FAKULTÄT FÜR
INFORMATIK

Otto-von-Guericke-University Magdeburg

Faculty of Computer Science

Department for Technical & Business Information Systems (ITI)

Trusted Provenance with Blockchain

A Blockchain-based Provenance Tracking System for Virtual Aircraft Component Manufacturing

Master's Thesis

In cooperation with the German Aerospace Center

Author:

Steven Kocadag

Supervisors:

Prof. Dr. Klaus Turowski

Dipl.-Math. Matthias Pohl

External Advisor:

Andreas Schreiber

Magdeburg, 14.04.2023

Abstract

The importance of provenance in the digital age has led to significant interest in utilizing blockchain technology for tamper-proof storage of provenance data. This thesis proposes a blockchain-based provenance tracking system for the certification of aircraft components. The aim is to design and implement a system that can ensure the trustworthy, tamper-resistant storage of provenance documents originating from an aircraft manufacturing process. To achieve this, the thesis presents a systematic literature review, which provides a comprehensive overview of existing works in the field of provenance and blockchain technology. After obtaining strategies to utilize blockchain for the storage of provenance data on the blockchain, a system was designed to meet the requirements of stakeholders in the aviation industry. The thesis utilized a systematic approach to gather requirements by conducting interviews with stakeholders. The system was implemented using a combination of smart contracts and a graphical user interface to provide tamper-resistant, traceable storage of relevant data on a transparent blockchain. An evaluation based on the requirements identified during the requirement engineering process found that the proposed system meets all identified requirements. Overall, this thesis offers insight into a potential application of blockchain technology in the aviation industry and provides a valuable resource for researchers and industry professionals seeking to leverage blockchain technology for provenance tracking and certification purposes.

Contents

1 Introduction **1**
- 1.1 Motivation & Problem Statement . . . 1
- 1.2 Goals . . . 1
- 1.3 Structure . . . 2

2 Provenance **3**
- 2.1 Definition . . . 3
- 2.2 PROV Standard & Data Model . . . 3

3 Blockchain **6**
- 3.1 Development History . . . 6
- 3.2 Features of Blockchain . . . 8
- 3.3 Architecture . . . 9
- 3.4 Types of Blockchain Networks . . . 10
- 3.5 Consensus Mechanisms . . . 13
- 3.6 Transactions . . . 15

4 Systematic Literature Review **18**
- 4.1 Search Strategy . . . 18
- 4.2 Study Selection Criteria . . . 20
- 4.3 Data Extraction . . . 20
- 4.4 Synthesis . . . 21
- 4.5 Comparison of Storage Strategies . . . 23

5 Requirement Engineering **26**
- 5.1 Elicitation of Requirements . . . 26
- 5.2 Documentation of Requirements . . . 31

6 Concept **35**
- 6.1 Architecture . . . 35
- 6.2 Storage of PROV Documents . . . 36
- 6.3 Blockchain Technology . . . 37
- 6.4 Smart Contracts . . . 39
- 6.5 Graphical User Interface . . . 40
- 6.6 Permission Management . . . 41

7 Software Prototype **43**
- 7.1 Smart Contract . . . 43
- 7.2 Graphical User Interface . . . 48

8 Tests and Analysis **51**
- 8.1 Environment . . . 51
- 8.2 Functional Requirements . . . 54
- 8.3 Quality Requirements . . . 57

9 Conclusion **60**

Statement of Authorship **62**

Appendices **63**
A. PROV-DM Types and Relations 63
B. Collection of Studies 63
C. PROV Document of a Workflow 65
D. Implementation 68
E. Evaluation 76
F. Data Extraction Table 77

References **83**

List of Figures **92**

List of Tables **93**

List of Listings **94**

1 Introduction

1.1 Motivation & Problem Statement

Provenance refers to the origin or history of an artifact, object or data. For centuries, provenance information has played an important role in estimating the value of art, books, and wines. With the advent of the digital age, provenance became increasingly important for digital content. Because of the widespread adoption of digital technology and the growing prevalence of online communication and collaboration, digital artifacts arise at a higher rate than ever before. Unlike physical objects, digital artifacts can be easily copied, altered, and distributed without leaving a trace. Provenance provides a way to trace the origin and history of a digital artifact, helping to establish its authenticity and ownership. This is especially important for assets used for research or legal proceedings, as well as for assets with high value, like digital art or cryptocurrency.
While digital artifacts are changing frequently, provenance records shall usually remain unchanged. This is due to the nature of provenance, which documents past events that have already happened. Altering provenance records involves manipulating historical activities, which diminishes the value of provenance data. To ensure the quality, reliability and trustworthiness of provenance data, it is therefore desired to store provenance data tamper-proof. For this reason, the storage of provenance information using blockchain technology has become a topic of significant research interest over the past few years.
Blockchain is a revolutionary technology that has gained widespread attention in the last decade. IBM defines blockchain as a "shared, unalterable ledger that simplifies the recording of transactions and tracking of assets." [1]. The core attributes of blockchain technology include immutability and traceability, making it a promising opportunity to ensure the tamper-resistance of provenance data. While being mostly associated with cryptocurrencies as bitcoin, blockchain became a flexible technology, that is increasingly used for a variety of other use cases, such as supply chain management, real estate or provenance tracking.
This master thesis aims to investigate the use of blockchain technology for tamper-proof storage of provenance data. Speaking of provenance data, we will refer to the PROV standard [2] introduced by the World Wide Web Consortium [3]. PROV is a specification for representing and exchanging provenance information on the web. It provides a set of concepts and relationships to describe the provenance of resources, including entities, activities, and agents involved in their creation, modification, and usage. By using the PROV standard, we ensure interoperability and consistency in the representation and exchange of provenance data across different systems and platforms. This facilitates the integration of blockchain technology into existing workflows and systems for provenance management.

1.2 Goals

In order to accomplish its aim, this thesis will follow two goals:

1. Conduct a systematic literature review to identify existing works that utilize blockchain technology to store provenance information using a PROV-related standard.

2. Design and implement a blockchain-based provenance storage system for the manufacturing of virtual aircraft components.

1.3 Structure

The structure of this thesis is organized as follows:

Following the introduction, the thesis provides an overview of the relevant theoretical background. This includes a brief introduction to Provenance in **Chapter 2**. Provenance is defined and the PROV standard and data model are introduced. This includes an overview of basic types, relationships, and forming provenance graphs, serializations, and documents.

Chapter 3 introduces blockchain technology, which is the second theoretical bachground section. Since a good understanding of blockchain technology is necessary to assess their capabilities and possible applications, this chapter is slightly more comprehensive than a classical, reproducing theoretical chapter. It provide the groundwork necessary for designing an appropriate blockchain solution at a later stage. The Chapter will begin with a short overview over its development history, its capabilities and have a closer look of immutability. Between 3.3.-3.6 the chapter have a more technical look on the architecture, types of networks, concensus mechanisms and transactions. The chapter ends with a brief classification of blockchain use cases.

Chapter 4 presents the systematic literature review, which satisfies the first goal of the thesis. The SLR aims to identify existing works that store provenance information using blockchain and follows Kitchenhams et als approach. The chapter is structured according to the phases through which the review has passed. Those cover Search Strategy, Study Selection Criteria, Data Extraction and Synthesis. The chapter ends with a brief summary of the results, which compares the identified storage strategies regarding their properties and helps to identify an appropriate approach later.

Chapter 5 leads to the second part the thesis and provides a requirement analysis of the aimed provenance tracking system. The chapters follows the approach from Rupp and Pohl and aims to identify the requirements of the system by interviewing the stakeholders of the project. The first part presents the elicitation method, including questions prepared for the survey and the results of the survey. The second part documents the requirements using combination of model and natural language representation.

Chapter 6 proposes a concept, that aims to satisfy the defined requirements. The chapter is organized according to the critical components of the system. Additionally, to discuss an appropriate storage strategy and blockchain solution, it proposes a system architecture, the design of Smart Contracts, a Graphical User Interface, and Permission Management.

Chapter 7 propose a Software Prototype, which aims to proof the feasibility of the proposed concept. It gives an overview of the architecture and presents the implementation of a Smart Contract and GUI.

In **Chapter 8**, the Prototype is evaluated regarding the Requirements of the System. After describing the Testbed, to make our Environment reproducible, we test or discuss each requirement, and whether it is satisfied.

The thesis closes with a conclusion in **Chapter 9**, reflecting its results, and limitations and providing an outlook for future works.

2 Provenance

2.1 Definition

Provenance, originating from the French term provenir means to come from and refers to the original source or place of something [4]. Historically, the concept of provenance was utilized in determining the origin of food and art, as the source directly impacted the authenticity and quality of the product. However, with the emergence of information technologies, it has become possible to trace the origin of information, and as a result, provenance has extended into other fields. Jeff Jarvis notes that "good curation demands good provenance" [5] indicating that provenance is no longer limited to artists, academics, and wine makers but has become an expected ethic in various fields. Since provenance also became more popular in computer science, the World Wide Web Consortium (W3C) defined Provenance as "information about entities, activities, and people involved in producing a piece of data or thing, which can be used to form assessments about its quality, reliability or trustworthiness." [2]. Besides definitions, W3C defined a standardized data model and serializations for the representation and exchange of provenance data. In the following sections, we gonna present the PROV standard alongside its Data Model and components.

2.2 PROV Standard & Data Model

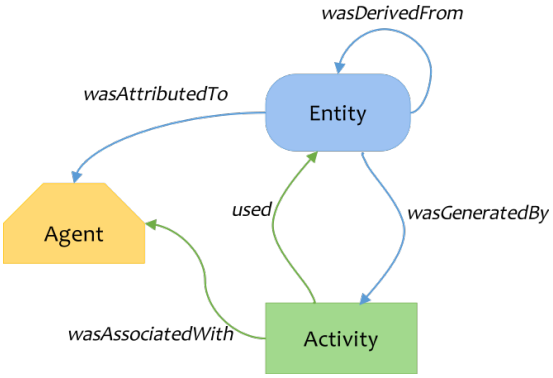
The W3C Provenance Working Group released the PROV standard [2] in April 2013, replacing the previous OPM Model [6]. The PROV standard provides a comprehensive framework that includes a model, serializations, and other supporting definitions, that are designed to promote the seamless interchange of provenance information in heterogeneous environments [2]. The PROV-Data-Model [7] defines formal semantics, which can be mapped into various representations, including PROV-O [8], PROV-N [9], and PROV-JSON [10]. PROV-O represents the model with the Web Ontology Language and RDF [8]. PROV-N is a specialized notation used to express provenance data in a more readable format and that uses constraints to validate provenance data against the standard [9]. PROV-JSON represents the PROV Data Model in JSON format. Its purpose is to enable easy manipulation of provenance information on the client side. By offering a concise and accurate representation of PROV, PROV-JSON allows for efficient data retrieval and is especially useful for exchanging PROV documents between web services and clients [10]
The PROV Data Model defines vocabulary to describe the Flow of data, processes, and responsibilities. PROV-DM consists of core structures that form the essence of provenance information [7]. The core structures are presented in Figure 1 and consist of three types and seven relations. A more detailed list that includes extended structures is provided in Appendix A..

Entities

Entities are the main type of PROV-DM. They describe the thing that provenance is collected about. An entity can be something physical, such as a book, a house, or a work of art, as well as something ideational, such as a data resource, a software application, or a concept.
An Entity in PROV-N is defined as `entity(id, [ attr1=val1, ...])`, where `id` represents a unique, mandatory identifier and is followed by an optional list of attribute-value pairs. There

	Name
Types	Entity Activity Agent
Relations	WasGeneratedBy Used WasInformedBy WasDerivedFrom WasAttributedTo WasAssociatedWith ActedOnBehalfOf

(a)



(b)

Figure 1: PROV-DM Overview of Core Types and Relations. Sources: [7] [11]

are predefined attributes, such as prov:type, which belong to the PROV namespace. More details about attributes and namespaces are provided in [7].

Activities

If entities are described as things that are, then activities are things that are happening. An activity may be coloring a house, sending a computer file, explaining an abstract concept, or driving a car from A to B. Activities are defined more formally as "something that occurs over a period of time and acts upon or with entities; it may include consuming, processing, transforming, modifying, relocating, using, or generating entities" [7].
An Activity is expressed by `activity(id, st, et, [ attr1=val1, ...])`. The identifier `id` is followed by the activities start `st` and end time `et`.

Agents

An Agent is responsible for the existence of an entity, for an activity taking place, or for another agent's activity. An agent that creates an entity might be a manufacturer that produces a car, an artist who draws a picture, or a non-government organization, that creates a human rights report. An agent that is responsible for an activity could be a text parser when reading a text file. An example of an agent taking place in another agent's activity might be a service that processes orders from a customer buying an article.
An Agent is defined as analog to an entity, by providing a unique identifier and a list of attributes: `agent(id, [ attr1=val1, ...])`.

Generation

The Generation relation represents the creation of a new entity by an activity. The entity is considered to have been generated by the activity. Examples of Generations are the production of bread through baking, the creation of a text file by a computer program, or the development of a concept through a critical thinking process.
Using PROV-N, Generation is presented by the following expression: `wasGeneratedBy(id; e, a, t, attrs)`. The `e` stands for the generated entity, `a` is the generating activity and `t` stands for the generation time. The only required attribute is e, which specifies the entity to create. All other attributes are optional.

Usage

This relation represents the use of an entity by an activity. The activity is considered to have used the entity. Examples of Usage may be using an ingredient for a baking activity, a program reading a CSV file as a dataset, or a service reading the value in an environmental variable. Usage is expressed as `used(id;a,e,t,attrs)`. The identifiers `a` and `e` specify the activity and the entity that it uses. The activity is the only required attribute.

Communication

The communication relation in PROV-DM is used to denote the transfer of an entity between two activities. It indicates that an entity was transferred from one activity to another.
The expression `wasInformedBy(id; a2, a1, attrs)` in PROV-N represents Communication. The `a2` identifier specifies the informed activity, that receives an entity. The other activity `a1` represents the sending activity. It is important to note, that the sent entity is not specified.

Derivation

When a new entity is created through the transformation or processing of one or more existing entities, the derivation relation in PROV-DM is used to represent this relationship. In PROV-N, this relation is expressed as `wasDerivedFrom(id; e2, e1, a, g2, u1, attrs)`. The identifier `e2` stands for the generated entity. `e1` is the entity used by the derivation. `a` stands for the activity using and generating the entity. `g2` is the generation involving the generated entity (e2) and activity (a). u1 specifies the usage involving the used entity (e1) and activity (a). The required parameters are the entities e2 and e1.

Attribution

The attribution relation in PROV-DM associates an entity with an agent. It is used when an entity is generated by an unspecified activity and will be associated with an agent. In PROV-N notation, this relation is expressed as `wasAttributedTo(id; e, ag, attrs)`. The required attributes are the generated entity `e` and agent `ag`.

Association

Association is similar to the Attribution relation, but associating an agent with an activity it is responsible for, instead of an entity. An association is defined as `wasAssociatedWith(id; a, ag, pl, attrs)`, where the activity `a` is the only required attribute. The optional identifier `pl` stands for the plan, the agent relied on. Plans are not among PROV-DMs core structures and are explained in detail in [7].

Delegation

The delegation relation in PROV-DM represents the transfer of responsibility for an activity to an agent. The responsibility can be transferred from another agent or itself. Delegation is presented in PROV-N by `actedOnBehalfOf(id; ag2, ag1, a, attrs)` and states that `ag2` acted on behalf of `ag1`. Both identifiers are required, while the activity `a`, for which the delegation holds, is optional.

3 Blockchain

This Chapter introduces Blockchain Technology. Section 3.1 provides a brief outline of the development history of Blockchain. In section 3.2 the features and capabilities of Blockchain are explained. Afterward section 3.3 takes a look at Blockchains architecture. Different types of Blockchain networks are distinguished and compared in section 3.4. Section 3.5 explains the most popular consensus mechanisms. The chapter closes with section 3.6, which proposes transactions and Smart Contracts.

3.1 Development History

Blockchain technology has its roots in the field of cryptography, and it was first proposed in 1991 by Stuart Haber and W. Scott Stornetta in their paper "How to Time-Stamp a Digital Document" [12]. The first implementation was proposed when the pseudonym Satoshi Nakamoto published the Bitcoin whitepaper "A Peer-to-Peer Electronic Cash System" [13] in 2008. Bitcoin is the first application that utilizes blockchain technology and became the largest and most widely used blockchain application [14]. The invention of Bitcoin is referred to as the first milestone of blockchain development. The evolution of Blockchain has progressed beyond the development of Bitcoin and can now be identified by four phases of transformation. The following sections give an overview of the 4 Phases, which are shown in Figure 2.

Blockchain 1.0: Digital Currency

The first generation of Blockchain including Bitcoin as a pioneering example is well-known for Cryptocurrencies, which established itself as the "cash for the internet" [16]. From a technical perspective, Bitcoin represents a noteworthy advancement in the realm of digital currency. Unlike conventional currencies and earlier digital currencies, Bitcoin distinguishes itself by not relying on any centralized authority, instead utilizing the concept of Distributed Ledger Technology (DLT), which involves a shared database among multiple participants. This fundamental shift eliminates the need for individuals to expend resources to establish trust, generating significant interest in Bitcoin and blockchain technology.
Bitcoin's reliability, efficiency, simplicity, independence, and security have enabled users to directly transfer authority and keep track of transaction records. By solving the Double-spending [17] and Byzantine General problem [18], the blockchain of Bitcoin effectively removes the primary barrier to the circulation of "digital currency." [14]. Consequently, numerous copycat digital currencies emerged [19] [20] [21], utilizing a three-layer technical architecture consisting of blockchain, protocol, and currency layers [14].
Blockchain 1.0 utilized the Proof of Work (PoW) consensus mechanism, which necessitated the completion of complex mathematical puzzles, resulting in high energy consumption and slow transaction approvals [16]. It was observed that Blockchain 1.0 could handle only seven transactions per second [16], leading to slow throughput. Additionally, Blockchain 1.0 was limited in its ability to support Smart Contracts and other application sectors beyond financial utilities.

Blockchain 2.0: Smart Contracts

The limitations of the first-generation blockchain, including wasteful mining, poor scalability, and its "inability to handle complex logic" [14], prompted the development of the second-generation

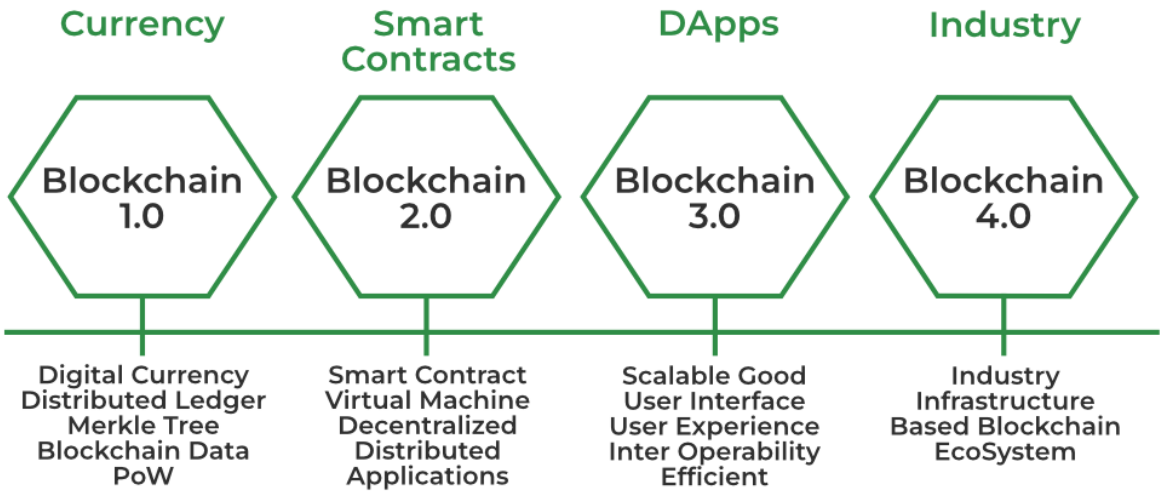


Figure 2: Phases of Evolution of Blockchain. Source: [15]

blockchain. Ethereum [22], the representative platform of this phase, is designed as a development platform based on blockchain. Users can create smart contracts, which are programs using Ethereum. The Turing-complete property of Ethereum smart contracts enables developers to upload any valid program and run it in the Ethereum Virtual Machine (EVM). As a result, Blockchain 2.0 gains flexibility and allows a variety of applications to run, such as voting systems, domain name management, financial transactions, crowdfunding, and smart property management [16]. Ethereum utilizes the Proof of Work consensus mechanism, where miners compete for Ether, a cryptocurrency used to reward miners for including transactions in their block. This way, the scalability problem of the first phase persists. In addition, there are costs for uploading code and data on the blockchain. Overall, Smart contracts provided an array of benefits to Blockchain 2.0, including accuracy, transparency, and fast execution speeds, making Ethereum attractive for several non-financial sectors such as e-voting systems, student evaluation systems, smart home schemes, and smart healthcare systems [16].

Blockchain 3.0: Beyond "Currency," Economy and Market

Blockchain 3.0 exploit the potential of Smart Contracts by building complete Decentralized Apps (DApps) [14]. Decentralized applications run on a network of computers within a blockchain network, rather than on a single computer. Consequently, they operate outside the control of any central authority. In a DApp, smart contracts are used to store the business logic and the related state of an application. This way the backend of a dApp is fully distributed and managed on a blockchain platform, making it resilient, transparent, and resistant to censorship.
Blockchain 3.0 employs a variety of consensus mechanisms such as Proof of Stake and Proof of Authority, which eliminate the need for separate transaction fees and enhance the speed and computing power of smart contracts. This new generation of blockchain, designed around the "FFM" concept [16], which stands for Fast, Feeless, and Minerless and seeks to improve upon the scalability, interoperability, privacy, and sustainability limitations of previous generations [16]. Unlike earlier versions, Blockchain 3.0 does not rely on miners to verify and authenticate transactions; instead, it employs built-in mechanisms for the same, resulting in extremely fast transaction speeds that can handle thousands of transactions per second. Additionally, the third generation of Blockchain has produced Directed Acyclic Graph (DAG) protocols, as described in [23] [24]. This platform does not rely on blocks, chains, or miners, but instead uti-

lizes a directed acyclic graph structure, where transactions are directly connected. DAG-based Blockchains promise fast confirmation and high scalability by avoiding competitive transactions caused by linear sequenced blocks [25]. While still in its early stages, Blockchain 3.0 has the potential to revolutionize the blockchain industry. At this stage, blockchain's potential role goes beyond monetary, economic, and market aspects and extends to social, industrial, and scientific fields [14].

