

7.4.2 Decentralizirano Spremenljivo Shranjevanje Podatkov:

V poglavju 4 smo predlagali decentralizirano rešitev, ki pooblaščenim akterjem omogoča pisanje, posodabljanje in dostop do svoje zgodovine transakcij. Predlagana rešitev je shranila metapodatke in ključe v blockchain, medtem ko se šifrirani podatki upravljajo na porazdeljena zgoščeni tabeli, kar lastnikom podatkov omogoča posodobitev svojih podatkov. Da bi zagotovili varnost podatkov, smo predlagali zasnovo šifriranja, ki akterjem omogoča izbiro med različnimi vrstami mehanizmov šifriranja za shranjevanje podatkov v decentraliziranem okviru. Zasnovali smo strukturo metapodatkov, ki ohranjajo nespremenljive podatkovne operacije v verigi blokov, kar zagotavlja zaupanje in sledljivost akterja. Razvili smo rešitev, ki lastnikom podatkov omogoča spreminjanje svojih podatkov in dostop do njihove zgodovine posodobitev. Predlagana zasnova rešitve zagotavlja varnostne lastnosti, kot so zaupnost, celovitost, razpoložljivost in nezavrnitev. Podatke ščiti pred napadi povezovanja, prisluškovanja, ponarejanja in spreminjanja. Učinkovitost naše rešitve

v smislu razširljivosti smo ocenili s prototipom, in izvedli več meritev z različnim številom akterjev. Eksperimentalni rezultati so pokazali, da naša predlagana rešitev obvlada veliko število akterjev.

7.4.3 Semantični Nadozor Dostopa na podlagi Vlog:

V poglavju 5 smo združili model nadzora dostopa na podlagi vlog (RBAC) z ontologijo OWL za nadzor nepooblaščenega dostopa do podatkov prek API-jev RESTful. Pravila za nadzor dostopa smo definirali z uporabo metod HTTP, ki pooblaščenim akterjem omogočajo izvajanje POST, GET, PUT in DELETE nad podatki. Predlagana rešitev upravlja kompleksna dovoljenja za dostop do podatkov in upravljanje odnosov med akterji. Modelirali smo razrede OWL (kot so vloge, igralci, viri) in posameznike v skladu z našim motivacijskim scenarijem verige oskrbe z lesom. Razpravljali smo o podrobnostih izvedbe in podali primerjavo povprečnega časa, medtem ko smo uporabljali različne argumente.

7.4.4 HATEOAS Odjemalec z REST APIs:

V poglavju 6 smo razvili proxy, ki omogoča navigacijo HATEOAS. Predlagani proxy obdeluje odziv API-ja in ponuja ustrezne povezave do odjemalca s pomočjo API-jev REST. Oblikovali smo odjemalski HATEOAS za navigacijo po podatkih z uporabo API-jev REST. Predlagani odjemalec HATEOAS je splošen in podpira vse spletne API-je REST. Podali smo podrobnosti o implementaciji in ocenili njen odzivni čas.

Kazalo vsebine

Stvarno Kazalo

Declaration

I declare that this PhD Dissertation does not contain any materials previously published or written by another person except where due reference is made in the text.

Sidra Aslam