Blockchain 4.0: For Business and Industry

The evolution of Blockchain technology continues to progress, with Blockchain 4.0 expected to become the next propitious step [16]. It aims to transform Blockchain Technology into a business-friendly platform that can create and operate applications, and support the accessibility of the technology for the mainstream. Blockchain 4.0 has the potential to integrate other prosperous technologies like Artificial Intelligence with Blockchain [16], facilitating a seamless integration of different platforms to meet business and industry demands. Unibright [26] and SEELE [27] are examples of Blockchain 4.0 platforms, that enable the integration of several blockchain business models and permit cross-communication between different protocols and various services [16]. The fourth generation has the potential to increase transactional speed to 1M transactions per second [16].

3.2 Features of Blockchain

As the development history shows, blockchain is a very new field that underlies ongoing development. There is now a multitude of implementations and types that differ in terms of their properties. Independent of specific implementations, the following features are attributed to blockchain [28] [29] [30]:

- Decentralization: All nodes in the network record transactions and have a local copy of the ledger, providing protection against single points of failure.
- Trustlessness: A trusted third party is not required to validate transactions, and nodes do not need to trust each other before transacting. The consensus algorithm validates and records transactions democratically
- Immutability: Modifying previous blocks invalidates all subsequent blocks due to one-way cryptographic hash functions, making it difficult to manipulate the blockchain.
- Non-repudiation: Transactions are cryptographically signed with a private key and can be verified by others via the corresponding public key. Transactions cannot be falsely initiated or denied by their originator
- Anonymity: Users can communicate with the blockchain by utilizing a randomly generated address that conceals their true identity
- Transparency: All nodes can access and verify transactions stored in the blockchain, providing public transparency.
- Traceability: Block headers are timestamped, allowing nodes to trace the origin of historical blocks

Although it is a fairly new technology, blockchain quickly became a trending topic outside of the tech communities. The rapid, cross-societal popularisation of new technology can be accompanied by over-optimism [31]. Claims about the capabilities of new technologies such as blockchain may be overstated, so it's important to assess them carefully. Here we want to take a

look at immutability since it is strongly desired for the storage of provenance data and therefore has an outstanding importance for our work.
According to Hasan et al. in "The Techniques and Challenges of Immutable Storage with Applications in Multimedia" [32], immutability means an unchangeable nature or quality given to an object. By definition, it prohibits any alterations to that object over time. In terms of data, this would imply that no modifications can be made since the data is permanent.
As Haubert et al. note in "Tamper-Resistant Storage Techniques for Multimedia Systems" [33], software-based solutions cannot provide complete protection against tampering if the storage medium itself is not immutable. Therefore immutability cannot be attributed to the blockchain. Its more precise, to consider blockchains as distributed storage with software protection mechanisms and algorithms, Those make blockchains strongly tamper-resistant, as modifying the data they contain is very difficult. Blockchain achieves tamper resistance through its decentralized nature and the utilization of consensus mechanisms. However, depending on whether it is a public, private, or hybrid blockchain, the level of anonymity and decentralization can vary, which will be discussed in section 3.4. Before that, the next section covers how the architecture of a blockchain is built.

3.3 Architecture

The actual blockchain is built as a sequence of blocks and is shown in Figure 3. Each block holds a reference to the previous block in the form of a cryptographic hash. The cryptographic hash of each block is derived by calculating the hash value of a block's header.

Hashing

Hashing is a cryptographic process that transforms a character string of arbitrary length into a fixed-length unique output [34]. This implies that regardless of the size of the input, a hash of fixed length is produced. Hashing is a deterministic process [35], meaning that the same input will always produce the same output. The process is non-invertible [34], which makes it practically impossible to retrieve the original input from its hash output.
In Blockchain, hashing is utilized to ensure the integrity of the data stored on the blockchain. Each block on the blockchain contains a hash of the previous block's header, which creates a chain of blocks that cannot be altered without affecting all subsequent blocks. Any attempt to change the data in a block will change its hash, making it invalid and breaking the chain. Additionally hashing is used to secure the blockchain against tampering and fraud. The hash function used in blockchain technology is designed to be collision-resistant [35], meaning it is difficult to find two inputs that produce the same output. Because of the Avalanche effect [36] a small change in the input should lead to a significant change in the hash output [35]. This makes it practically impossible to alter the data on the blockchain without being detected.
The hash method a blockchain utilizes varies between implementations. Bitcoin utilizes a two rounds of SHA-256 [37] [14], while in Ethereum Keccak-256 is used [38].

Structure of a Block

Each block contains a header and a body. The header stores the hash of the previous block. The only block without a predecessor is the first block in the chain, also known as the Genesis block. Besides the hash of the previous block, the header contains a timestamp, a Nonce, and a Merkle Root as shown in Figure 3. A Nonce is a random number that is generated as part of the process of mining a block. Miners try different nonce values and use them to calculate the hash of the block. The first miner to find a hash that meets certain criteria gets to add the block to the chain. The Merkle Tree is a hash binary tree, which is used to effectively store and summarize transactions inside a block's body. A transaction can be a contract, agreement,

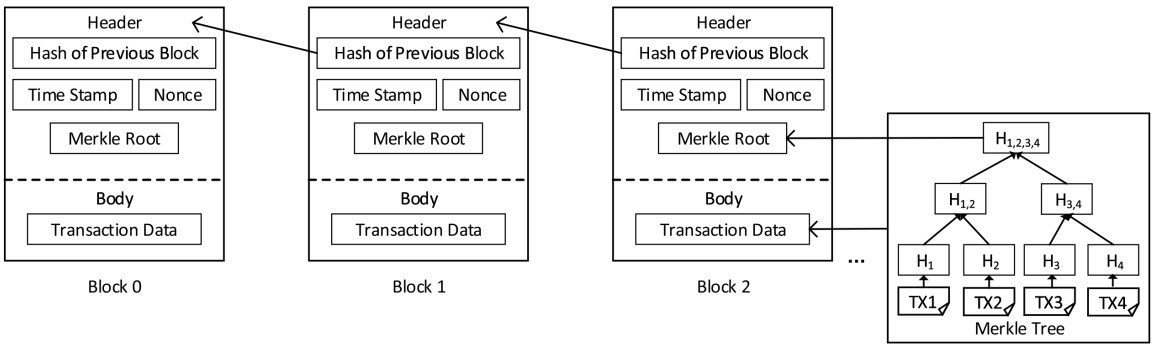


Figure 3: Sequence of Blocks of a Blockchain. Source: [29]

transfer, or exchange of assets between two or more parties. Transactions are covered in section 3.6. For now it is sufficient to know, that blockchains are mainly built to allow the immutable storage of transactions. In a Merkle Tree, each leaf contains the hash of a transaction. Parent nodes combine the hashes of their children into a new hash until the root node is reached. A merkle tree guarantees that if any transaction is changed, the hash of the Merkle root will change too. Since the Merkle root is stored in the header, it will change the the block's hash, which makes it simple to detect if a block's transaction has been modified.
The example in Figure 3 shows the most common and simple form of a Merkle tree, i.e. a Binary Merkle Tree. There are four transactions in a block: TX1, TX2, TX3, and TX4. Each of them is hashed and stored in a leaf node, resulting in H_1, H_2, H_3 and H_4. Consecutive pairs of leaf nodes are then summarized in a parent node by hashing H_1 and H_2, resulting in $H_{1,2}$, and separately hashing H_3 and H_4, resulting in $H_{3,4}$. The two hashes $H_{1,2}$ and $H_{3,4}$ are then hashed again to produce $H_{1,2,3,4}$, which is the Merkle Root. By including the Merkle root in the block header, the transactions become resistant to tampering.

3.4 Types of Blockchain Networks

The described blockchain structure is distributed across a network of nodes, with each node maintaining a copy of the blockchain. This makes blockchain a Distributed Ledger Technology (DLT) [39]. A distributed ledger is a database that is spread across multiple nodes or computers in a network. Each node in the network maintains a copy of the ledger, and any changes to the ledger are replicated and verified by all nodes in the network through a consensus mechanism. This ensures that the ledger is immutable and tamper-proof, as any attempt to modify or falsify data on one node will be rejected by the other nodes. Distributed ledgers are commonly used in blockchain technology, where they serve as the underlying data structure for maintaining and verifying transactions. Blockchain networks are distinguished by who can access them. The three main types are public, private, and hybrid networks. They are shown in Figure 4 and explained in the following.

Public

A public network, also known as a permissionless network, is a decentralized network [41] that allows anyone who meets the hardware requirements to participate in the network, validate transactions, and access the data stored on the blockchain [29]. Anyone can develop applications and services on top of the network. Bitcoin and Ethereum are two of the most well-known public blockchain networks. Public networks are highly resistant to tamper because the blockchain is distributed among a high number of nodes and utilizes a consensus mechanism to ensure that all participants agree on the state of the network [42]. They are transparent, as all participants can

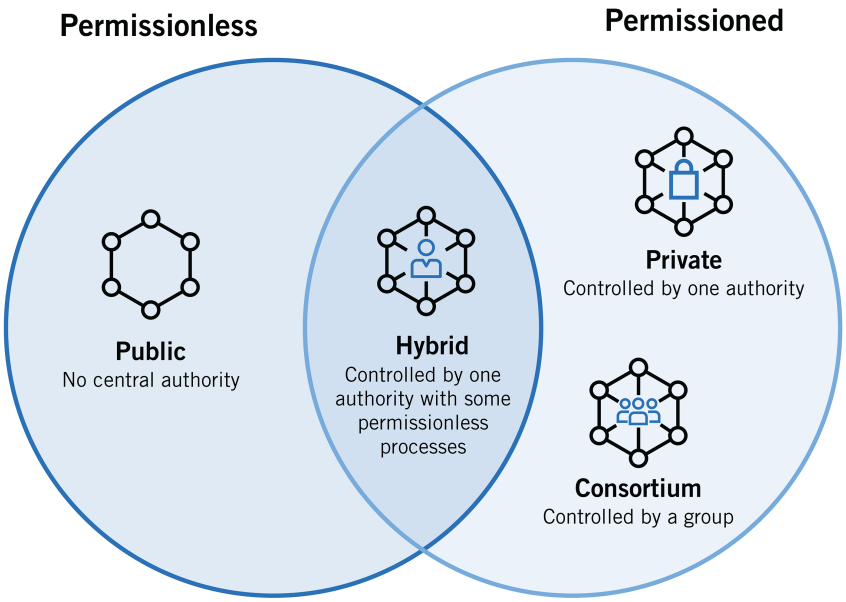


Figure 4: Public, Private and Hybrid Blockchains. Source: [40]

view the transactions and data stored on the blockchain. However, public networks can be slow and expensive, as a large number of nodes are required to validate transactions and maintain the network [43]. Reading and especially writing data can take longer and accompany by higher amounts of transaction fees, compared to private and hybrid blockchains.

Private

On the other hand, private networks, also known as permissioned networks, are usually maintained by a single organization [29]. The organization controls, who can access the network and validate transactions. Hence, private networks are accessible for a selected group of participants, that usually belong to the organization. Private networks are typically used by businesses and organizations that want to keep their transactions and data private. They are more efficient in terms of cost and speed [41] than public networks, as they have fewer nodes and can therefore validate transactions more quickly and usually utilize consensus mechanisms, that requires less computational power. They are also more secure in a way, as only authorized participants have access to the network. However, as they are typically controlled by a single entity, they have a single point of failure, which makes them more vulnerable to tampering compared to decentralized public blockchains.

Hybrid: Federated and Consortium

Hybrid blockchains, as the name suggests, are a combination of public and private blockchains. They provide a balance between immutability, decentralization, and performance. Federated blockchains are a type of hybrid blockchain that combines the decentralization of public blockchains with the performance and privacy of private blockchains. In a federated blockchain, the network is governed by a group of nodes, which are selected and trusted by the members of the network. The consensus mechanism in a federated blockchain is typically managed by the trusted nodes, which work together to validate transactions and maintain the network's state. The main advantage of a federated blockchain is its increased security, as the trusted nodes are selected based on their reputation and expertise, ensuring the security and integrity of the network. Federated blockchains also offer better performance and scalability compared to public blockchains, as the

network is controlled by a smaller number of trusted nodes. However, federated blockchains are less decentralized than public blockchains, as the network is governed by a small group of trusted nodes, and is therefore more susceptible to centralization and potential attacks. In addition, the selection of trusted nodes can be subjective, leading to potential biases and unequal representation in the network.
Consortium blockchains are another type of hybrid blockchain that combines the decentralization of public blockchains with the privacy and control of private blockchains. In a consortium blockchain, a group of organizations or entities collectively control and operate the network. The consensus mechanism in a consortium blockchain is typically managed by a limited number of nodes, which are selected and trusted by the consortium members. Consortium blockchains offer better privacy and control compared to public blockchains, as the network is controlled by a group of trusted organizations. However, consortium blockchains are less decentralized than public blockchains, as the network is controlled by a group of trusted organizations, and are therefore more susceptible to centralization and potential attacks. In addition, the selection of consortium members can be subjective, leading to potential biases and unequal representation in the network.

Comparison

After introducing different types of blockchain networks, we gonna compare their key features now. Table 1 compares the different types of blockchain networks. Public blockchains are designed to be decentralized and accessible to anyone. While this promotes transparency and immutability, it also means that all data stored on the blockchain is visible to all participants. This can lead to privacy concerns, as sensitive information may be exposed to the public. Therefore, data privacy on public blockchains is generally considered to be low. Private blockchains are designed to restrict access to a select group of participants, usually within a single organization. This allows for greater control over who can view and modify data, as well as increased privacy for sensitive information. However, the level of data privacy on a private blockchain largely depends on the access control mechanisms implemented by the organization. To join a hybrid network, permission needs to be asked first. The level of data privacy on a hybrid blockchain can vary from high to low and depend on the specific implementation. In terms of Immutability, data stored on a public blockchain is almost tamper-proof. That's because a public blockchain is decentralized and all the nodes of the network have to agree on a state. When a node tries to tamper a transaction inside a block, its hash changes too. Hence the previous block pointer in the next block won't match its hash anymore, the tampered block would be unchained. If other nodes in the network would recognize the unchaining, they won't accept the block. To make the tamper successful, the node would have to change the hash of the previous block in the following blocks as well. This is theoretically possible, but to be able to broadcast this block to the network, the node needs to have power over the consensus mechanism, i.e. by having more computational power or a higher stake than the rest of the network. Section 3.5 will give more details about possible attacks on blockchain networks. In private and hybrid networks, the risk of tampering is higher, because they are more centralized. In a private blockchain, the network is controlled by a single entity, which opens a single point to tamper the blockchain. Additionally, private and hybrid blockchains often use less robust consensus mechanisms, such as proof-of-authority, which gives more power to pre-selected nodes. Despite the higher risk of tampering, PoA allows adding blocks much faster, which makes blockchain applications more scalable. Since the participants in private and hybrid blockchains are usually known, in PoA particularly reliable nodes are selected as validators. This way achieving consensus involves less effort and the costs for generating new blocks are lower than in public blockchains.

	Public	Hybrid	Private
Access	Anyone	Ask for Permission	Selected Participants
Consensus	PoW, PoS, others	PBFT, PoA, Raft	PBFT, PoA, Raft
Immutability	Almost tamper-proof	Risk of tamper	Risk of tamper
Speed	Slow	High	High
Cost	High	Medium - Low	Low
Centralized	No	Partial	Yes
Anonymity	Yes	No	No
Data Privacy	Low	Medium	High

Table 1: Comparison between Public, Hybrid and Private Blockchains. Own representation based on: [41] [29] [43] [42]

3.5 Consensus Mechanisms

A consensus mechanism is a fundamental component of blockchain. Its purpose is to ensure that all nodes in the network agree on the state of the ledger. The consensus mechanism is responsible for verifying transactions, preventing malicious actors from compromising the network, and ensuring that the ledger remains secure and accurate. A consensus mechanism ensures that all nodes in the network have a common understanding of the state of the network and that malicious nodes are unable to manipulate the network. Several types of consensus mechanisms are used in blockchain technology, including Proof of Work (PoW), Proof of Stake (PoS), and Proof of Authority (PoA).

Proof of Work (PoW)

Proof of Work is the first and a still most widely used consensus mechanism in the blockchain. It was introduced by Satoshi Nakamoto in the original Bitcoin white paper [13] and builds the basis for the security of the Bitcoin network. In PoW, nodes in the network compete to solve a cryptographic puzzle, and the first node to solve the puzzle is allowed to validate the transactions and add a new block to the blockchain. As a reward, the node receives a specific amount of cryptocurrency. The cryptographic puzzle used in PoW is designed to be computationally intensive, making it difficult for a malicious node to manipulate the network. At the same time, checking if a solution solves the puzzle can be performed fast. The computational effort required to solve the puzzle is known as "work," hence the name "Proof of Work." The difficulty of the puzzle is adjusted periodically to ensure that blocks are added to the blockchain at a consistent rate, typically once every 10 minutes in the case of the Bitcoin network. The main advantage of PoW is its security, as it is computationally intensive to solve the cryptographic puzzle, making it difficult for a malicious node to manipulate the network. In addition, PoW is a decentralized consensus mechanism, as all nodes in the network have an equal opportunity to validate transactions and add new blocks to the blockchain. However, PoW is computationally intensive and requires a large amount of energy to validate transactions, leading to high energy costs and environmental concerns. In addition, PoW can be slow and inefficient, as it can take time for a node to solve the cryptographic puzzle and validate transactions. This can lead to long confirmation times for transactions and decreased scalability for the network.

51% Attack

In a Proof of Work (PoW) consensus mechanism, if 51% or more of the computing power in the network becomes malicious, the network is vulnerable to a 51% attack. In a 51% attack, the malicious nodes control the majority of the computational power in the network and have a high chance to perform a succeeding attack [44].

One possible attack vector is a double-spend attack [44] [45]. In a double-spend attack, the attacker may mine a block, but withhold the solution instead of broadcasting it to the network. By not sharing the solution, the attacker maintains a local version of the blockchain, which is isolated from the rest of the network. Hence, there exist 2 versions of the blockchain: The clean one, which is accessed by the uncorrupted nodes, and the isolated version of the corrupted node. The corrupted node continues working on his local copy, i.e. adding blocks to it while buying an expensive product on the clean version. This transaction is only included in the clean version, but not in his local one. When the attacker can make his chain longer, than the public owns, he can broadcast it. Blockchain protocols like Bitcoin are designed, to follow the longest chain. Hence, the attacker could broadcast a blockchain to the network, that reverts his spending and allows him to spend that amount again.
To be able to mine blocks faster than the rest of the network and win the mining race is why a higher mining power is necessary to successfully attack Proof of Work Networks. The required computing power rises with the depth of the fork, which makes older blocks practically immutable [44]. Consensus attacks can only affect future blocks or at best the last 10 blocks [44]. Also, Bitcoins cannot be stolen, spent without signatures, or redirected through a consensus attack. An attacker cannot simply create and send transactions between arbitrary nodes, as long as he doesn't know their private keys. A 51% attack can have serious consequences for the security and stability of a PoW network. This is why networks need to have a decentralized distribution of computational power so that no one group or entity can control 51% of the network.

Proof of Stake (PoS)

Proof of Stake is a consensus mechanism that is designed to address the limitations of PoW. In a PoS system, nodes in the network validate transactions based on the number of tokens they hold or their "stake". The more tokens a node holds, the more likely it is to be selected to validate transactions and add new blocks to the blockchain. The main advantage of PoS is its efficiency and low energy costs, as it does not require intensive computations to validate transactions. Due to its lower hardware requirements, the PoS consensus process is more accessible and less exclusive to high-end computing resources. However, PoS also has several disadvantages. It is more centralized than PoW, as nodes with a larger stake are more likely to validate transactions and add new blocks to the blockchain. In addition, PoS can also lead to a concentration of power and wealth in the hands of a few large token holders, leading to potential centralization and inequality in the network. While being less energy-consuming than PoW, Proof of Stake can be more prone to several attacks [46]. To protect a PoS-based Blockchain from those, additional rules are necessary, which increases the complexity of those systems.

Proof of Authority (PoA)

Proof of Authority (PoA) is a consensus mechanism used in some blockchain networks. Unlike Proof of Work (PoW) and Proof of Stake (PoS), which rely on computational power and token ownership, respectively, PoA relies on identity and reputation. In a PoA network, nodes are not required to solve complex mathematical problems or stake tokens to validate transactions and add them to the blockchain. Instead, a small group of trusted nodes, called validators or authorities, are responsible for validating transactions and adding them to the blockchain.
In a PoA network, the validators are selected based on their reputation and credibility within the network. They are usually individuals or organizations with a proven track record of being reliable and trustworthy. The validators are responsible for signing transactions, which proves that they have validated and approved them. Once a certain number of validators have signed a transaction, it is added to the blockchain.
One of the benefits of PoA is that it is faster and more efficient than PoW and PoS. Since validators are pre-selected and trusted, they can quickly validate transactions and add them to

the blockchain without the need for extensive computational power or staked tokens. This makes PoA a good choice for private blockchain networks, where efficiency and speed are important.
However, one of the drawbacks of PoA is that it is less decentralized than PoW and PoS. Since the network relies on a small group of validators to validate transactions, it is vulnerable to centralization and censorship. If a validator becomes malicious or is compromised, they could potentially harm the network by validating fraudulent transactions and adding them to the blockchain. Additionally, if the validator nodes are controlled by a single entity or organization, they could potentially collude and harm the network by censoring transactions or manipulating the blockchain.
To mitigate these risks, some PoA networks implement rotation or randomization of validator nodes. This ensures that no single validator has too much power or influence over the network. Additionally, some networks have governance models that allow participants to vote on changes to the network or the selection of validators. This helps to ensure that the network is run in a fair and transparent manner.

Validator takeover attack

While PoA is generally considered to be more efficient than PoW and PoS, there is still a risk that a group of validators could collude to harm the network. This could occur if a group of validators decide to maliciously validate invalid transactions or attempt to censor legitimate transactions.
For example, in a PoA network with 10 validators, if a group of five validators collude to validate invalid transactions or censor legitimate transactions, the network could become vulnerable to attacks and the integrity of the blockchain could be compromised. This type of attack is known as a "validator takeover attack" and it can be difficult to detect and prevent.
To mitigate the risk of a validator takeover attack in a PoA network, it is important to carefully select and monitor validators and to have mechanisms in place to detect and respond to suspicious behavior. Additionally, some PoA networks implement a rotating validator selection process, in which validators are periodically rotated to prevent collusion and centralization.

3.6 Transactions

Transactions are a key component of blockchains. The whole Blockchain structure is there to ensure that transactions can be "created, propagated on the network, validated, and finally added to the global ledger of transactions" [44]. Transactions allow the transfer of values from one address to another. This way, Alice can transfer as many Bitcoins to Bob, as her credit balance holds. Essentially a transaction is a digital messages that contain the necessary information to be executed. A transaction typically includes the sender's address, the recipient's address, the amount of cryptocurrency being transferred and a digital signature. The digital signature is generated using the sender's private key and serves as proof that the transaction was authorized by the sender. When a transaction is created, it has to be signed by all the parties who's funds are involved and needs to be validated before it can be added to the blockchain. This is where the consensus mechanism comes into play, which we discussed in section 3.5. As a short refresher, a consensus mechanism is a set of rules that govern how transactions are validated and added to the blockchain. In a network with PoW, the transaction is broadcast across the network, where each node verifies it against a checklist of criteria [44] and forward it to their neighbours, until the transaction reached the whole network. This way invalid transactions are discarded at the first node that encounters them [44]. From the valid transactions, each node maintains a transaction pool of unconfirmed transactions. After a mining node propose a block including the transaction and after each node has verified it against another list of criterias [44], the transaction is stored on the blockchain permanently and cant be changed anymore. The relocated funds can be spent

inside new transactions, which which in turn must be signed and validated before they extend the blockchain as part of new blocks. The described transaction, that transmit value from one account to another is the essence of blockchain technology's cryptocurrency use-cases and was already included in Blockchain 1.0.

Smart Contracts

Many of today's Blockchain implementations allow transactions to include programs called smart contracts. Smart Contracts were introduced by Ethereum during Blockchain 2.0 and extend Blockchain technology by supporting the execution of computer programs. Smart Contracts are often referred to as self-executing contract with the terms of the agreement between buyer and seller being directly written into lines of code. The code and the agreements contained therein are stored and replicated on the blockchain network. But Smart Contracts are not limited to transferring assets between parties. A smart contract is a collection of code and data [47], that can be used for a wealth of different applications, such as information management and executing business logic.
Smart Contracts can be implemented flexible in programming languages such as Solidity or Go. To illustrate how a smart contract can look like, Figure 5 gives a practical example of a Smart Contract implemented in Solidity. The example contains `counter.sol`, which is a simple counter application. Line 1 shows a machine-readable SPDX-identifier to indicate its license. If no license shall be given or if the code is not open source, developers can write "UNLICENSED". More information about SPDX can be found in [48]. The second line specifies, which compiler should be used to call this contract. By specifying ^0.8.17 the contract accept any version starting from 0.8.17 and below 0.9. Line 4 defines the contract with the identifier Counter. A contract has some similarities to a static class. Line 5 defines a public count attribute of type unsigned-integer, where the value of the counter application is stored. The public keyword allows the method to be called from inside and outside of the contract. Other options are private, internal and external. Line 8 - 10 define a public getter function, that returns the actual count. This function is not nescessary, since Solidity automatically creates getter functions for public variables. The view keyboard indicates, that the method don't modify the state. View methods cant modify state variables, create other contracts or send ether. The inc method in line 13 - 15 is meant to increment the state by 1. Hence it is modifying the state, it is not a view method. Since it dont return anything, it dont have the return keyword the header. The decrement function in line 18 - 20 works analoge to increment. All methods and classes are opened and closed by curly brackets.
In Ethereum Smart Contracts introduce two more types of transactions: Deployment and Execution (Call) [50]. After a contract deployment transaction is stored inside a blocks list of transactions, it cant be changed. The Contract is deployed to a account address, from which the contract can be referenced and its methods can be called. Since the blocks are immutable, methods cannot change attributes, unless they are packed inside a new transaction. Thus, the current state of a smart contract can be understood from the successive execution of all transactions that call that contract.
Blockchains that allow Smart Contracts come with a Virtual Machine, that enable Turing complete code to run on the blockchain. Virtual Machines calculates the state of the network after a block has been added to the network. Transactions can have different impacts on the networks state. A regular transaction is likely to change the amount of currency, e.g. reducing the amount of Ether that Alice's holds by the amount that she has sent to Bob. A contract call can influence the data stored inside a contract, i.e. the values assigned to a variable.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.17;

contract Counter {
    uint public count;

    // Function to get the current count
    function get() public view returns (uint) {
        return count;
    }

    // Function to increment count by 1
    function inc() public {
        count += 1;
    }

    // Function to decrement count by 1
    function dec() public {
        // This function will fail if count = 0
        count -= 1;
    }
}
```

Figure 5: Smart Contract of a Counter Application `counter.sol`. Source: [49]

4 Systematic Literature Review

One of the most famous use cases for Blockchain technology is its application in the cryptocurrency industry. Bitcoin, the first and most well-known cryptocurrency, was created in 2009 using Blockchain technology to provide a decentralized and secure alternative to traditional financial systems. Since then, numerous other cryptocurrencies have been developed, relying on Blockchain technology to function. However, the potential applications for Blockchain technology extend far beyond cryptocurrencies. Smart contracts have opened up a world of possibilities for Blockchain technology by making it much more flexible for a wide range of use cases, such as Supply chain management, Real Estate, and Provenance Tracking. In this chapter, we will take a closer look at the storage of Provenance information on Blockchain. To be precise, we will conduct a Systematic Literature Review using the approach proposed by Kitchenham [51], which provides a rigor, reproducible and clear methodology. By analyzing the literature, we aim to identify possible ways to store Provenance on Blockchain that can later be applied to our use case.

Background

Provenance information is important for a wide range of applications and industries, including art, finance, supply chain management, and digital manufacturing. Provenance information refers to data that describes the origin, ownership, and history of an asset, such as a piece of art, a financial transaction, a product in a supply chain, or a designed component. This information can be critical for ensuring the authenticity, integrity, and quality of the asset, as well as for verifying compliance with legal and ethical standards. Storing Provenance information on a Blockchain can provide a tamper-evident and permanent record of this data, which can help to ensure the integrity and reliability of the information. Additionally, because a Blockchain is a decentralized system, it can enable multiple parties to access and verify the Provenance information without requiring a central authority. This can increase transparency, trust, and accountability in the Provenance data. For this reason, many studies have proposed the use of Blockchain for storing Provenance information. Through this SLR we aim to gain a deeper insight and an overview of existing studies in the field.

Goal

To state it clearly, the Systematic Literature Review aims to identify existing works that use Blockchain technology to store Provenance information using W3C-PROV or a related standard.

4.1 Search Strategy

Keyword Search

To find existing works that use Blockchain technology to store Provenance information using a PROV-related standard we first searched for articles, that contain Prov AND Blockchain in their Title, Abstract, or Keywords:

TAK(PROV AND Blockchain)

Searching directly for PROV is restrictive because some articles and PROV modifications put a prefix or suffix around PROV. By adding an asterisk, we can make our search more flexible. Asterisk (*) represents any group of characters, including no character. Hence, we tried *PROV, PROV* and *PROV*. It turned out that PROV* and *PROV* were adding too many results since PROV is included in many words such as prove or provide. However, *PROV seemed effective since PROV is added as a suffix to frameworks such as BSTPROV or ETHERPROV:

*TAK(*PROV AND Blockchain)*

A search with the query *PROV AND Blockchain delivers 24 results over the four databases (duplicates included). This query assumes, that the authors specify in the title, abstract, or keywords which standard they use. However, some articles refer to the general term Provenance and specify the used encoding in the paper. As we changed our query to the more general query Provenance AND Blockchain, our number of results increased to 217 for just one database. Checking the resulting articles manually, whether they encode Provenance with the PROV or a PROV-related standard, would be too costly with that number of entries. That's why we extended our query, in a way that allows also papers that mentions the more general term Provenance in combination with Blockchain in their Abstract while mentioning the PROV in any other field. Using this query, we found 33 unique articles in four databases. The final query looks as follows:

*TAK (*PROV AND Blockchain) OR (Abstract(Provenance and Blockchain) AND ALL(PROV))*

The databases used vary depending on the allowed parameters and syntax. Table 2 shows the queries used and the number of results per database.

Database	Query	#Results
Web of Science	((TS=(*PROV AND Blockchain)) OR (TS=(Provenance and Blockchain) AND ALL=(PROV)))	3
Scopus	(TITLE-ABS-KEY(*prov AND Blockchain) OR (ABS(Provenance AND Blockchain) AND ALL (prov)))	32
IEEE Xplore	(("Document Title":*PROV AND "Document Title":Blockchain) OR ("Abstract":*PROV AND "Abstract":Blockchain) OR(("Author Keywords":*PROV AND "Author Keywords":Blockchain) OR ("Abstract":Provenance AND "Abstract":Blockchain) AND ("All Metadata":PROV)))	5
ScienceDirect	TAK (PROV AND Blockchain) OR (Provenance AND Blockchain) AND ALL(PROV)	4

Table 2: Search Results of Queries among Databases

Backward Search

The keyword search enables the collection of articles from the desired research area. Despite the use of different databases, relevant articles may not be found due to the limitations of the keyword search. One way to find more relevant articles is to look at the referenced articles. With the help of the Open Citations API, we collected the articles that are referenced most by the articles in our collection. Therefore, for each article that appears in the references, we count in how many articles of our collection it is mentioned. We looked at each article that appeared more often than the median value, thus at least three times. Apart from standardization documents

about W3C and Hyperledger Fabric, we discovered two more articles, that haven't been in our collection and added those. Hence we found a total of 35 studies, which are listed in Appendix B.

4.2 Study Selection Criteria

1. The Study must be accessible.
2. The Study must be available in English or German language.
3. The Study must encode Provenance according to W3C Prov or a related standard, like OPM.
4. The Study must either
   a) store Provenance model using a Blockchain or
   b) provide a theoretical concept on how a Provenance model can be stored on a Blockchain.

Due to our Study selection criteria, we had to eliminate 15 studies and continue our Systematic Literature Review with 20 remaining studies. Which study has been selected, is documented in the last column of Appendix B. The Data Extraction proceeds with the 20 remaining studies.

4.3 Data Extraction

According to Kitchenham, the Data Extraction form must be designed to "Collect all the information needed to address the review question" [51]. In our case we want to identify existing works that use Blockchain technology to store PROV-related documents. Therefore, we are looking for fields that will help us group the work in this research area in a meaningful way. It should be added that, in contrast to the intended case of Systematic Literature Reviews, we do not primarily collect numerical values. Of course, numerical values can be compared better, however different architectures for storing Provenance information can be captured more comprehensibly via text.

Goal What are the aims of the study?

Application area What's the domain of the entities, that Provenance is tracked (e.g., Medicine or Finance)?

Blockchain What Blockchain is used?

Provenance Standard How is Provenance Information encoded?

Provenance Data Generation How is Provenance Data created?

Blockchain Storage How is Provenance Data stored on the Blockchain?

The first step in our systematic literature review is to extract the aim of each study. Understanding the overall goal of the research is crucial as it justifies the claim and methodology of the work and helps to inform other relevant fields such as the Blockchain technology used and access permissions. Additionally, we extract the application area of each study to group our findings and identify areas where storing Provenance information on the Blockchain is particularly popular. We also pay attention to the Blockchain technology used in each study, as this allows us to understand which platforms are commonly used for this purpose. As we move on to the process of storing Provenance information, we analyze how the information is encoded, and document any processes used to generate Provenance documents if they are not received in a PROV format. We then examine the methods used to store the information on the Blockchain, allowing us to compare and contrast different approaches.

4.4 Synthesis

This literature review examined a total of 20 studies that mainly propose systems for tracking and storing Provenance information using Blockchain technology. An Overview gives the Data Extraction Table in Appendix E. The studies explored used different Blockchain platforms, such as Hyperledger Fabric and Ethereum, and different W3C PROV-related Provenance standards such as SWPROV and OPM. The majority of studies used Hyperledger Fabric as Blockchain. Among 20 studies, out of which 16 named the Blockchain they used, 11 choose Hyperledger. The authors highlight Hyperledger Fabric's modular architecture as a key factor in their decision [52, 53][54]. This architecture grants flexibility in configuring and plugging in components, making it well-suited for use in industries such as banking, finance, and health insurance that require frequent innovation and flexible domain logic. Additionally, the authors point out that Hyperledger Fabric uses general-purpose programming languages (GPL) such as Java, Go, and Nodejs, which makes it more accessible and easier for developers to work with compared to other platforms that use domain-specific-languages (DSL). Finally, the authors mention that Hyperledger Fabric is a permissioned Blockchain platform, which is essential for organizations that want to keep their data private and should not be exposed to public audiences. Besides Hyperledger Fabric and Ethereum, Tendermint and Tieron API are used. The application areas are broadly distributed across Open Data, Food Delivery, Internet of Things, and Healthcare.
In the study conducted by Kirstein[55] in 2019, the author defined requirements for a system that tracks and stores Provenance data in the context of linked open data. The W3C PROV standard was used in this study to guide the development of the proposed system. The next steps to develop the system were outlined in the study, but haven't been carried out.
Similarly, Hogan and Helfert [56] in 2019 investigated the application of distributed ledger technology (DLT) in the tracking and storage of Provenance data theoretically. Using the W3C PROV standard as a reference, the authors mapped PROV components to Blockchain elements and evaluated the alignment between the PROV data model and the Bitcoin implementation of Blockchain. The study found that while a simple alignment existed between the two, certain relationships such as WasInformedBy, WasDerivedFrom, and ActedOnBehalfOf could not be expressed in a single linked list DLT such as Blockchain. This was considered a key finding, as it suggests that data Provenance instances which rely on these circular reference relationships may not be implementable on a Blockchain.
In the most proposed systems for tracking and storing Provenance information, data is stored directly on the Blockchain. Ioni and Pahl [57] propose a Blockchain container-based architecture to track identities and Provenance of orchestration decisions of a business network using the W3C PROV standard. They store the PROV components as classes in smart contracts, mapping entities to assets, agents to participants, and activities to transactions. Similarly, Fadhel et al. [58] modified PROV-N to model attacks towards IoT Devices, storing documents in Hyperledger Fabric's keystore.
Markovic et al. [59, 60] in 2019 and 2020 use Provenance to keep track of food deliveries, storing FS-PROV data in smart contracts that are exposed via a RESTful API [60]. They also use an EP-PLAN ontology, an extension of the PROV model for linking plans [60]. Dang and Anh [52] in 2020 use a similar approach, storing data in five smart contracts, which combine to form the actual Provenance information. To track Provenance in the context of Open Data, their tool receives a synchronization request from Open Data Platform, whenever a user changes data. Then the received context changes of the Objects properties result in Provenance data. Each action creates a new "chapter" of Provenance information, resulting in a Provenance Story.
In 2021, Dang and Duong [53] continue this line of research, using the W3C PROV standard to track changes in open data and storing the URL and checksum of the dataset in the Blockchain.
Song et al. [61] in 2020 also focus on tracking and sharing Provenance data, using the W3C PROV standard and storing relevant data, such as a person's identity signature and the type and time of an operation, in the body of the Blockchain in order to detect tamper. To generate

Data Provenance, they mapped relevant components such as Assets, Actions and Initiators to the W3C PROV Elements Entities, Activities and Agents.
In the field of healthcare, Margheri et al. [62] in 2020 track data Provenance using the W3C PROV standard. They consider the underlying health documents as Entities of PROV documents. Their approach is to canonicalize the PROV documents and store key-value pairs of a document's hash and Provenance information in the Blockchain. Similarly, Lautert et al. [63] in 2020 focus on fog computing and use the W3C PROV standard, storing Provenance information in private Blockchains for each fog, which are later shared with a global Blockchain.
Sigwart et al.[64] in 2020 propose a generic framework for tracking Provenance in IoT systems, using a data Provenance model for IoT and storing relevant data in the Blockchain [64]. This framework includes a storage layer for low-level representation and storage of Provenance data, a generic Provenance layer for general-purpose functionality, and a specific Provenance layer for fine-tuning the framework to the requirements of a specific use-case.
Demichev in 2021 presents a decentralized data management system using Hyperledger Fabric and an adapted version of the W3C PROV standard [54]. This system, called PROVHL, stores key-value pairs of assets such as files in the Blockchain. The assets, operations or transactions, and participants are all mapped to the adapted W3C Prov elements.
Additionally, Zhang et al. [65] in 2021 proposed a system for dispatching and control of Provenance data in power grids. This system uses Hyperledger Fabric and a modified version of the PROV standard for power-grid systems. To store data, they use three smart contracts: one for data access, one for data logging, and one for data Provenance. The data logging contract creates a Provenance chain based on the activity, such as initialization, deletion, update, or query, and the data Provenance contract accesses the data operation log to create a complete Provenance chain.
In recent studies, researchers moved on from storing Provenance data directly on the Blockchain. Bose et al.[66] point out that there are challenges in building decentralized Blockchain applications for data-intensive applications, as storage of data on some Blockchain platforms like Ethereum can be costly due to cryptocurrency costs. Additionally, Sun et al. [67] argue that Provenance data cannot be embedded in blocks as a whole because its size may exceed the block limit, and it may include sensitive information that can only be legally accessed by specific users. Thus, alternative storage solutions are being proposed to address these challenges.
Liang et al. [68]in 2017 proposed a system called ProvChain, which uses the Tierion API to track cloud operations while storing the actual Provenance Data in a Database. The system generates Provenance data in the form of Data Records, which include information such as the date and time of the operation, the username of the user who performed the operation, the name of the affected file, and the action performed (e.g., file creation, modification, or copy). The Data Record is then published to the Blockchain using the Chainpoint standard, which combines the hashes of the Data Record elements to form a Merkle tree.
In the field of knowledge management, Ramachandran and Kantarcioglu [69] proposed in 2018 a system that uses Ethereum to collect and verify data Provenance. The system utilizes the Open Provenance Model (OPM), which represents Provenance data as a triplet of agents, artefacts, and processes (e.g. (user, file: old version, file: new version, process used for modifications). The system also includes a Document Tracker that logs change event data and stores the included OPM data in a database.
Tunstadt et al. [70] proposed 2019 a general framework for tracking data Provenance that utilizes the Open Provenance Model. The system stores data on-chain and off-chain. On-chain data includes checksum, editors, operations, data ownership, and data pointers, while off-chain data includes the actual data.
In the field of software Provenance, Bose et al. proposed in 2019 a system that uses Ethereum to capture, store, explore and analyze Provenance data [66]. The system utilizes the SWProcess Specification (Prov for Software Development) and generates Provenance data by converting

review data to SWProcess. While storing the critical parts of PROV Data and its hashes on the Blockchain, full PROV data is stored in an Off-Chain database. They also suggest the use of cryptographic immutable distributed databases such as IPFS to store full Provenance data.
Coelho et al. [71] in 2021 proposed a system for capturing, storing and analyzing Provenance-related data in collaborative research. The system uses ProvONE (PROV for scientific workflows) as the Provenance standard.
Sun et al. [67] in 2022 proposed a system for sharing Provenance data using Ethereum and the W3C PROV standard. The system partitions the PROV graph into several subgraphs using a BFS-based Provenance Graph Partition method, encrypts the subgraphs, and uploads them onto the Blockchain. Users can then obtain a subset of the Provenance subgraphs and compose them into a new graph. The system stores the Provenance subgraphs, overall dependency structure, and access control policies on-chain and the local Provenance graph, information of the transaction and block where a subgraph is embedded in and access control policies off-chain.

4.5 Comparison of Storage Strategies

The SLR has identified various strategies for storing Provenance information using Blockchain. To find out, which approach is suitable for our interest, we first gonna summarize the different approaches and secondly reflect on them critically, regarding their advantages and disadvantages for storing and retrieving W3C PROV documents. Having a closer look at the approaches, the first thing to distinguish is where the data is stored. One approach is to directly store data on the Blockchain, while others solely keep the hash on-chain and refer to a database or URL to obtain the actual data. Additionally, there are differences in the structure of stored Provenance data. One way is to store the entire W3C PROV document or its hash, although some works combine documents beforehand or divide them into subgraphs or elements like entities, agents, and activities and then store their interrelationships. The units differ depending on the Provenance standard used. We classify the approaches based on the location and structure of the stored data in Table 3.
Most of the works are considered to store Provenance data in smart contracts on the Blockchain. Storing data directly on-chain has several advantages, which are summarized in Table 4. Data stored on-chain is immutable and cannot be altered once it is stored, which provides a high degree of trust and transparency in the data. In decentralized networks, stored on-chain data is replicated across all nodes in the network. This means that there is no single point of failure, and the data is always available as long as the network is up and running. Since data stored on-chain is distributed and cryptographically secured, it is highly resistant to tampering and hacking attempts, which is a major advantage over traditional database systems. However, due to its nature data on the Blockchain is accessible to everyone. The transparency of data can cause privacy issues, i.e. when sensitive data is stored and information shouldn't be visible to other participants of the network. Additionally storing data on-chain also have two major downsides: Speed and Cost. Blockchains have a slow writing speed and are also not particularly fast when reading data. Thus, even low-performance databases can surpass the speed of Blockchains and to this point, there is no known Blockchain, that is able to compete with databases in terms of speed. Uploading large amounts of data can take an extremely long time, making storage more error-prone. If speed is an important factor, on-chain storage is impractical. When considering

	Document-wise	Canonicalized	Element-wise	Subgraphs
On-Chain	[63] [59] [60] [58]	[62]	[52] [65] [57] [61] [64] [54]	
Off-Chain	[66] [53] [70] [69]		[68] [72]	[67]

Table 3: Overview of PROV Storage Strategies

	On-Chain	Off-Chain
Immutability	✓	○
Dezentralization	✓	○
Data Integrity	✓	✓
Cost	−	+
Speed	−	+
Privacy	○	✓

Table 4: Comparison of Storage Methods: On-Chain vs Off-Chain

a public Blockchain, writing to the Blockchain usually requires a payment. The cost of writing depends on the size of the data being stored and the particular Blockchain in use. In terms of the Ethereum Blockchain the cost to store 1 kilobyte is 640 gas, which is around 0.20€ in March 2023. While the cost for one kilobyte seems quite affordable, the cost of storing 1 megabyte is around 20€. When transaction fees are that high, in use cases where expenses matter, it doesn't make sense to store more data than necessary directly on the chain. Instead, approaches use the possibility of storing intensive amounts of data outside the Blockchain, for example in databases or InterPlanetary File Systems (IPFS). The Blockchain only keeps the hash value of the stored data along with a reference to the data, e.g. in the form of an ID, URL or IPFS address. In this way, the downsides of on-chain solutions can be mitigated. Since the actual data is stored off-chain, the solution is more scalable in terms of speed and cost and can handle large amounts of data. Also the privacy of data can be preserved, by controlling access to the external file system or database. Storing the hash value of data allows for the detection of any manipulation, thus ensuring the integrity of the data. However, in the event of data tampering, the ability to restore the referenced data and maintain version history will determine whether or not the data can be recovered. Depending on the security of the hash method chosen and whether the hacker knows the hash method and the hash of the data, which he wants to modify, there is also a risk that the change will not be detected. If the hacker manages to modify the data in such a way that the hash remains the same, the tamper may remain undetected. However for SHA-256, which is the most prominent hash function in Blockchain, finding a document that would produce the exact same hash is already extremely hard, due to the avalanche effect and 2^{256} possible hash-values of SHA-256. Another weak point of storing data off-chain is that, depending of the type of storage, it might open up a single point of failure. When the data, which is referenced by the Blockchain is stored in a centralized database, the decentralized nature of Blockchain cant protect the data. As stated before, the Blockchain can only ensure to detect if data has been tampered, but cannot restore it.

When comparing the structure of stored Provenance documents, significant differences occur between document-wise storage and element-wise storage. The differences are summarised in Table 5 Because their properties are similar, we combined document-wise storage and canonicalized documents to the first group and element-wise storage and subgraphs to the second one. Storing a document as a whole is pretty straightforward and is associated with low computational effort. Imagine a smart contract, that would store a whole Provenance document. As stated earlier, storing the data on-chain is pretty inefficient, so that approach might only be suitable for little graphs. The element-wise storage and retrieval of documents involve computational effort. Before storing a document, single identities or subgraphs need to be revealed and stored individually, together with their relationships, to make it possible to combine them later again. When considering on-chain storage, data storage can be more efficient, since components that are used among multiple documents can be reused, instead of stored in every single document. Also uploading tiny bits of data to multiple smart contracts, instead of storing everything in one, might be less sensitive to failure during the upload. However, when considering off-chain storage, a lot of references and hashes need to be stored to utilize the benefits of element-wise storage.

	Document-based	Element-based
Computational Effort	-	+
On-Chain Storage Cost	++	+
Off-Chain Storage Cost	--	-
Off-Chain Diff	×	✓
Tamper Risk	-	--

Table 5: Comparison of Storage Methods: Document-based and Element-based

This multiplies the number of stored hashes by the number of separated elements. Therefore, while element-wise storage can reduce the Blockchains data load compared to document-wise storage in an on-chain scenario, it is the opposite when storing data off-chain. When tampering happens off-chain, element-wise storage is able to provide a diff from the previous version, by identifying which element has been modified. This is not possible for off-chain document storage. Additionally, the risk of tampering is slightly higher in document-wise storage, because only one hash needs to be cracked. Storing data element-wise creates multiple hashes. This makes it difficult to change the content of a document in a meaningful way.

5 Requirement Engineering

In order to lead the development project to success, the research institute's involvement in the project is crucial to ensure that the new blockchain-based system meets the specific needs of their application area. In order to determine these needs, we employed the systematic approach from Pohl and Rupp [73] to requirement engineering.
According to Pohl and Rupp a requirement can be described as a "necessary need of a stakeholder" [74], a "capability or property that a system must fulfil" [74] or a "documented representation of a need, capability or property" [74]. Requirements engineering is a systematic and disciplined approach to specifying and managing requirements with the goal of understanding stakeholder wants and needs and minimizing the risk of delivering a system that does not meet those wants and needs. According to Pohl and Rupp [75] three types of requirements are typically distinguished:

1. **Functional Requirements** concerning the results or behaviors that shall be provided by a function of the system.

2. **Quality Requirements** refer to a quality characteristic that is not covered by functional requirements. They are often related to functional requirements and can specify them further. Examples are requirements that refer to the Performance, safety, reliability, or usability of a system.

3. **Constraints** restrict the solution space beyond what is necessary to meet the functional requirements and the quality requirements. They can refer both to the system under consideration (e.g. "The system should be realized by blockchain") and to the development process of the system (e.g. "The system should be finished until June 2025").

Following Rupp's approach, the first step of Requirements Engineering is the elicitation of requirements. There are various strategies to elicit Requirements. Section 5.1 will present our approach and give a comprehensive summary of the results. The second step is the documentation of requirements. Therefore section 5.2 document the wealth of information into well structured requirements. According to Rupp there are two more phases: Validation and Managing. The third phase of validation was ensured by coordinating the process of identifying and documenting requirements with stakeholders. After Requirements have been formulated, they were presented to stakeholders and discussed. The management of the requirements has played a subordinate role, as the project had only one processor and thus the requirements did not have to be shared.

5.1 Elicitation of Requirements

We used stakeholders as our main source to elicit requirements, because stakeholders have a vested interest in the project and are directly impacted by the system or process being developed. By using stakeholders as a resource, we can gain insights into their needs, goals, and expectations, which are critical to developing a system that meets their requirements. As a technique to gather requirements, we decided to conduct an interview, because compared to a questionnaire, it allows us to ask specific follow-up questions in case of ambiguities or previously unknown topics. When conducting an interview, it must be taken into account that basic factors, i.e. features that have already been fulfilled by a previous system, are taken for granted by stakeholders. To counteract

ID	Aspect	Question
Q1	Purpose	What is the purpose of the blockchain-based system that you want to build?
Q2	Users	Who are the intended users of the system and what are their roles?
Q3	Data	What data does the system need to store and manage, and what are the security and privacy requirements for this data?
Q4	Interactions	How will users interact with the system and what actions are expected to take place?
Q5	Scalability	What is the expected scalability of the system in terms of the number of users, number of actions, and data storage?
Q6	Integration	Are there any existing systems or processes in place that the blockchain-based system needs to integrate with?
Q7	Performance and Availability	What are the performance and availability requirements for the system?
Q8	Success Criteria	What are the success criteria for the project and how will they be measured?
Q9	Regulatory and Compliance	Are there any regulatory or compliance requirements that the system needs to meet?

Table 6: List of Interview Questions

this shortcoming of interviews, we included questions regarding the previous system and the current storage. From previous communication, we had some information about the project, that we used to estimate the scope of the desired system and prepare the interview.

Interview Preparation

The scope of the system is to design and build a blockchain-based system that stores provenance data on the blockchain. The stakeholder is a research institute, that put us in contact with one experienced researcher, whose directly involved into the application area of the new system.
As part of our preparation for the requirement elicitation process, we developed a set of questions to guide our interview. When determining the categories for requirement elicitation, it is important to ensure that they cover all aspects of the system that are relevant to the stakeholders. We have identified nine categories in Table 6 by drawing on our experience and knowledge of system design and requirements engineering. To facilitate comprehensive and detailed responses during the interview, each category has a specific open-ended question. The categories address a broad range of topics essential for any software system, such as system goals and objectives, user needs, data management, user-system interaction, system scalability and integration, performance and availability requirements, project success criteria, and regulatory or compliance requirements. Our categories align with established principles and practices in the requirements engineering and system design field. For example, the IEEE 830 standard [76] for software requirements specification recommends functional requirements, performance requirements, design constraints, and external interfaces. Additionally, our categories align with the widely used Volere Requirements Specification Template [77], which includes stakeholders, project drivers, functional requirements, non-functional requirements, constraints, and acceptance criteria.

Interview Conduction

The interview was conducted as a video-interview and has been split in two sessions. The conversation was recorded by mutual consent and transcribed later. We begin the interview by asking the first question about the purpose of the blockchain-based system. This opening question aimed to gain a first impression about the initial situation, the environment and the contribu-

tion that stakeholders expect from the system. The stakeholder mentioned in his response that the system should be realised as part of the VPH process. In order to gain deeper insights into the process environment, we followed up with asking, what the VPH process is and inquired about the process steps and their content.
After that we moved on with Q2, which aimed to identify the different types of users that interact with the system and their roles. The expected users reveals who needs access to the system and help to design access control policies.
Question 3 addressed, What data needs to be stored and managed and how the security and privacy requirements for this data looks like. The types of data that the system has to store affect the design of smart contract attributes and optionally functions, to obtain those from an input. The data's security and privacy requirements can be tackled by the choice of the blockchain platform, since it determines how accessible the data is to public and the decision about which data is stored on the chain.
The fourth question aimed to learn how users should interact with the system. The answer helps to design the functionalities of smart contracts and the user interface, which they are called from.
Q5. asked for the number of users, transactions and data storage. The three parameters specify the required scalability of the blockchain system. The expected number of users affects the size of the blockchain network and can help to determine wether creating a new network is eglible. The required scalability for transactions and data storage will affect the choice of consensus mechanisms.
The sixt question aimed to learn whether there are any existing systems or processes in place that the blockchain-based system should be embedded into. This would affect the interfaces the systems need to provide and which data it can expect and deliver in order to interact with its environment.
Question 7 asked for the performance and availability requirements. The desired Performance is crucial for the design of the network architecture, since it depends on the size of the network and the consensus mechanism. While estimating the estimated lifespan of the network infrastructure, the availability must be taken into consideration.
In Q8. we inquired about the success criteria and how it will be measured. Defining success criteria and how they will be measured is important for determining the value of the project and ensuring that it meets the goals and objectives of the research institute.
The last question Q9. aimed to learn about regulatory or compliance requirements that the system has to meet. Regulatory and compliance requirements can have a significant impact on the design of the blockchain system, including the choice of consensus algorithm, data storage, and security measures. Knowing these requirements will help ensure that the system meets all necessary standards and regulations

Interview Results

The interview revealed that the goal of the project is to create a blockchain-based system to track the VPH-process. VPH stands for Virtual Product House (VPH) and is a Integration and Test Centre for Virtual Certification of aircraft components [78]. Developing an aircraft component is relatively expensive and when a production team realizes that a component doesn't meet the requirements, it has to repeat the whole designing, building and test process. The development effort is associated with high costs that can turn out as a barrier that prevent future innovations. To give an example, if a company thinks that introducing a new component could result in a 2% fuel saving, but they must produce it to confirm, the potential financial risk may be deemed too great, leading them to opt out of pursuing the idea. To reduce costs and risks in the production of aircraft components, VPH aims to digitalize the aircraft component manufacturing process. The process is shown in Figure 6. It consists of four phases, which are Digital Design, Virtual Manufacturing, Simulated Testing and Certification. The initial three phases takes place inside

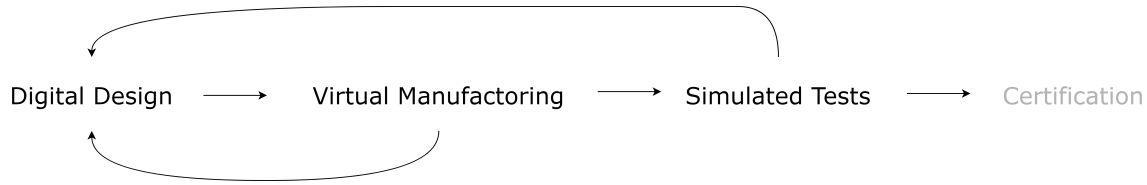


Figure 6: VPH Process

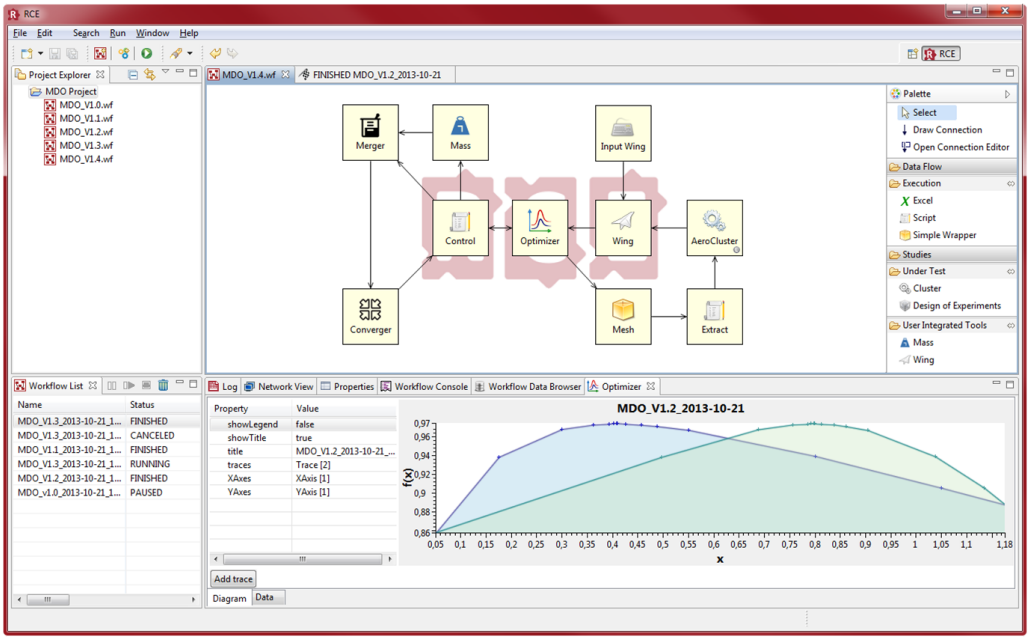
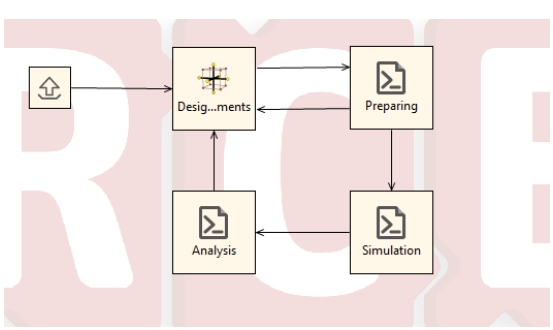


Figure 7: Graphical User Interface of RCE. Source: [81]

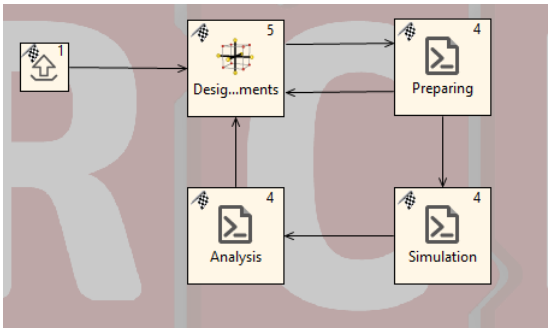
a open-source software called RCE (Remote Component Environment) and it will be our main objective to assist in the certification process by documenting these steps thoroughly. Before heading there, lets have a look at RCE first.

RCE

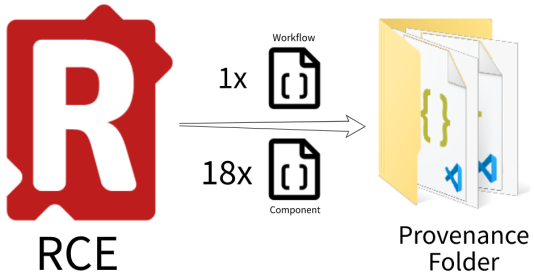
RCE is a "distributed, workflow-driven integration environment. It is used by engineers and scientists to analyze, optimize, and design complex systems" [79] and was published by the German Center of Aerospace [80]. To improve the readers understanding, Figure 7 shows the Graphical User Interface of RCE that Engineers interact with. On the left side we see the project Explorer on the top and a list of Workflows on the bottom. The right side shows an editor palette on the top. The remaining space at the bottom shows the properties of the currently selected component. In the center we can see a workflow called MIDO_V1_4. A workflow is made up of components, which are represented as squares and can either be user-integrated tools or multi-purpose functionalities provided by RCE. These components can be coupled by building connections between them, which allows a sender component to transmit data to one or multiple receiver components inside the same workflow. The illustrated Workflow starts with an Input Wing, which sends data to the Wing. From the Wing data gets transmitted to an Optimizer, which invokes a loop. A loop is executed multiple times until the driver component, here the Optimizer, finishes it based on some certain criteria. After pressing the Run Button, the Workflow starts its execution. RCE Workflows can get much more complex than in this example. However, for our purpose its sufficient to keep a basic understanding of RCE, without diving to deep into the details.



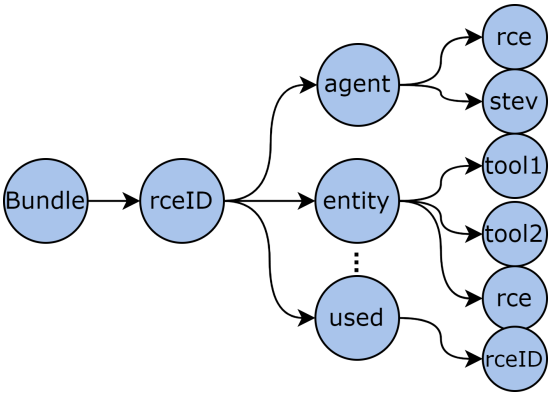
(a) Pre-Execution Workflow



(b) Post-Execution Workflow



(c) Overview of exported Files



(d) Structure of a Components JSON-File

Figure 8: Generation of Provenance Data from RCE

Use-Case, Goals and Users

Engineers exploits the variety of tools inside RCE to compose workflows that model the entire VPH-process. As shown in Figure 9 the process begins with the Digital Design, where an engineer build and store a digital prototype of an aircraft component. During the Virtual Manufacturing, that takes place afterwards, different operations like simulating carbon fibres, infusion with synthetic resin and curing taking place. If standards are violated by deformation of components, it is possible to return to the design process at low cost. If all requirements are met, the process continue with the third Phase. During Simulated Testing components are runs through finite element simulation. If the tests fail, its easy to return to the Digital Design again. Otherwise the last phase is reached. In Certification the tests and simulation results have to be verified by a responsible authority. Since the authority decides on the approval of the component and a defective component poses a high risk to flight safety and human life, it is essential that the inspection is carried out conscientiously and that the entire development process has been documented in a comprehensible manner. To create a documentation system which accomplish this task, is the overall project goal. Therefore the intended users of the system are engineers of a research institute and the authorities that are responsible for certification. The number of expected users is between 5-20 for the research institute and between 1-10 for the authority. It must be possible to add new users flexibly. It is desired that the engineers, who work on the VPH-process can store their results on the documentation system. When the engineer find a design configuration that pass the virtual manufacturing and the simulated testing, he will send a certification request to the certifier. The certifier should be able to check these work results from the documentation system.

Data

Lets have a closer look at how the work result, i.e. the data looks like. The first three phases inside the Remote Component Environment are Workflows, which are stored as a RCE internal file format on the engineer's local system. Having a copy of this file allows a certifier to open and execute the whole workflow on their local system if they installed RCE. However, to ensure traceable documentation, it is desirable to determine whether the workflow has been documented in the system and which employee produced it at what time. This can be achieved by exploiting the RCE-Provenance-Plugin. To understand how the Plugin works and what data it delivers, we take a look at an example. Figure 8 shows the Process of when and how Provenance Data is generated. In 8a we can see an example workflow in RCE. It comprises 5 components, with one input and one loop around the Design Experiments component. After executing the workflow, we can see the result in 8b. The small number in the top right corner of each component indicates how often a component has been run through the workflow. Summed up we had 18 component runs in total. This number indicates how many Provenance Documents are exported. When we look at 8, we can see the 19 files got exported from RCE. These are 1 JSON file for the whole workflow and 18 JSON files for each component run. The JSON files are stored automatically in a folder on the local filesystem, where the instance of RCE was running. Each file contains provenance information about the run of a component or the whole workflow. To understand the structure of the Documents we can have a look at 8d. Each document starts with a bundle node, that contains an ID of this run. Under the ID we can find the Provenance information encoded according to the W3C PROV Standard. The diagram is only intended to provide an overview, but it is worth mentioning that the produced documents usually contain all types and relationships of PROV that are presented in Chapter 2. A complete Provenance Document for a WorkflowNode is shown in Appendix C.

5.2 Documentation of Requirements

By employing survey techniques, the previous section gathered a large amount of information about the use case and the requirements of the system. In order to structure the wealth of information, natural language and model-based representation methods can be used, each with their respective advantages and disadvantages [73]. Natural language allows the expression of all types of requirements without the need to learn a notation, however it can be ambiguous or misleading. On the other hand, model-based representations permit the isolated examination of a system from different perspectives and avoid misunderstandings, but they require the learning of at least one and sometimes even several notation forms. In order to take advantage of the benefits of both approaches, Pohl and Rupp [75] suggest using hybrid forms, where one or more models are combined with natural language requirements. Figure 9 shows a hybrid model for the underlying use-case, using BPMN [82] notation to visualize the use-case and Rupp's sentence template [74] to describe the requirements by natural language. The requirements were derived by reviewing the notes taken during the interview and highlighting key points related to the system's goals, user needs, and other categories we specified in Table 6.

5.2.1 Functional and Quality Requirements

As a result of requirements eliciting, we have learned that the stakeholders' requirement is a documentation platform. The platform is connected to two parties: The research institute, which drives the development of aircraft components, and the authority, which is responsible for the certification of aircraft components. The processes within and between both parties, the research institute and the certifying authority, have been described in the previous section and are summarized in the hybrid model, shown in Figure 9. To support the documentation process, the documentation platform shall provide engineers with the ability to upload W3C PROV docu-

ments, keep record of the uploaded documents and provide certifiers with the ability to check, whether a document has been uploaded, as noted in F1 - F3 of Figure 9. The engineers, that are involved in the manufacturing are not fixed. The same goes for the staff of the certification authority. In order to be able to react flexibly to changes in staffing situations, it is the fourth functional requirement to enable both institutions to create new user accounts. Moving on to quality requirements, according to Q1 the system shall ensure that only valid W3C PROV-JSON documents are uploaded. The documents of interest are provenance documents exported by the RCE-Provenance-Plugin and have the PROV-JSON format by default. Besides that, there is no external data that is intended to store, hence we believe it would improve data quality by restricting the upload of unwanted file formats. It can happen by mistake that the wrong data is selected and uploaded. Therefore, such a constraint is an additional protection mechanism for the uploading engineer.
Requirement Q2 states that the system shall provide a user-friendly interface to interact with, that does not require any knowledge of programming languages. Accessibility is important because the users have varying levels of technical expertise and the system should be straightforward to use for all of them.
Q3 focuses on the need for the system to capture and save metadata about the upload of data, specifically the responsible engineer and timestamp. This information is critical for tracking changes and identifying accountability for documents that have been uploaded. Having an understanding of who has been uploaded and when can be an important component of tracing the development process for the certification authority.
The fourth quality requirement Q4 specifies that only authorized persons within the organizations shall be able to add data. Only members of the research institute shall be able to have this permission, as they are the only party that creates workflows. In order to maintain system reliability and trustworthiness, it is critical that no other person can upload data.
Q5 emphasizes the need to protect sensitive data and ensure that no private information or provenance data becomes publicly accessible. It is a serious concern and also legally relevant to protect employees' identities. The system should not expose any personal data that could identify the individual.
Q6 specifies the need for the system to ensure that stored data remains available for at least 80 years. Based on the interview, it appears that documents pertaining to the development of approved aircraft components must be retained for this period of time. It highlights the importance of longevity and long-term storage of the data, as it may need to be accessed and referenced for many years to come.

5.2.2 Constraint: Why Blockchain?

Our system's primary constraint is the use of blockchain, which brings up the inquiry of why utilizing blockchain instead of a conventional database. We could trust our stakeholders, but in our perception is vital to ask such questions because trends and buzzwords surrounding new technologies can lead to unrealistic expectations and over-optimism among stakeholders. Often, technology vendors and solution providers make promises of significant benefits, cost savings, and growth, which may not necessarily align with the actual characteristics and requirements of the system. Therefore, it is crucial to be cautious and critically evaluate whether a blockchain is a right fit for a use case, to prevent unnecessary spending, disappointment, and wasted resources.
A good reason to use blockchain is the absence of trust between the parties involved. The research institute and the certifying authority can be considered as counterparts. The certifier is obliged to thoroughly check that the development process and the component itself meet all requirements. Otherwise, the certifier can be held responsible later when certifying a faulty component. Critically questioning and not trusting the work of the Institute is a part of his responsibilities. The research institute on the other hand invests work, time and money in the development of a component and has an economic interest in a fast and successful certification.

Every delay and problem costs money and submitting documents and waiting can be experienced as bureaucratic. Additionally, there is generally a lot of mistrust regarding public administrations in Germany. In a study conducted in the Summer of 2022 around 34% said that they do not trust public administrations [83]. Public institutions are seen as disorganized, slow, and inefficient. Therefore, problems and delays in certification can be blamed on internal work errors and lost documents, instead of mistakes by the engineers themselves. A conventional database may not be the optimal choice in this case as there is a risk of data tampering, loss, or manipulation, which can lead to trust issues between the parties. In contrast, a blockchain-based solution with immutable and secure data storage can establish trust and allow both parties to track and quote uploaded data, making it more suitable for the creation and certification of aircraft components. Immutability ensures that the data uploaded to the blockchain cannot be altered or tampered with, providing an indisputable record of the work performed by the engineers. This is especially important here, where the safety of the end-user depends on the quality and reliability of the product. Furthermore, data sharing is necessary because the engineers of the research institute need to share their work results with the certifying authority. This sharing of data ensures that the certifying authority has access to all the information necessary to make an informed decision about the quality of the aircraft components.

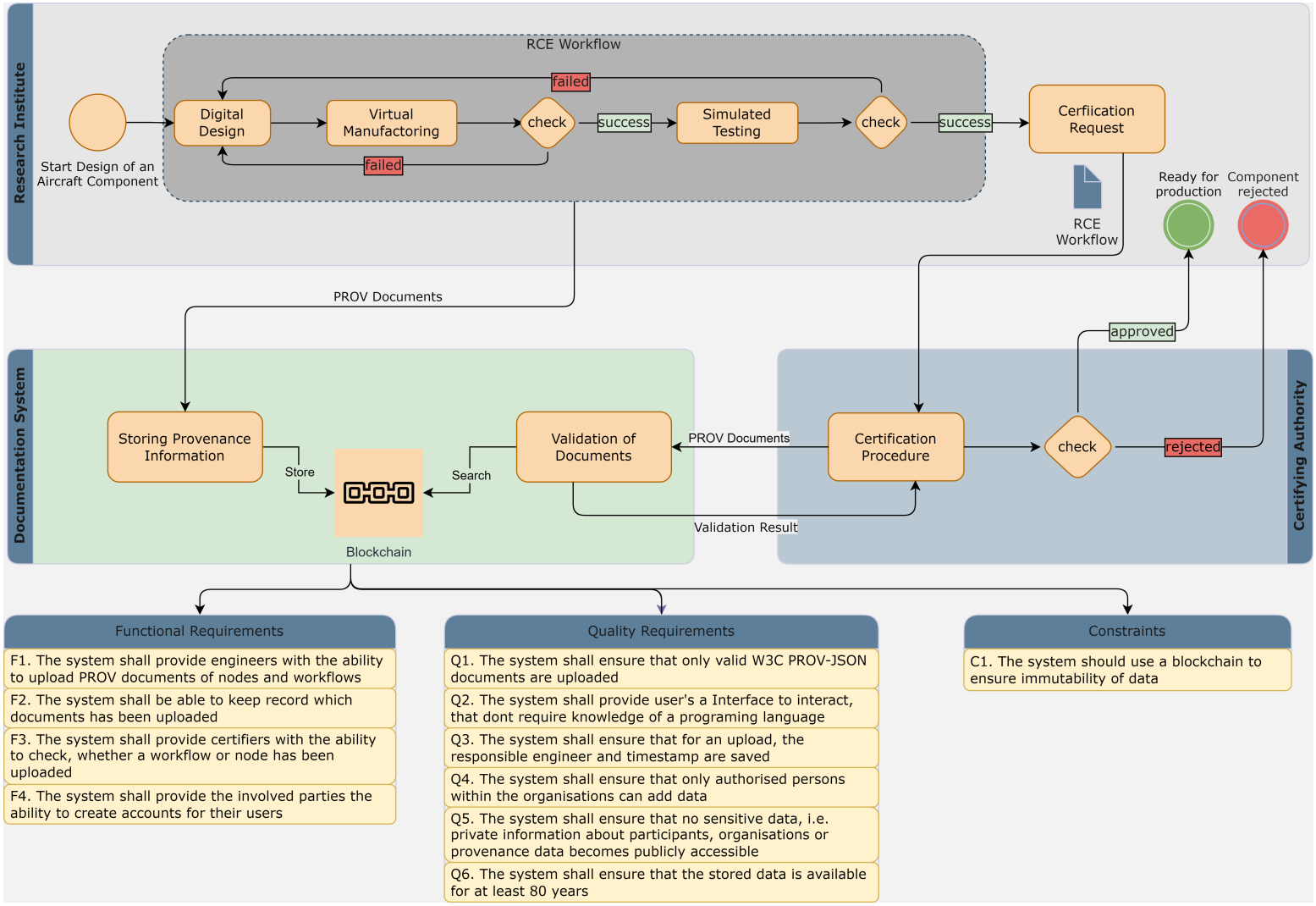


Figure 9: Mixed Documentation Model: Use Case Diagram and Requirements

6 Concept

This chapter presents a concept for a blockchain-based documentation platform that aims to provide tamper-resistant storage of PROV documents. It is intended to give a high-level overview of the system and to discuss design decisions behind the platform, with a focus on the non-technical aspects of the system.
The proposed architecture in section 6.1 introduces the components of the system and establishes a connection between the requirements of the system and the proposed components that strive to satisfy them. The following two sections, 6.2 and 6.3 provide an in-depth discussion of the storage type for PROV documents revealed in the Systematic Literature Review (SLR) and the blockchain technology most appropriate for the use cases based on the theoretical background. Section 6.4 proposes a smart contract structure, which aims to design methods and variables necessary, to ensure that the functional requirements are met. Afterward, section 6.5 proposes a user-friendly GUI to facilitate easy interaction with the system. The permission management is handled in section 6.6. Overall, this chapter provides a comprehensive blueprint for the development of a blockchain-based documentation platform, with a focus on practical applications and user experience.

6.1 Architecture

We designed the architecture of our blockchain-based system according to the "form follows function" principle. The components that we have chosen to include are intended to contribute to meeting the requirements outlined in the previous chapter. In the following, we aim to give an overview of the components, that are discussed in detail in the following sections and explain how they are related to the requirements. The architecture is presented in Figure 10. To meet the four functional requirements, we introduce a Smart Contract and a Graphical User Interface. Smart Contracts are the common way to implement and deploy logic on the blockchain. After deployment, the Smart Contract is stored as bytecode inside the Blockchain, which is distributed across the network. To simplify the interaction with the blockchain and meet requirement Q2 we provide a Graphical User Interface. In order to interact with the contract and read its bytecode from the chain, the GUI accesses the contract's Application Binary Interface (ABI) code. Uploading PROV documents (F1), providing the ability to check whether a workflow has been uploaded (F3) and the ability to create accounts for users (F4), are functionalities included inside the Smart Contract and the GUI. The document store lies inside the Smart Contract (F2), which makes sure that the data is stored on the blockchain. To prevent duplicate, redundant storage and preventing another attack vector, the GUI doesn't maintain a data store and is mainly a simple interface to interact with the contract and access its functions. Additionally, the GUI implements the functionality to encode documents and check if they have the valid format (Q1). We'll execute those steps in the GUI to reduce the computation that run on the blockchain. Additionally its more convenient to implement more complex logic in the GUI, because it can be implemented in a general-purpose programming language. Compared to domain-specific programming languages like Solidity, GPL's are more versatile and provide a collection of libraries and packages that simplify the processing, validation and encoding of provenance data. Using blockchain was a constraint of the system. Since blockchain stores the sender of a transaction and the timestamp, it provides the information necessary to satisfy requirement Q3. To ensure that only authorized persons within the organizations can add data (Q4), the smart contract func-

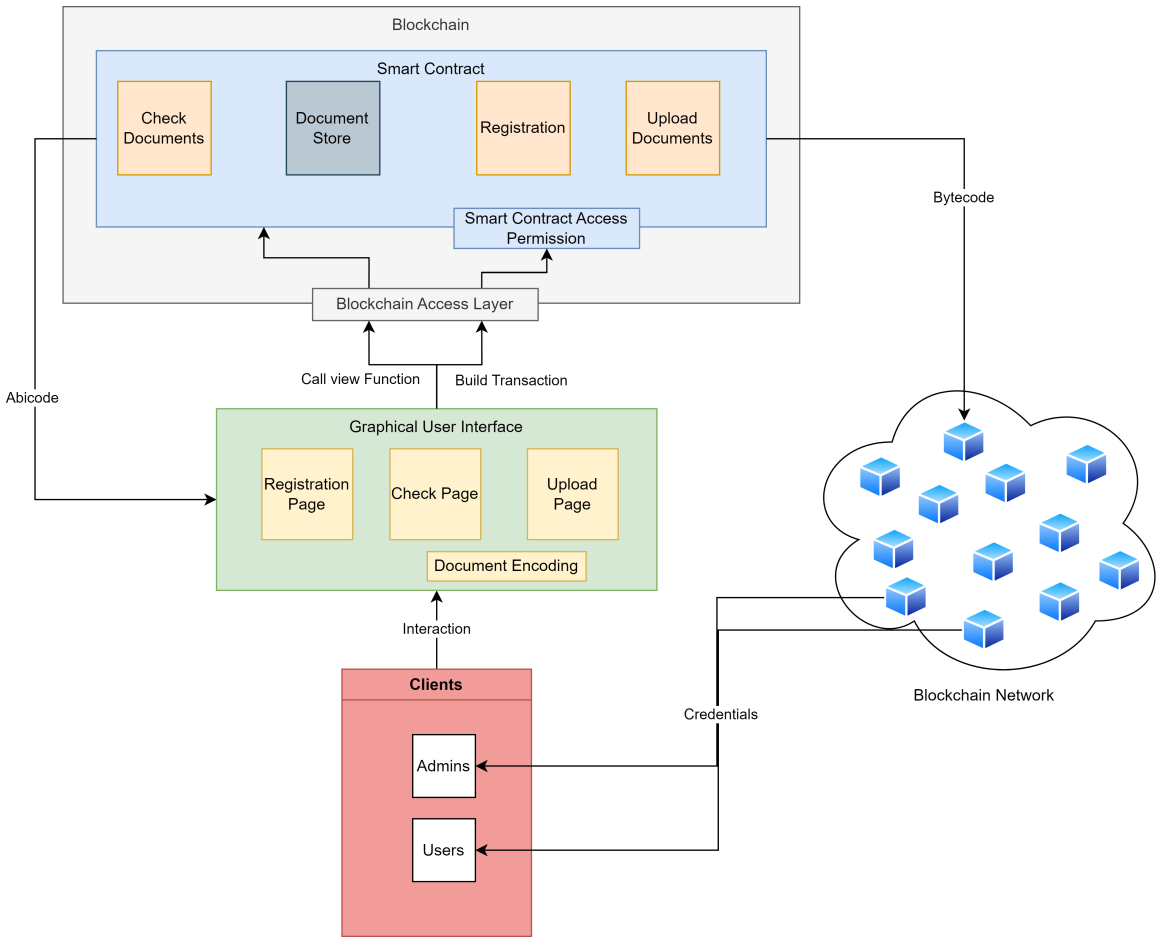


Figure 10: Architecture of the System

tions are secured by a role-based permission management system. The permission management restricts uploading documents to the intended users. Users are authenticated by their credentials on the network. Whenever a user wants to upload a document using the GUI, he has to pass the private key that only the user itself knows. Additionally, to users, there are admins, that have the right to register users and admins. Both organizations are issued one admin initially, who holds the responsibility to manage the users of the organization. The permission management is explained further in section 6.6. The requirement Q5, which is to ensure that no sensitive data becomes publicly accessible, affects the decision of how PROV data can be stored on the blockchain and is discussed in section 6.2. By relying on the blockchain authentication system, which identifies users according to cryptographic keys, we avoid exposing personal identities to the public. The only remaining requirement is that stored data should be available for at least 80 years. While being persistently stored on the blockchain, we cannot guarantee that the lifetime of the network will succeed 80 years. However, through the smart contract it's possible to extract data from the data store at any time, which can later be used to migrate the system to another blockchain network.

6.2 Storage of PROV Documents

The Systematic Literature Review revealed different strategies to store W3C PROV Documents on Blockchain. In the underlying use case, each workflow consists of multiple W3C PROV Documents, one for each component run, which will result in between 50 to 100 PROV Documents for one workflow on average. The underlying documents can reach enormous sizes with a high

number of included entities, activities, agents, and relationships. As stated in the SLR, keeping data off-chain and only storing its fingerprint on-chain is an efficient way to handle large amounts of data. In our use case, .rce files are stored locally on the engineer's systems. Engineers keep track of their own rce files and when they finish a workflow, which they potentially want to certify, they can upload the included provenance files to the system. The system shall keep a tamper-proof record of the upload to ensure the traceability of the development process. Since the certifier, who will verify the process later, will receive a copy of the original file, from which he can reproduce the workflow and export included provenance files, it's sufficient if the system stores a fingerprint. When a certifier wants to check if a workflow has been uploaded, the responsible person can upload the provenance records to the system, which calculates and compares its hashes. Keeping only the fingerprint on-chain also avoids exposing the actual provenance data to the public, which is in line with Requirement Q5. As a result, we decided to store hashes of data on the Blockchain. The second dimension revealed in the SLR is if we would store W3C-PROVs document or elementwise. We want to make use of the property, that each run of a component produces a own provenance entry and store the hash of a workflow together with each component run. This way we benefit from higher security and the ability to provide diffs. The higher security is achieved by making it necessary for attackers, to hack multiple hashes. In the case of SHA-256 it is believed to be almost impossible to modify a document in meaningful way to achieve the same hash. This risk is decreasing further, when attackers would have to crack between 50-100 hashes. Secondly, by storing the components runs to a workflow, in case of tamper the system can tell exactly which components have been tampered, providing more specific information about the abbreviation for certifiers. It is possible that two semanticly equal provenance documents produce different hashes. This is the case when documents contain special characters that have no effect on the actual PROV document, such as blank spaces or when PROV elements inside the document are listened in a different order. To avoid these effects on the hash, the PROV documents are canonicalized before hashing, as proposed in [62]. Since the input provenance data is in PROV-JSOn format, we'll use the JSON Canonicalization Scheme [84]. JCS creates a uniform JSON representation by serialising primitive JSON data types and sorting JSON object properties lexicographically, which allows to exchange JSON data "as is" [84].

6.3 Blockchain Technology

This section will discuss the selection of an appropriate type of Blockchain. After careful consideration, we have identified the following options: forming a private Blockchain network, creating a consortium Blockchain, joining a public network, or joining a consortium network. The upcoming discussion will explore the uses of each option based on the properties of Blockchain types that were identified in the theoretical background and captured in Table 1.

6.3.1 Discussion

Private Blockchains are widely utilized by businesses and organizations due to their high level of scalability and privacy, which surpasses that of public Blockchains. Typically, private Blockchains are used within a single organization. However, their centralized nature creates a single point of failure that increases the risk of tampering. As we are seeking to develop a solution that involves two organizations, consortium Blockchains become relevant. These types of Blockchains offer a balance between the scalability and privacy of private Blockchains, while also reducing the risk of a single point of failure.
Consortium Blockchains keep most advantages of private Blockchains, as high scalability and privacy and bring a higher level of decentralization, so that they avoid a single point of failure. The members of both parties could participate as nodes in the network. However, since both

parties have opposing interests and there is a lack of trust between them, there is a risk that one party gathering their power to attack the network. To reduce the risk of attacks, its important that none of the parties has the majority of power when it comes to achieving consensus. This can be quite difficult in a 2-party network. In case of Proof of Work, it would be necessary to limit the computational power of each party below 51%. That is only the case when both have exactly 50% computational power, which is hard to guarantee, especially with the chance of nodes going off- and online. Consortium's usually utilizes concensus mechanisms like PBFT or proof of authority. PBFT cant be a choice, because it fails when more then 1/3 of nodes are fraudulent. In a PoA network it would be comparable easy to make sure, that the validators are equaly distributed among the nodes in the network. This way it can be ensured, that none of the parties have the majority of power over the concensus mechanism and tamper the chain. However, it could still happen that both parties don't agree on a state, e.g. when one party blocks that valid transactions are added to the chain. Therefore we believe that its difficult to construct a secure consortium, including only two opposing parties. Hence we consider to join an existing network.
Public Blockchain are known for providing higher security with a much lower risk for data to be tampered. They eliminate the need for a third party, since the immutability and traceability of data can be trusted. On the other hand their low scalability and high transaction cost makes them a burden to use. scalability was no specific requirement of the stakeholders, however it can be a basic threshold feature. The high transaction cost and the volatility of transaction fees lead to a incalculably risk of exploding cost. There might be several public Blockchains, that are cheaper than the most famous ones, but believe they are more risky to disappear unexpectedly or more vulnerable to attacks.
The advantages of Consortium Blockchains were stated earlier in this section. Creating a consortium Blockchain was not favoured, because of the difficulty to achieve trust without a third party. However, joining an existing consortium Blockchain that distribute validation power among multiple independent institutions can provide a balance of power and lower the risk, that a malicious party takes over. Joining a consortium Blockchain can provide a significant amount of security combined with the scalability and low cost of a private Blockchain. That only trusted members can join the network and the level of decentralization lead to a lower risk of tamper and eliminate the need for us to include a 3rd party. At this point, we do not want to endorse any particular consortium Blockchain for participation. However, based on our the scientiftic focus of this use case we believe that the Bloxberg Blockchain would be a good match.

6.3.2 Bloxberg Blockchain

The bloxberg Blockchain [85] operates as a consortium with a focus on facilitating scientific collaboration, enabling research institutions to join and issue nodes to the organizations they collaborate with. Currently, the consortium comprises of 42 independent institutions, each contributing an authority node to validate transactions. We are of the opinion that the diversity of reputable institutions provides a significant level of trust, which in turn enables the potential of Blockchain technology to ensure immutability and traceability.
Offering a decentralized network to the global scientific community, the bloxberg infrastructure is the first of its kind to be maintained on a truly global scale. It delivers autonomous and robust services that transcend institutional boundaries, providing researchers with unprecedented access to a decentralized scientific network
Bloxberg was established by the Max Planck Society and ten founding research organizations in the February of 2019. It comprises organizations that possess in-depth knowledge of DLT in scientific research and recognize the potential of a decentralized global scientific infrastructure for the scientific community worldwide. The bloxberg infrastructure is composed of two primary components; the bloxberg technology, which includes nodes and smart contracts, and the governance model that outlines the network's rules. The infrastructure is built on top of a

permissioned Ethereum Blockchain. Ethereum is a good fit for our application, because of its stability, which it has proven under a high number of nodes on the mainnet and its ability to run smart contracts on the Ethereum Virtual Machine. Smart Contracts give us the flexibility to implement programmable code that fullfil our functional and quality requirements.
Bloxberg uses the Authority Round (AuRa) algorithm [86], which is based on Proof of Authority (PoA). Unlike Proof of Work (PoW), that require nodes to solve complex mathematical problems, a PoA consensus algorithm utilizes a set of "authorities" (in bloxberg called Authority nodes) that are explicitly permitted to create new blocks and secure the Blockchain. For a block to become a permanent part of the record, the majority of the authorities must sign it. To do so, first a leader is selected among the authority nodes. The leader is the only one that can issue a block. It is selected by the following formula:

$$Leader = (unixtime \ / \ block\ creation\ time)\ mod\ Validators \tag{6.1}$$

In Aura, two separate queues are maintained by the authorities; one for transactions (Q_t) and another for pending blocks (Q_b). The leader proposes a new block during each step of the consensus process, which includes a header containing the step number and the leader's signature of the block hash. The block is then broadcasted to the authorities for verification. Aura depends critically on the time synchronization of the validators. The Bloxberg infrastructure consists of Authority nodes and non-Authority nodes, with each member required to operate at least one Authority node. The Authority nodes perform computational activities and validate and store blocks and transactions of the distributed ledger, while the non-Authority nodes are responsible for communicating with the network. To join the Bloxberg network, an applicant must fill out a form on the official website, and the Iron Throne adds the applicant to the voting application. If the voting is successful, the new member is accepted as an authority node. Bloxberg's currency is named bergs, which are provided without cost, making deploying and using smart contracts entirely free.

6.3.3 Accessing the Network

Bloxberg allows to issue a limitless number of non-authoritive nodes. This nodes can participate in the network without having the responsibility to validate transactions. This has the advantage, that even when we the number of users from both institutions differ, it wont impact the power balance in the network. In case bloxberg will be utilized, we issue one node to the research center and one to the certification authority. Both institutions can, whenever they need access for employees, request additional nodes from the Bloxberg Consortium. Admins can register the new nodes to the system. With their private key, users can then interact with the network. The registration of new users and the interaction with the network is simplified by providing a graphical user interface, which is described in detail in section 7.2.

6.4 Smart Contracts

As a Ethereum-based Blockchain Network, Bloxberg supports the execution of smart-contracts in the Ethereum Virtual Machine. To enable users to interact with the Blockchain in the desired way and to satisfy the requirements, smart contracts will be employed in our concept. The structure of the smart contract is organized in a way, that align with the goals of our requirements. Since every component call can be considered a node in the workflow graph, we will stick to the terms node and workflow.
In section 6.2, we mentioned to store the hashes of provenance documents on-chain. Hence, the smart contract need to store workflow hashes and node hashes (component calls). Workflows and nodes could be implemented as follows:
A Workflow will consists of two fields:

- Hash of the workflow
- List of included node indices

A Node consists of its hash. We will obtain multiple workflows and nodes, hence we can store them in a list:

- `List[Workflow] workflows`
- `List[String] nodes`

The required functionalities of F1 - F3 can be implemented by the following methods:

- Methods to upload workflows and nodes
- Methods to check whether workflows and nodes are already stored on the chain

The simplest version to achieve the first functionality is by creating a `Upload_Workflow(string)` function, that takes the hash of a workflow as a parameter and adds it to the list of workflows. Uploading nodes can be approached with a similar logic. However, as it is required to upload workflows alongside its nodes and we wanna reference the nodes inside a workflow object, its more convenient to define one function: `add_workflow_and_nodes(string, List[string])`, that stores workflows and nodes at once.
Next the contract should provide the functionality to check, whether a workflow or a node has been uploaded or not. Checking workflows and nodes individually can be achieved straightforwardly by the methods:

- `check_Workflow(string) -> Bool`
- `check_Nodes(List[string]) -> List[Bool]`

To check for workflows, the first method takes a hash and returns true when the hash has been uploaded or false if not. Similarly, check_Nodes takes a list of hashes and returns a same-sized list of boolean values, that tells for each node, if it was uploaded. This way it is possible to check individually for a workflow and a list of nodes if they got uploaded to the chain. However, yet it is not possible to check if a list of nodes is completely representing a workflow and to which workflow they belong. Hence we add a third function:

```
check_WorkflowAndNodes(string, List[string]) -> Tuple(List[bool], uint)
```

This function will take a string representing the workflows hash and a list of strings representing the nodes hashes as parameter. It returns a list and an integer. The list has the same length as the passed nodes and indicates for each, if its uploaded and assigned to the passed workflow. Hence it tells the user, if the nodes have been uploaded together with the workflow. The unsigned integer indicates the number of missing nodes, i.e. nodes that belong to the workflow, but haven't been passed. Overall the function reveals, if the passed nodes belong to the workflow and whether the passed nodes cover all nodes of the workflow.
The methods to add PROV Documents and check if they exist expect hashed documents as parameters. To this point we left the question out, how those hashes are obtained from PROV documents. This task will be fulfilled by the GUI, which is going to be explained in the next chapter.

6.5 Graphical User Interface

After conceptualizing basic attributes and functions one question to ask is, how users may interact with the smart contract. To keep the user experience straightforward and simple, a

GUI-client will be provided. The client extends the functionalities of the smart contract and helps the user to interact with the system without the need to write code. Additionally the GUI performs some pre-checks before data is uploaded. While Blockchain can deliver tamper-resistant and traceable storage, it cant determine whether uploaded data is valid. Therefore, data input is prone to errors and can lead to the permanent storage of erroneous data. By checking input operations before uplods are performed, we hope to improve the quality of stored data.
The GUI provides different panels. One panel is for the Upload of Provenance Documents. This panel allows users of the research institute to select PROV-JSON from workflows and nodes and upload them to the system. We restrict the selection to.json files, because thats the export-format of the RCE-Plugin and expect the filenames to match the original Plugin export name format. After selecting documents the interface performs a pre-check. During the pre-check the system verifies, if the number of selected nodes matches the number of nodes specified in the workflow document. This information can be achieved by counting the number of hadMembers inside the workflows PROV Document. We distinguish workflows and nodes according to their filenames. Only if the number matches, the user can click a upload button. After clicking the upload button, the user is asked for his private key. The key wont be stored, to avoid the client from storing sensitive information. After the user entered his key, the documents get converted to PROV-XML, cancollized and hashed. The hash will then be broadcasted to the Blockchain network, by building a transaction that pass the hashes to the upload_workflow function in the Smart Contract.
Another panel allows to check if a document has been uploaded, hence establishing a connection between the GUI and the `check_nodes`, `check_workflow` and `check_NodesAndWorkflow` functions of the Smart Contract. The User will have the option to select workflow and node files and can check them seperately by clicking either the check workflow or check node button. Both of the buttons will cancolize and hash the documents before calling the smart contracts, to ensure that the processing procedure is the same as at the Upload Phase. The Check Workflow Button will call the `check_workflow` method and display the result to the user. If the check returns true, i.e. if the workflow was stored on the Blockchain, the system will additionally provide address of the uploading user and the transaction time. The functionality of the check node button, depends on whether a workflow has been selected or not. When the check node button is clicked and no workflow has been selected, the nodes are checked by making a call to the `check_nodes` function. When the result is ready, a list of uploaded nodes is displayed alongside with the information, if the node was found on the chain or not. The user can press a details button, to show the upload time and address of the uploading user. If workflows and nodes has been selected, a click on the check node button calls the `check_NodesAndWorkflow` of the Smart Contract. This time, additionally to the information if the node has been uploaded, the user recieves the information if a node is included in the workflow. Again the details button provides uploader and upload time information for each entry.
The third Panel is the Registration Panel. The Registration Panel allows admins to add new users and admins to the system. To add them they specify the private key of the candidate and press the "add user" or "add admin" button. To check whether the person who try to register new users has the permission and is an admin, the user has to type in their private key. The added users and admins are institution specific. Hence admins of the DLR register DLR users and admins; CA admins register CA users and admins. The different roles and the rights associated with them are discussed in more detail in the next section.

6.6 Permission Management

As a last component we enrich our system by a role-based access control system to provide institutions and users with varying levels of access based on their responsibilities. The control system will be directly integrated into the smart contract. This way user roles can be identified

by their address in the network and we avoid adding another authentification layer. We define four roles: Creater, CreaterAdmin, Viewer and ViewerAdmin, where the first two are issued to members of the DLR and the last two to members of the Certification Authority. Creaters are provided with permission to write. Members of the DLR shall be able to upload their workflow provenance exports to the documentation system and hence need the right to write data to the contract. To implement this, Creaters have an exclusive right to the `add_workflows` function and can call the check and get functions as well. CreaterAdmins inherits the Creater rights and can additionally grant the Creater and CreaterAdmin role. CreaterAdmin and ViewerAdmins are the roles, which are issued to the institutions initially, which allow them to manage and register additional users to the system. Viewers and ViewerAdmins belong to the Certification Authority, which needs to check if documents have been uploaded, but shouldn't be able to add elements to the contract. Hence they can call the check and get functions. The ViewerAdmin works like the CreaterAdmin and allows the registration of new Viewers and ViewerAdmins.
We are aware of the fact, that data stored on the Blockchain can be accessed by public. Hence the access control cant prevent data on the chain to be viewed. So it might be argued, that the access control is useless for the function that don't modify the contracts state, like the check and get functions. Since we do not store any sensitive information on the chain, it would not be harmful at all if someone else were to disclose data. However, we believe that by restricting access to the smart contract functions, we are making it more difficult to interact with the contract.

7 Software Prototype

In this chapter, we will propose a software prototype that was developed to test the feasibility of our concept. The prototype was designed and implemented with the goal of demonstrating the architecture, key features, and functionality of the proposed solution. It consists of the implementation of a Smart Contract in section 7.1 and a Graphical User Interface in section 7.2. Both implementations are explained by providing pieces of code. However, the full implementations can be accessed in Listing 2 and 3 of the Appendix.

7.1 Smart Contract

Access Control

To control the access to our smart contract we used OpenZeppelins `AccessControl` [87]. Compared to common alternatives like `Ownable` [88] and `Whitelist` [89], `AccessControl` is the only contract that is capable of handling multiple roles and owners at a time. AccessControl is an abstract class, that allows to grant and revoke roles flexibly. As the class-diagram in Figure 16 shows, `AccessControl.sol` inherits from `IAccessControl`, `ERC165` and `Context.sol`. `IAccessControl` is an Interface, that defines method signatures and events for setting, revoking and getting roles. `ERC165.sol` and `Context.sol` are abstract classes. The first one provides an enhanced way of inter-contract communication, while the latter makes messages sender and data accessible.
`AccessControl.sol` stores the current state of roles as a mapping, which is a datatype storing key-value pairs, similar to a dictionary or a hashmap in Java and Python. Via the internal `_grantRole` and `_revokeRole` methods, roles can set and revoked flexibly. The `onlyRole` modifier can be added to functions, to restrict its execution to certain roles. Roles can be grant and envoked by the defaultAdmin, which is initially set to the address who deploys the contract.

```
6  contract Prov is AccessControl {
7      bytes32 private constant CREATER_ROLE = keccak256("CREATER_ROLE");
8      bytes32 private constant VIEWER_ROLE = keccak256("VIEWER_ROLE");
9      bytes32 private constant CREATER_ADMIN_ROLE =
10         keccak256("CREATER_ADMIN_ROLE");
11     bytes32 private constant VIEWER_ADMIN_ROLE = keccak256("VIEWER_ADMIN_ROLE");
```

Listing 7.1: Access Control related Attributes of the Contract

The Prov Contract inherits from `AccessControl.sol`. As shown in Line 7 - 11 of Listing 7.1, Prov contract defines four roles: Creater, Viewer, CreaterAdmin and ViewerAdmin. As mentioned in section 6.6 of the concept, Creater roles correspond to the research institue and viewer roles to the certification authority. Roles are identified by a 32 bytes identifier, which is generated by applying the keccak256 hash function to the role name. Roles can be granted by calling the respective grant functions, defined in lines 29 - 53 of Listing 7.2. Each grant function is restricted by an onlyRole modifier, which restricts the access to the function to a specific role. The grantViewer and grantViewerAdmin functions are restricted to ViewerAdmins; grantCreater and grantCreaterAdmin to CreaterAdmins. When a Account is granted ViewerAdmin or CreaterAdmin rights, they receive the Viewer (Line 33) or Creater privillege (Line 40) aswell.
Initially one ViewerAdmin and one CreaterAdmin are assigned. This is realized by passing the respective account addresses to the constructor defined in line 22. Storing documents, realized by

the `add_workflow_and_nodes` method in 7.6 is restricted to the role Creater. All other functions are restricted to members of the four roles, which is checked by the `hasAccess` method defined in line 55.

```
22      constructor(address creater_admin, address viewer_admin) AccessControl() {
23          _grantRole(CREATER_ADMIN_ROLE, creater_admin);
24          _grantRole(CREATER_ROLE, creater_admin);
25          _grantRole(VIEWER_ADMIN_ROLE, viewer_admin);
26          _grantRole(VIEWER_ROLE, viewer_admin);
27      }
28
29      function grantViewerAdminRole(
30          address account
31      ) public onlyRole(VIEWER_ADMIN_ROLE) {
32          _grantRole(VIEWER_ADMIN_ROLE, account);
33          _grantRole(VIEWER_ROLE, account);
34      }
35
36      function grantCreaterAdminRole(
37          address account
38      ) public onlyRole(CREATER_ADMIN_ROLE) {
39          _grantRole(CREATER_ADMIN_ROLE, account);
40          _grantRole(CREATER_ROLE, account);
41      }
42
43      function grantViewerRole(
44          address account
45      ) public onlyRole(VIEWER_ADMIN_ROLE) {
46          _grantRole(VIEWER_ROLE, account);
47      }
48
49      function grantCreaterRole(
50          address account
51      ) public onlyRole(CREATER_ADMIN_ROLE) {
52          _grantRole(CREATER_ROLE, account);
53      }
54
55      function hasAccess() internal view returns (bool) {
56          return (hasRole(CREATER_ADMIN_ROLE, msg.sender) ||
57              hasRole(CREATER_ROLE, msg.sender) ||
58              hasRole(VIEWER_ROLE, msg.sender) ||
59              hasRole(VIEWER_ADMIN_ROLE, msg.sender));
60      }
```

Listing 7.2: Methods for granting Roles

Storing Data

Workflows and nodes are stored as attributes of the Contract as shown in Listing 7.3. Nodes are encoded as a list of strings, where each string stores the hash of a node. The workflow struct consists of a workflows hash and a list of nodes. The list of nodes holds the index of each node, that belongs to the workflow. By storing the index, the respective nodes can be accessed efficiently without the need of additional identifiers.

```
13      struct Workflow {
14          string hash;
15          uint256[] nodes;
16      }
17
18      //arrays for list of workflows and nodes
19      Workflow[] private workflows;
20      string[] private nodes;
```

Listing 7.3: Definition of Workflow and Node contract attributes

The `add_workflow_and_nodes` method in 7.6 provides the functionality to store provenance documents. The parameters are a string including the workflows hash and a list of strings containing hashes of associated nodes. The onlyRole modifier in Line 147 restricts the access to the function to Creaters. The logic is straightforward. The function iterates over the list of nodes hashes (Line 151). Each nodes hash is pushed to the contracts list of nodes (Line 152) and its index is stored in a temporal `node_list` (Line 153). After iterating over all nodes, a workflow object, containing the hashed workflow and the node list, is pushed to the contracts list of workflows.

```
151     add_workflow_and_nodes(
152         string memory hashed_workflow,
153         string[] memory hashed_nodes
154     ) public onlyRole(CREATER_ROLE) {
155         uint256 node_id = nodes.length;
156         uint256[] memory node_list = new uint256[](hashed_nodes.length);
157 
158         for (uint i = 0; i < hashed_nodes.length; i++) {
159             nodes.push(hashed_nodes[i]);
160             node_list[i] = node_id;
161             node_id++;
162         }
163         workflows.push(Workflow(hashed_workflow, node_list));
164     }
```

Listing 7.4: Definition of `add_workflow_and_nodes` function

Checking for PROV-Documents

The second desired functionality is to check whether a document has been uploaded to the blockchain. For this purpose, Listing 7.5 defines check_workflow (Line 72) and check_nodes (Line 85), which examines whether a workflow or a list of nodes is stored on the chain. Both methods are view methods, hence they don't modify the contracts state. The check_workflow function expects a hashed workflow document as a string and returns a boolean. The method checks iteratively in lines 74 - 81, whether the string is stored in the contracts workflow array. Strings in solidity are like a bytes array, so comparing them directly would iterate over the arrays. Therefore we use abi.encodePacket to transform them into comparable raw bytes before. If the string is found, true is returned, otherwise false.
The check_nodes in Line 85 method expects a list of hashed nodes as an array of strings. In contrast to check_workflows, the check_nodes functions returns a list of booleans. Each boolean entry indicates if a node was stored on the chain or not. Therefore line 89 initialize result, which is an empty boolean array with the same length as the parameters node list. Between line 90 - 100 a nested loop is performed, that checks for each node of the nodes list, if its stored on the blockchain. If a nodes hash was found on the chain, its respective entry in the result list is true, else its false.

```
72     function check_workflow(string memory prov) public view returns (bool) {
73         require(hasAccess(), "Access Denied");
74         for (uint i = 0; i < workflows.length; i++) {
75             if (
76                 keccak256(abi.encodePacked(workflows[i].hash)) ==
77                 keccak256(abi.encodePacked(prov))
78             ) {
79                 return true;
80             }
81         }
82         return false;
83     }
84 
85     function check_nodes(
```

```
86          string[] memory node_list
87      ) public view returns (bool[] memory) {
88          require(hasAccess(), "Access Denied");
89          bool[] memory result = new bool[](node_list.length);
90          for (uint i = 0; i < node_list.length; i++) {
91              result[i] = false;
92              for (uint j = 0; j < nodes.length; j++) {
93                  if (
94                      keccak256(abi.encodePacked(nodes[j])) ==
95                      keccak256(abi.encodePacked(node_list[i]))
96                  ) {
97                      result[i] = true;
98                      break;
99                  }
100             }
101         }
102         return result;
103 }
```

Listing 7.5: Definition of `check_workflow` and `check_nodes` functions

But how to check nodes and workflows at the same time? This purpose is fulfilled by the check_workflow_and_nodes function in Listing 7.6. Unsurprisingly its a view function, that don't modify the state of the contract. The method takes the hashes of a workflow and of a list of nodes as input. It returns a boolean array and an uint256. The boolean array indicates for each node if it's assigned to the workflow, i.e. if it's stored in the workflows node list. The uint256 is the difference between actual and provided nodes of the workflow, telling how many nodes the workflow contains, that haven't been upload. To say it straight the method provides information about, which nodes are included in the workflow and how many nodes are missing.
To achieve this, in line 109 we check if the workflow is stored on the blockchain. Instead of check_workflow, which was defined earlier, we use the private helper function find_workflow, because we'll gonna need the index of the workflow later. If the workflow isnt stored on the blockchain, the method returns a list containing false for all nodes and 0 (Line 115). Otherwise we proceed with checking, which nodes are included in the workflow. For that line 120 extracts the nodes of the workflow. Line 121 initialize found_nodes, that counts how many nodes were found, allowing us to calculate the missing ones later. Between Line 122 - 133 a nested loop checks for each passed node, if its contained in the workflows nodes list. If so, its entry in the node_check_result array is set to true and the found_node counter is incremented (Line 129 131). The last line returns the result array and the number of missing elements.

```
105     function check_workflow_and_nodes(
106         string memory hashed_workflow,
107         string[] memory hashed_nodes
108     ) public view returns (bool[] memory, uint256) {
109         require(hasAccess(), "Access Denied");
110         int index = find_workflow(hashed_workflow);
111         bool[] memory node_check_result = new bool[](hashed_nodes.length);
112 
113         // workflow not existing
114         if (index == -1) {
115             return (node_check_result, wf_nodes.length);
116         }
117 
118         uint u_index = uint256(index);
119 
120         uint256[] memory wf_nodes = workflows[u_index].nodes;
121         uint256 found_nodes = 0;
122         for (uint i = 0; i < hashed_nodes.length; i++) {
123             for (uint j = 0; j < wf_nodes.length; j++) {
124                 if (
125                     keccak256(abi.encodePacked(hashed_nodes[i])) ==
126                     keccak256(abi.encodePacked(nodes[wf_nodes[j]]))
127                 ) {
128                     node_check_result[i] = true;
129                     found_nodes++;
```

```
130                     break;
131                 }
132             }
133         }
134 
135         return (node_check_result, wf_nodes.length - found_nodes);
136     }
137 
138     function find_workflow(string memory prov) private view returns (int) {
139         require(hasAccess(), "Access Denied");
140         for (uint i = 0; i < workflows.length; i++) {
141             if (
142                 keccak256(abi.encodePacked(workflows[i].hash)) ==
143                 keccak256(abi.encodePacked(prov))
144             ) {
145                 return int(i);
146             }
147         }
148         return -1;
149     }
```

Listing 7.6: Definition of `check_workflow_and_nodes` function

Interacting with the contract

To illustrate how we the interaction with the contract works, Listing 4 give an example. We will skip the deployment of the contract, because its covered in detail during the Test and Analysis in section 8.1. For now, its sufficient to know, that the smart contract has been deployed and is stored in a variable called `deployed_prov_contract`. The variables dlr_admin_adress and viewer_admin_address contains accounts, that have been initially assigned as createrAdmin and viewerAdmin, respectively.
The first line shows a call to the get_nodes function, that results in an "Access Denied" Error. All function of the contract are permissioned to the roles, we defined earlier. Hence we need to call the function from one of those. Line 4 repeats the call, passing the dlr admin address to "from" entry of the transaction dictionary. This address got assigned as CreaterAdmin and hence is authorized to call that function. Therefore Line 4 returns `[]`, which is the expected output, since the nodes list was initialized empty. Line 7 calls the `add_workflow_and_nodes` function with random strings as workflow and nodes. When calling the get_nodes function in line 10, the output is still an empty list. The workflows and nodes haven't been added, because its not possible to tamper the state of contract without performing a transaction. A transaction have to be signed by a private key, which makes it more secure compared to a function call. While it might be possible to obtain someones public key to call a function without authorization, to modify the state the private key is necessary. Line 13 - 20 includes the function call into a new transaction. After singning, sending and recieving the transaction recipe, Line 23 calls get_nodes again. This time, the output in line 24 shows the nodes passed to the add function.

```
1  deployed_prov_contract.functions.get_nodes().call()
2  # Access Denied error
3  
4  deployed_prov_contract.functions.get_nodes().call({'from': dlr_admin_adress})
5  # outputs: []
6  
7  deployed_prov_contract.functions.add_workflow_and_nodes('5', ['6','7','8'])
8  .call({'from': dlr_admin_adress})
9  
10 deployed_prov_contract.functions.get_nodes().call({'from': dlr_admin_adress})
11 # outputs: []
12 
13 transaction = deployed_prov_contract.functions.add_workflow_and_nodes('5', ['6','7', '8↩
       '])
14 .buildTransaction({"chainId": chain_id,
15                     "from": dlr_admin_adress,
```

```
16                      "gasPrice": w3.eth.gas_price,
17                      "nonce": dlr_nonce})
18 sign_store_contact = w3.eth.account.sign_transaction(
19     transaction, private_key=dlr_admin_private_key
20 )
21 send_store_contact = w3.eth.send_raw_transaction(sign_store_contact.rawTransaction)
22 transaction_receipt = w3.eth.wait_for_transaction_receipt(send_store_contact)
23 deployed_prov_contract.functions.get_nodes().call({'from': dlr_admin_adress})
24 # outputs: ['6', '7', '8']
```

Listing 7.7: Interaction with the Contract

7.2 Graphical User Interface

We provide a GUI to simplify the interaction of users with the blockchain. The GUI consists of an Upload Page, where users can upload documents to the chain, a Verfication Page, where users can check if a document has been uploaded and an Admin Page, where Admins can grant User or Admin rights to other accounts. The prototype includes a complete GUI for the upload page that can already functionally communicate with the blockchain. The feasibility was assessed through the development of the Upload Page, while the verification and admin pages may be implemented at a later time

Upload Page

Figure 11 shows a prototype of the Upload Page. The initial View after opening the Upload Page is displayed in Figure 11a. The Check Files button is gray until the user selected files to upload. The user can select files by clicking the white folder button right above the Check Files button. Once a file was selected, it is added to the file list if it is a .json file and if its not already in the list. This logic is handled by the select_files method defined between lines 144 - 176 of `UploadPage.py` in Appendix D. The file list can be cleared by the remove button in the top left corner. Figure 11b shows the Interface after selecting documents and clicking the check files button. We selected one workflow, that consists of seventeen components, but only selected fifteen of those. Hence the check fails and opens a popup that directs the user about the missing documents. Clicking the button calls the check_files method defined in lines 183 - 231 of Appendix D. This function checks whether:

- exactly one workflow is provided
- number of provided nodes matches number of nodes in the workflow
- files are named like a workflow or node exported by the rce2prov plugin.

After providing all 17 nodes together with the workflow, the check displays a popup including a success message, as shown in Figure 11c. After a positive check, the check Files button is substituted by an upload button. Whenever the file list is changed, the user is required to check the files again, before the interface allows to upload files. By pressing upload the checked files can be uploaded to the chain. As shown in Figure 11d, therefore the user is required to insert his private key. If the private key is valid and permissioned to upload the file, the documents get hashed and uploaded to the chain. This is handled by the upload_files function in Listing 7.8. After recieving the private key (line 2) and initalizing variables for workflow and nodes (line 3), it simply iterates over the files, which are stored inside the itemlist (line 4). Since itemlist only store the ids, and not the paths of the files, line 5 obtains the paths. Line 6 opens a file reader and pass it to the jcs.canonicalize method in line 7. This method calls the JSON Canonicalization Scheme [84], that performs three tasks:

- Serialization of primitive JSON data types

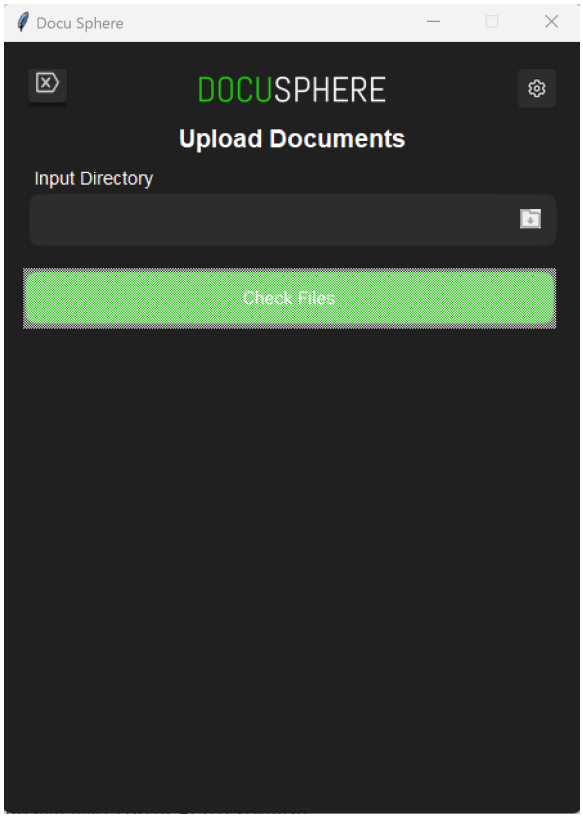
- Lexicographic sorting of JSON Object properties
- JSON Array data is also subject to canonicalization [84].

After obtain its canonicalized version, line 8 calculate the documents hash by using the SHA256-method. Depending on whether its a workflow or a node, the hash is stored inside the workflow variable or appended to the node list. After iterating over all files, line 14 makes a call to Web3Provider.upload_nodes_and_workflows function. This function is a class-method defined in Webprovider.py in Appendix D. It basically creates a transaction including a call to the add_workflow_and_nodes function, as we seen earlier in Listing 4. To upload workflows and nodes to the blockchain, line 14 passes the private_key, the workflow and nodes to the method. Depending on whether the transaction was accepted by the blockchain, the user either receive a success-message (line 18) or an error (line 16). In any case, the file list is cleared afterwards (line 20).

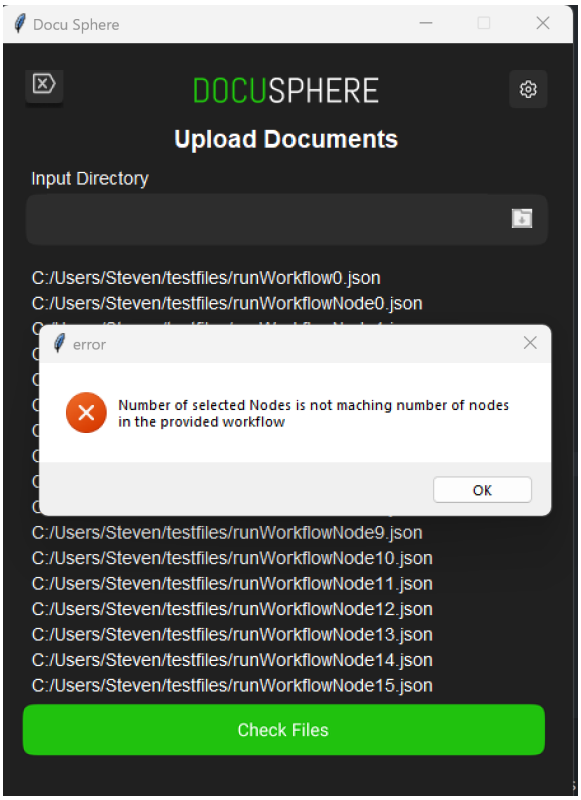
```
1     def upload_files(self):
2         private_key = simpledialog.askstring(title="Key", prompt="Insert Private Key")
3         workflow, nodes = "", []
4         for item_id in self.itemlist:
5             path = self.canvas.itemcget(item_id, "text")
6             file_reader = open(path)
7             json_file = jcs.canonicalize(json.load(file_reader), utf8=False)
8             hash = sha256(json_file.encode("utf-8")).hexdigest()
9             if "runWorkflow" in path and "Node" not in path:
10                workflow = hash
11            else:
12                nodes.append(hash)
13        try:
14            Web3Provider.upload_nodes_and_workflows(private_key, workflow, nodes)
15        except Exception as e:
16            showerror("Error!", e)
17        else:
18            showinfo("Upload successful!", "Done! Documents uploaded succesfully.")
19
20        self.clear_files()
```

Listing 7.8: Definition of `upload_files` method

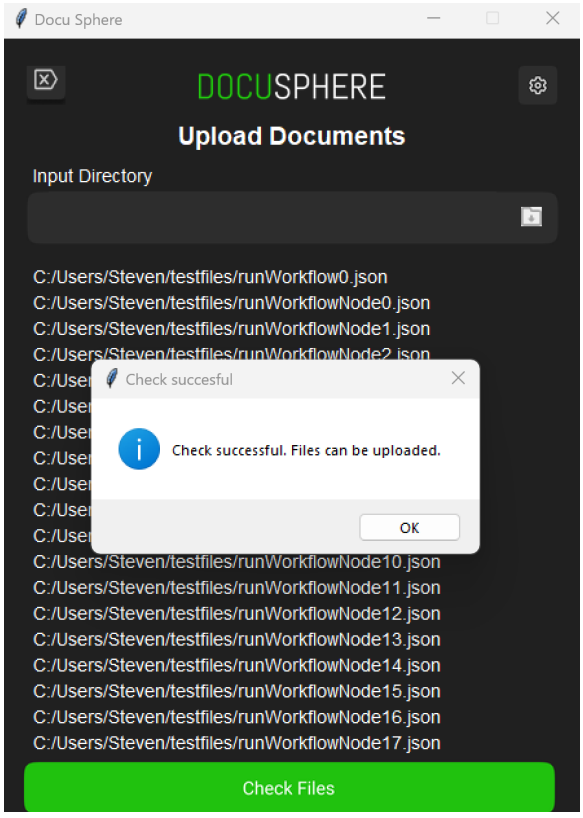
Figure 11: Upload Page of Graphical User Interface (GUI)



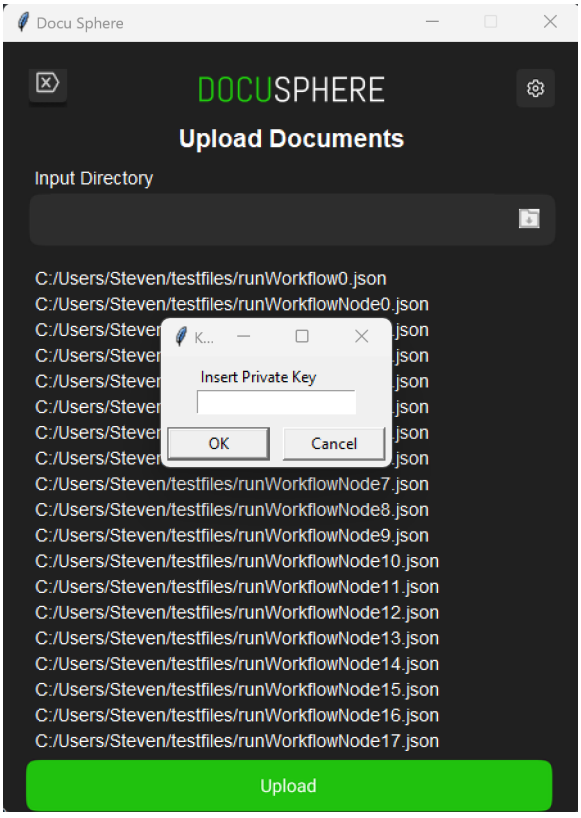
(a) Initial Upload Page



(b) Failed Check Case



(c) Successful Check Case



(d) Upload View

8 Tests and Analysis

8.1 Environment

Ganache

We use Ganache to set up a Blockchain Testbed. Using Ganache we can build an Ethereum Blockchain to develop, deploy and test our smart contract in a local and safe environment. Ganache brings an UI that simplifies setting up the blockchain and tracking its state. After creating a workspace, the User Interface looks as shown in Figure 12. The header allows switching between six pages: Accounts, Blocks, Transactions, Contracts, Events, and Logs. Initially, the account page is active. It shows a list of accounts, containing their public address, balance, transaction count, and index. Each account is initialized with 100 ether. The Block page shows mined blocks and allows checking each header, contained transactions, and gas. The other pages display executed transactions, deployed contracts, triggered events, and server logs.

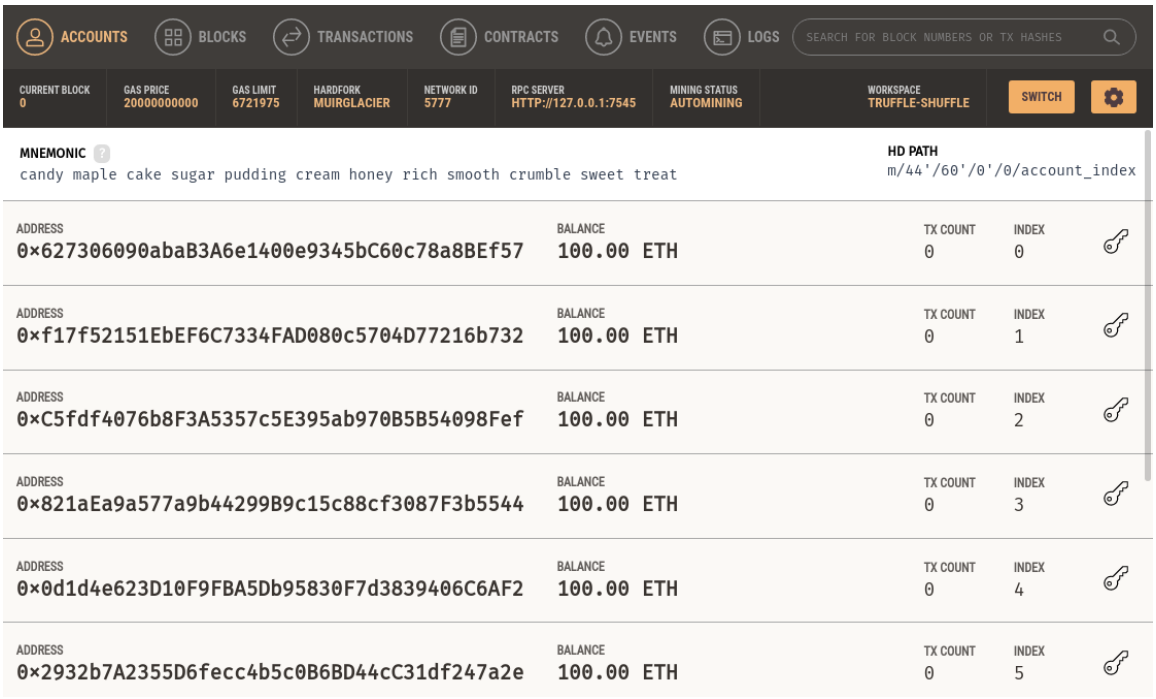


Figure 12: Ganache User Interface. Source: [90]

To execute the tests in an independent environment and to provide all steps from the beginning, we use Ganache to initialize a local blockchain including 6 accounts. Table 7 shows a list of the account addresses and the roles we intend to set them to. Address 1 will be used to deploy the smart contract to the blockchain. Accounts 2 and 3 are assigned to CreaterAdmin and ViewerAdmin roles and are intended to be issued to the research institute and the certification authority, respectively. Accounts 4 - 6 are not assigned any role and will be used to test the assignment of roles later. The contract address in the last line is obtained after deploying the smart contract, which is explained later in this section.

Account_ID	Address	Role
1	0x1a3f71Eef963bC2Be646feb318bFEd5a07993016	-
2	0x0Ad802eF9c327d7c6A742b118Ced23DF20eDCbFf	CreaterAdmin, Creater
3	0xf075bbAED18B752d076f6D16f75a9c9CBcCb0b25	ViewerAdmin, Viewer
4	0x6A3Fd877d4688bA35dCc7D134956C1D2075133e1	-
5	0xf4A68d730B37925e04efe3D64cFcfDa7ae3A60b3	-
6	0x17035CEC72f39c1f8694EBdF2307769637BE536e	-
-	0x9835781C0f5dc776F0EB439fF9EfCB999662B935	Contract

Table 7: Ganache Test Environment Accounts

Brownie

Before a contract can be deployed to the blockchain network, it has to be compiled. Ethereum relies on the Ethereum Virtual Machine (EVM), which executes bytecode. Therefore, Smart contract code written in high-level languages must be compiled into EVM bytecode before it can be executed. We use Brownie to compile the Smart Contract. Brownie is a Python-based development and package managing tool for smart contracts. Without Brownie, it can be challenging to import external libraries into a Solidity Smart Contract. Brownie simplifies installing and importing packages into Solidity Smart Contracts, by providing a Package Manager. Additionally, Brownie supports the organization and deployment of projects. We use Brownie to keep a clean, reproducible environment. To do so, we started with creating an empty brownie environment in an empty directory using `brownie init`. Brownie created multiple folders, including a contract directory, which we pasted the contract file `prov.sol` in. Then we installed the required package: `brownie pm install OpenZeppelin/openzeppelin-contracts@4.8.2` installed the current version of Openzeppelins open access library. In the environment's main directory, we inserted the default configuration file [91] and made some modifications. We set the compiler version to 0.8.0 and under dependencies we added `-OpenZeppelin/openzeppelin-contracts@4.8.2`. The `prov.sol` contract was compiled by running `brownie compile`, which creates a prov.json in the environments build/contracts subdirectory.

Deployment of Smart Contract

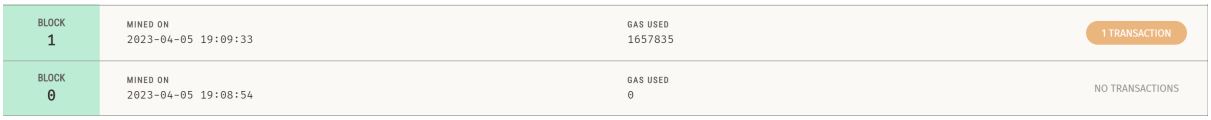
To deploy the contract on the Blockchain we use the Web3 Library. Web3 [92] is a Python library for interacting with Ethereum. We installed the Web3 package inside an Anaconda Environment, running with Python version 3.7. Listing 8.1 shows the Python script we run to deploy the Prov Contract to the blockchain. Before deploying the contract, line 4 read in the contracts build created earlier by Brownie. Line 8 and 10 extracts its abi- and bytecode and store it inside separate variables. Line 12 establishes the connection to the local blockchain we created. In Line 14 we assign the Chain ID, which is necessary to build a transaction later. In Line 17, we use the contracts abi- and bytecode to create a w3.eth.contract object. Line 20 specifies the account, that will send the transaction to deploy the contract, by its address. As nonce, line 21 uses the transaction count of that account. When deploying the smart contract, the constructor of the Prov contract is called. The constructor expects two addresses to initialize an admin of the research institute and certification authority. We take the account addresses 2 and 3 from Ganache and assign them as admin addresses in lines 23 and 24. Then we create a transaction object between lines 27 and 34. For that, line 27 calls the constructor of the `prov_contract` object and pass the two admin addresses to it. The constructor call is packed inside a new transaction, that is built in line 28. We pass the chain_id, the gas price, the sender address, and the nonce to the `buildTransaction` method. Line 38 signs the transaction using account 1's private key and line 40 sends the transaction as bytecodes to the blockchain. After the transaction is finished, we can obtain a transaction receipt. The transaction receipt includes the

transaction hash and the transaction address, which we'll need to interact with the contract. Line 46 creates a contract object from the contract address and the abi code. The contract object can be used to interact with the contract, i.e. call its functions.

```
1  import json
2  from web3 import Web3
3
4  with open("ProvProject/build/contracts/Prov.json", "r") as file:
5      compiled_sol = json.load(file)
6
7  # get bytecode
8  bytecode = compiled_sol["bytecode"]
9  # get abi
10 abi = compiled_sol["abi"]
11
12 # For connecting to ganache
13 w3 = Web3(Web3.HTTPProvider("http://127.0.0.1:7545", request_kwargs={'timeout': 600}))
14 chain_id = 1337
15
16 # Create the contract in Python
17 prov_contract = w3.eth.contract(abi=abi, bytecode=bytecode)
18
19 # Get the number of transactions
20 deyployer_address = "0x1a3f71Eef963bC2Be646feb318bFEd5a07993016"
21 deployer_private_key = ""
22 nonce = w3.eth.getTransactionCount(deyploying_address)
23
24 dlr_admin_adress = "0x0Ad802eF9c327d7c6A742b118Ced23DF20eDCbFf"
25 ca_admin_adress = "0xf075bbAED18B752d076f6D16f75a9c9CBcCb0b25"
26
27 # build transaction
28 transaction = prov_contract.constructor(dlr_admin_adress,ca_admin_adress)
29     .buildTransaction(
30     {
31         "chainId": chain_id,
32         "gasPrice": w3.eth.gas_price,
33         "from": deyployer_address,
34         "nonce": nonce,
35     }
36 )
37 # Sign the transaction
38 sign_transaction = w3.eth.account.sign_transaction(transaction, private_key=↩
       deployer_private_key)
39 # Send the transaction
40 transaction_hash = w3.eth.send_raw_transaction(sign_transaction.rawTransaction)
41 # Wait for the transaction to be mined, and get the transaction receipt
42 transaction_receipt = w3.eth.wait_for_transaction_receipt(transaction_hash)
43 print(f"Done! Contract deployed to {transaction_receipt.contractAddress}")
44 # Done! Contract deployed to 0x9835781C0f5dc776F0EB439fF9EfCB999662B935
45
46 deployed_prov_contract = w3.eth.contract(address=transaction_receipt.contractAddress, ↩
       abi=abi)
```

Listing 8.1: Deployment of Smart Contract

Figure 13 shows the block list after deploying the contract. In addition to the genesis block, a second block has been added to the blockchain. The block contains one transaction of type `CONTRACT CREATION`. The from address corresponds to the deploying address in Table 7. Instead of "To", a contract creation contains a "Created Contract Address" field, indicating the address to which the contract has been deployed. We'll use this blockchain state to test the functional and quality requirements in the following sections.



(a) List of Blocks

(b) Block 1: Header and Body

Figure 13: Blockchain after Smart Contract Deployment

8.2 Functional Requirements

F1

As F1 stated, the developed system shall allow engineers to upload PROV-JSON documents of workflows and nodes. To show that the requirement is satisfied, we'll upload a workflow consisting of eighteen nodes using the proposed Upload Page and account 2, which holds the proposed Creater role for engineers. The workflow file is named *runWorkflow0* and can be accessed together with the node files from *runWorkflowNode0* to *runWorkflowNode17* in the *Workflow1* directory accessible under: [93]. The uploading process went as covered earlier in Figure 11. A popup message displaying *Upload successful* indicated, when the process was finished.
Using Ganache we examined the third block, that has been added to the blockchain. Figure 14 shows the content of the block. The transaction in the block body is a contract call. The from address belongs to Account 2, which was used to upload the document.

F2

The second requirement is, that the system shall be able to keep record of which documents have been uploaded. To demonstrate that the system is capable of storing multiple workflows, we added another workflow containing two nodes, which can be accessed in the *Workflow2* folder under: [93]. The script is responsible for loading, canonicalizing, and hashing the documents is shown in Listing 5 of Appendix E. The `deployed_prov_contract` variable storing the smart contract has been defined in Listing 8.1.
To show that the system keeps record of uploaded documents, we call the `get_nodes` and `get_workflows` functions in Listing 8.2. The output of `get_workflow` starting from line 3 shows, that both workflows are stored in the contracts workflow variable. Each workflow contains its hash and a list of included nodes. The `get_nodes` function contains 20 elements, as line 9 indicates. Each of them contains a hash, which is shown for the first element in line 8.

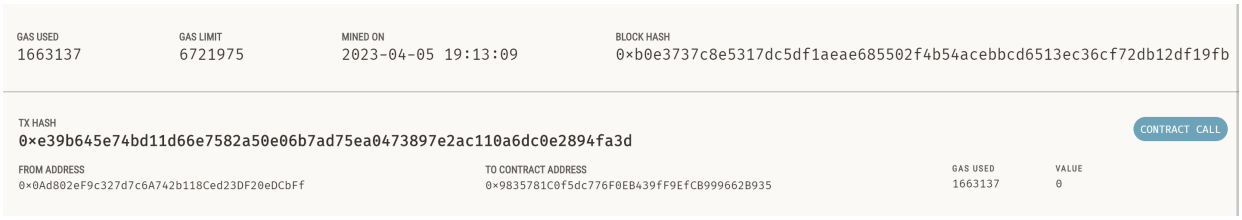


Figure 14: Header and Body of Block 2

ID	Description	Line
1	`check_workflow` with a stored workflow	1
2	`check_workflow` with a random string	4
3	`check_nodes` with stored nodes and random string	7
4	`check_workflow_and_nodes` with stored workflow and all nodes belonging to the workflow	10
5	`check_workflow_and_nodes` with stored workflow, all nodes belonging to the workflow and random string	13
6	`check_workflow_and_nodes` with stored workflow, a node belonging to the workflow and random string	16
7	`check_workflow_and_nodes` with stored workflow and nodes belonging to another workflow	20

Table 8: Evaluation Scenarios for Check Functions

```
1  deployed_prov_contract.functions.get_workflows().call({"from": ca_admin_adress})
2  """
3  outputs: [('fda4b58bb405600041522c87aea1333ee72d8840c3062ccfd84acecc892f004f',
4  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17]),
5  ('5e566c2f44ba7db8e384f1d269244874ea51ad1d1ead59519f0b35dda9595d02',
6  [18, 19])]
7  """
8  len(deployed_prov_contract.functions.get_nodes().call({"from": ca_admin_adress}))
9  # outputs: 20
10
11 deployed_prov_contract.functions.get_nodes().call({"from": ca_admin_adress})[0]
12 # outputs: ac254608fce6471abfd63e2719abd72ae0088f3f30b74c941b8e76d62836a572
```

Listing 8.2: Obtaining stored Workflows and Nodes

F3

The third requirement is, that the system shall provide certifiers with the ability to check, whether a workflow or node has been uploaded. The Methods to achieve this functionality have been proposed in section 7.1. We provide individual methods to check whether a workflow or a node has been uploaded, or if a workflow has been uploaded together with nodes. To showcase this functionality, we run each method with positive and negative samples, i.e. strings containing document hashes we uploaded and some containing random content.
Table 8 contains all cases we evaluate. Next to the ID we describe each test case. When speaking of a stored workflow or node, we refer to workflows or nodes that we uploaded before and that hence are stored in the contract. The last column includes the line in Listing 8.3, that runs the test case. All runs were successful and provided the expected results. As Listing 8.3 shows, the first 3 cases returned True for stored data and False for random strings. To remind quickly, the `check_workflow` function returns one bool for each provided node, indicating if it is included in the workflow and the number of missing nodes. Test case 4 in line 10 passes the workflow alongside all included nodes. Hence for each node true is returned and the number of missing nodes is zero. Adding a RandomString in line 13 adds one "False" to the list. Leaving one included node out, increases the number of missing nodes to 1. For the last test case, we obtained the workflow, we uploaded initially when testing F1, in line 19. Then we passed the workflow together with two nodes, that are included in another workflow. The result indicates the expected: Both nodes do not belong to the workflow and 18 nodes are missing.

```
1  deployed_prov_contract.functions.check_workflow(wf_hash).call({"from": ca_admin_adress↩
       })
2  # outputs: True
```

```
3
4  deployed_prov_contract.functions.check_workflow("RandomString").call({"from": ↩
       ca_admin_adress})
5  # outputs: False
6
7  deployed_prov_contract.functions.check_nodes([n1_hash,n2_hash, "randomString"]).call({"↩
       from": ca_admin_adress})
8  # outputs: [True, True, False]
9
10 deployed_prov_contract.functions.check_workflow_and_nodes(wf_hash, [n1_hash, n2_hash]).↩
       call({"from": ca_admin_adress})
11 # outputs: [[True, True], 0]
12
13 deployed_prov_contract.functions.check_workflow_and_nodes(wf_hash, [n1_hash, n2_hash, "↩
       RandomString"]).call({"from": ca_admin_adress})
14 # outputs: [[True, True, False], 0]
15
16 deployed_prov_contract.functions.check_workflow_and_nodes(wf_hash, [n1_hash, "↩
       RandomString"]).call({"from": ca_admin_adress})
17 # outputs: [[True, False], 1]
18
19 wf_0_hash = deployed_prov_contract.functions.get_workflows().call({"from": ↩
       ca_admin_adress})[0][0]
20 deployed_prov_contract.functions.check_workflow_and_nodes(wf_0_hash, [n1_hash, n2_hash↩
       ]).call({"from": ca_admin_adress})
21 # outputs: [[False, False], 18]
```

Listing 8.3: Testing Check Functions of the Smart Contract

F4

The fourth and last functional requirement is, that the system shall provide the involved parties with the ability to create accounts for their users. We gonna show how each party can assign accounts to their members. After obtaining an address of the network from the network provider, the administrator of each party can assign the new account as a user or admin. Listing 8.4 shows this with the empty Accounts 4 and 5, defined earlier in Table 7. First lines 2 and 3 initialize the roles, with their keccak256 hash values. This is because, we wanna check before and after the assignment, if the accounts are assigned to the Creater and ViewerRole. Inside the Smart Contract the roles are identified by their keccak256 hashes, hence when using the hasRole function, we need to pass the roles hash as an identifier.
Line 6 and 9 show, that accounts 4 and 5 initially dont have the Creater and viewer Roles. Hence they are not allowed to send Contract Call transactions to the smart contract, which is shown later in Q1. The CreaterAdmin Account (Line 13-19) and the ViewerAdmin Account (22-28) are granting Creater and Viewer roles respectively, by calling the defined grantRole functions. In Line 30 and 33 accounts 4 and 5 are identified as Creater and Viewer and hence belong to the four roles that are permissioned to interact with the smart contract.

```
1  # initializing roles hashes
2  CREATER_ROLE = "1ac401dd2c6f22b9676f8528d637846252ce7c4e00341d11c712c96f29bc47b3"
3  VIEWER_ROLE = "4e8c8e9ca39468bb90c81b80f979119441019fdac7cbbbd930e0a604f8777d1d"
4
5  # Check if Account 4 and 5 have CreaterRole and ViewerRole
6  deployed_prov_contract.functions.hasRole(CREATER_ROLE,"0↩
       x6A3Fd877d4688bA35dCc7D134956C1D2075133e1").call()
7  # outputs: False
8
9  deployed_prov_contract.functions.hasRole(VIEWER_ROLE,"0↩
       x17035CEC72f39c1f8694EBdF2307769637BE536e").call()
10 # outputs: False
11
12 # Assinging Account 4 to CreaterRole
13 transaction = deployed_prov_contract.functions.grantCREATERRole("0↩
       x6A3Fd877d4688bA35dCc7D134956C1D2075133e1").buildTransaction({"chainId": chain_id, ↩
```

```
    "from": dlr_admin_adress, "gasPrice": w3.eth.gas_price, "nonce": w3.eth.↩
    getTransactionCount(dlr_admin_adress)})
sign_store_contact = w3.eth.account.sign_transaction(
    transaction, private_key=dlr_admin_private_key
)

send_store_contact = w3.eth.send_raw_transaction(sign_store_contact.rawTransaction)
transaction_receipt = w3.eth.wait_for_transaction_receipt(send_store_contact)

# Assinging Account 4 to ViewerRole
transaction = deployed_prov_contract.functions.grantViewerRole("0↩
    x17035CEC72f39c1f8694EBdF2307769637BE536e").buildTransaction({"chainId": chain_id, ↩
    "from": ca_admin_adress, "gasPrice": w3.eth.gas_price, "nonce":  w3.eth.↩
    getTransactionCount(ca_admin_adress)})
sign_store_contact = w3.eth.account.sign_transaction(
    transaction, private_key=ca_admin_private_key
)

send_store_contact = w3.eth.send_raw_transaction(sign_store_contact.rawTransaction)
transaction_receipt = w3.eth.wait_for_transaction_receipt(send_store_contact)

deployed_prov_contract.functions.hasRole(CREATER_ROLE,"0↩
    x6A3Fd877d4688bA35dCc7D134956C1D2075133e1").call()
# outputs: True

deployed_prov_contract.functions.hasRole(VIEWER_ROLE,"0↩
    x17035CEC72f39c1f8694EBdF2307769637BE536e").call()
# outputs: True
```

Listing 8.4: Add Users to both Parties

8.3 Quality Requirements

Q1

Moving on to quality requirements, Q1 ensures that only valid W3C PROV-JSON documents are uploaded. The graphical user interface limits the selection of documents to files with the .json format. During the pre-check, which is invoked by clicking the check files button, each file is checked, whether it contains a valid W3C PROV document.
To show that, we generated five random JSON files using a generator [94]. The files are included in the *testfiles* directory in: [93]. To check, whether the system can detect the files, we selected each of them individually and pressed the check files button. The check failed for each file and displayed the error message shown in Figure 15. Using the GUI it was not possible, to upload random documents.
However, permissioned users may use their private key to send transactions to the smart contract, avoiding the GUI. This way it is possible, to add any string to the contracts data store. While this issue is out of the scope of this thesis, we refer to it as future work, that has to be done. We believe that the risk of invalid modification is rather small. The sender of a transaction account is known, so the uploader of provenance data can be identified. If the uploader is asked for a document during certification, he would have to prove a document that produces the exact hash value. If this is an invalid provenance document, his upload would be detected.

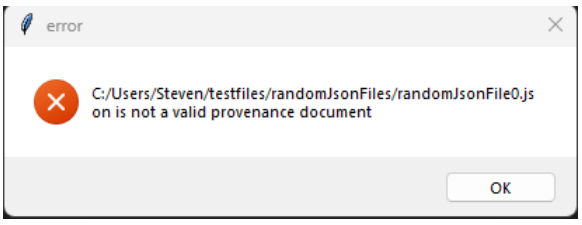


Figure 15: Error Message: Invalid PROV Documents

Q2

The system provides users a Graphical User Interface to interact with, which doesn't require knowledge of a programming language. More details about the GUI are given in section 7.2.

Q3

The system shall ensure that for an upload, an identifier of the responsible engineer and a timestamp are saved. In Blockchain timestamps are stored in the block header, when a block is added to the chain. The timestamp differs slightly from the transaction time, as adding a block takes some time depending on the consensus mechanism. As we aim to use a consortium with PoA, the deviation can be expected between a few seconds and minutes. The uploaders address is stored in the transaction. How both information can be revealed, is shown by the `find_uploader_and_time_of_workflow` function in Listing 8.5. The method takes a workflows hash as input and returns the uploader's address and timestamp.
The general strategy is to iterate over all blocks of the chain to obtain the transactions it includes. Then the passed parameters of each transaction are retrieved and decoded using the contracts abi. By decoding the parameters, we can retrieve the exact input of a function call. Then we check if the `hashed_workflow` is included inside the decoded function `hashed_workflow parameter`. If it is, we store the transaction's sender and the timestamp of the block. The functions logic is transferable to find nodes instead of workflows aswell. It is intended to integrate the functionality of delivering sender and timestamps into the Check Page of the GUI.

```
1  def find_uploader_and_time_of_workflow(hashed_workflow):
2      sender,timestamp = -1,-1
3      for i in range(2, w3.eth.blockNumber+1):
4          transactions = w3.eth.get_block(i).transactions
5          for tx_hash in transactions:
6              inp = w3.eth.get_transaction(tx_hash).input
7              decoded_inp = deployed_prov_contract.decode_function_input(inp)[1]
8              if (decoded_inp.get("hashed_workflow")==hashed_workflow):
9                  sender = w3.eth.get_transaction(tx_hash)['from']
10                 timestamp = w3.eth.getBlock(i).timestamp
11     return sender, timestamp
```

Listing 8.5: Determination of a Documents Uploader and Time

Q4

The system shall ensure that only authorized persons within the organizations can upload data. So far we granted Accounts 2 and 4 the necessary rights to upload data. To check if the Access Permission works, we test to call the upload function from Accounts 3, 5 and 6. As parameter we passed `wf1_hash` and `n1_hash` defined in Listing 5. The script defined in Listing 8.3 has been run three times, one time for each address and key. Each run produced an error message for missing permission, as shown in line 6. Contracts that are not issued with the Creater right are not capable of adding documents to the system.

```
1  deployed_prov_contract.functions.add_workflow_and_nodes(wf_hash, [n1_hash]).↩
       buildTransaction({"chainId": chain_id, "from": address, "gasPrice": w3.eth.↩
       gas_price, "nonce": w3.eth.getTransactionCount(address)})
2  sign_store_contact = w3.eth.account.sign_transaction(
3      transaction, private_key=key
4  )
5
6  # outputs: execution reverted: VM Exception while processing transaction: revert ↩
       AccessControl: account 0xf4a68d730b37925e04efe3d64cfcfda7ae3a60b3 is missing role 0↩
       x1ac401dd2c6f22b9676f8528d637846252ce7c4e00341d11c712c96f29bc47b3}. The account
```

Q5

The system shall ensure that no sensitive data, i.e. private information about participants, organizations or provenance data becomes publicly accessible. To evaluate this requirement let's first take a step back and consider which data is accessible. All methods are restricted to the defined roles. To call a `view` function, which doesn't modify the contracts state, no transaction is necessary. Hence knowing the public key of an account, that is issued with the necessary permission, is sufficient. Since all blocks and transactions can be accessed by the members of the consortium, it's possible that public addresses get exposed. Additionally, since the contract is open source other participants can obtain its abi to decompile the inputs of transactions. Overall there is a high chance, that third parties who want to obtain the data we store on the blockchain are successful with it.
To prevent sensitive data from being exposed, the proposed system dont store any sensitive information. The provenance documents are hashed before being uploaded. For each workflow and node we store its SHA-256 hash. From that hash its not possible, to obtain the actual provenance document. The actual data is safe. Also we dont store any personal information, that might expose a users identitiy. The role-based permission system works completely with addresses. The only information third parties can obtain, is the users public address. Which user is issued which address, i.e. the mapping of address to user identity, is handled by the users organization, i.e. its admin.

Q6

The last quality requirement is Q6, that mention that the stored data shall be available for at least 80 years after its upload. Although we consider blockchain as a reliable, persistent and tamper-resistant data store, there's no guarantee about how long the blockchain will run. In case the blockchain is inevitably discontinued, it is possible to export all stored data by calling the respective view functions. By iterating over the blocks additionally, information can be obtain as needed. Due to the flexibility of the system, its possible to substitute the discontinuing blockchain by another one. The proposed system can be connected to any Ethereum-based blockchain.

9 Conclusion

This thesis pursued two objectives. The first goal was to identify the current state of research regarding the storage of Provenance data using Blockchain technology. To achieve this, a Systematic Literature Review of existing works was conducted, which revealed a high number of recent publications, indicating significant interest in utilizing Blockchain technology for tamper-proof and trustworthy storage of Provenance data. The review provided a comprehensive overview of the state of research in Blockchain-based Provenance storage and discusses the benefits and challenges associated with different storage strategies. The storage types identified through the SLR were also utilized as the basis for developing a proprietary system, which leads to the second aim of the work. The second goal of the thesis was to design and implement a Blockchain-based Provenance Tracking System for the manufacturing of virtual aircraft components. The systems requirements were gathered using a systematic requirement engineering approach, which involved conducting interviews with stakeholders. These elicited requirements were used to develop a conceptual design that aimed to meet the needs of the stakeholders. A practical implementation was provided, that involved the development of smart contracts and a graphical user interface, which enables users to interact with the system. In the evaluation stage, it was shown that the proposed system met all the requirements. The proposed system effectively achieves a balance between ensuring tamper-resistant and traceable storage of pertinent data on a transparent Blockchain while also safeguarding the privacy of users and documents. This is achieved through the storage of the Provenance document's fingerprints and leveraging the authentication system of the Blockchain.

Overall, the thesis addresses the intersection of Provenance and Blockchain technology and offers two significant contributions. The first contribution involves identifying and analyzing the existing literature in the field of Provenance and Blockchain. The second contribution demonstrates how Blockchain technology can be used to implement a system for certifying aircraft components. By combining these two contributions, this study offers valuable insights into the potential applications of Blockchain technology for the storage of Provenance data and beyond. We hope the findings of this study provide a valuable resource for researchers and industry professionals seeking to leverage Blockchain technology for Provenance tracking and certification purposes.

While the thesis demonstrates how Blockchain technology can be leveraged for Provenance tracking and certification purposes, there are some limitations to its implementation that need to be taken into account. One limitation of the proposed implementation is, that it cannot protect the smart contracts data store from unintended inputs, that are uploaded without using the GUI. The risk of authenticated users connecting directly to the Blockchain using their key and uploading arbitrary data remains unresolved. This shortcoming could be addressed by moving the pre-checks of documents directly on the chain, which can be investigated in further works. Another limitation of this study is the limited scope of the implementation. The thesis focuses on designing and implementing a Blockchain-based Provenance tracking system for aircraft manufacturing. While this is a valuable contribution to the field, the scope is relatively narrow and may not be applicable to other industries or use cases. Additionally, while the proposed system might be useful for the specific use case, there may be significant differences in the requirements and implementation details for other use cases inside the same industry. Furthermore, while conducting a requirement-based evaluation, the thesis lacks a comprehensive evaluation of the system's performance and effectiveness, which leaves the possibility of unanticipated limitations open. The presented GUI was also not tested for its usability, which would be essential in the

context of future work to enable its user-friendly use. Lastly, the Systematic Literature Review conducted was limited to a few attributes due to the limitation of a master's thesis. Future reviews could include which factors motivate the authors to adopt Blockchain technology. This can help to understand which features make Blockchain a popular technology for storing Provenance. The theoretical foundation has shown that Blockchain technology is still evolving, and the absence of established practices necessitates experimentation, evaluation, and possible rejection of approaches. The thesis aimed to contribute to the current understanding of Blockchain and Provenance, and it is anticipated that forthcoming advancements in Blockchain, such as DAG-based frameworks, may further enhance the storage of Provenance information.

Statement of Authorship

Hiermit erkläre ich, dass ich die vorliegende Bachelorarbeit selbstständig und außschließlich unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.
Die aus fremden Quellen direkt oder indirekt übernommenen Stellen sind als solche kenn-tlich gemacht.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form weder einer anderen Prüfungsbehörde vorgelegt oder noch anderweitig veröffentlicht.

Name: Kocadag
Vorname: Steven
Geburtsdatum: 27.05.1994
Matrikelnummer: 211218

S.Kocadag
Unterschrift

14.04.2023
Datum

Appendices

A. PROV-DM Types and Relations

Type or Relation Name	Representation in the PROV-N notation
Entity	entity(id, [attr1=val1, ...])
Activity	activity(id, st, et, [attr1=val1, ...])
Generation	wasGeneratedBy(id;e,a,t,attrs)
Usage	used(id;a,e,t,attrs)
Communication	wasInformedBy(id;a2,a1,attrs)
Start	wasStartedBy(id;a2,e,a1,t,attrs)
End	wasEndedBy(id;a2,e,a1,t,attrs)
Invalidation	wasInvalidatedBy(id;e,a,t,attrs)
Derivation	wasDerivedFrom(id; e2, e1, a, g2, u1, attrs)
Revision	... prov:type='prov:Revision' ...
Quotation	... prov:type='prov:Quotation' ...
Primary Source	... prov:type='prov:PrimarySource' ...
Agent	agent(id, [attr1=val1, ...])
Attribution	wasAttributedTo(id;e,ag,attr)
Association	wasAssociatedWith(id;a,ag,pl,attrs)
Delegation	actedOnBehalfOf(id;ag2,ag1,a,attrs)
Plan	... prov:type='prov:Plan' ...
Person	... prov:type='prov:Person' ...
Organization	... prov:type='prov:Organization' ...
SoftwareAgent	... prov:type='prov:SoftwareAgent' ...
Influence	wasInfluencedBy(id;e2,e1,attrs)
Bundle constructor	bundle id description_1 ... description_n endBundle
Bundle type	... prov:type='prov:Bundle' ...
Alternate	alternateOf(alt1, alt2)
Specialization	specializationOf(infra, supra)
Collection	... prov:type='prov:Collection' ...
EmptyCollection	... prov:type='prov:EmptyCollection' ...
Membership	hadMember(c,e)

Table 9: PROV-DM Types and Relations. Own Representation based on: [7]

B. Collection of Studies

Ref	Author	Title	In
[95]	Amin et al.	A Step toward Next-Generation Advancements in the Internet of Things Technologies	0
[96]	Boland et al.	Modeling and contextualizing claims	0

[66]	Bose et al.	BLINKER: A Blockchain-Enabled Framework for Software Provenance	1
[71]	Coelho et al.	Integrating blockchain for data sharing and collaboration support in scientific ecosystem platform	1
[72]	Coelho et al.	A Blockchain-Based Architecture for Trust in Collaborative Scientific Experimentation	0
[53]	Dang and Duong	An effective and elastic blockchain-based provenance preserving solution for the open data	1
[52]	Dang and Anh	A Pragmatic Blockchain Based Solution forManaging Provenance and Characteristics in the Open Data Context	1
[54]	Demichev et al.	Business Process Engineering for Data Storing and Processing in a Collaborative Distributed Environment	1
[58]	Fadhel et al.	Towards a semantic modelling for threat analysis of IoT applications: A case study on transactive energy	1
[97]	Geng et al.	Novel blockchain transaction provenance model with graph attention mechanism	0
[98]	Gouru and Vadlamani	DistProv-Data Provenance in Distributed Cloud for Secure Transfer of Digital Assets	0
[99]	Hasan et al.	Hybrid Blockchain Architecture for Cloud Manufacturing-as-a-service (CMaaS) Platforms	0
[56]	Hogan and Helfert	Transparent cloud privacy: Data provenance expression in blockchain	1
[57]	Ioini and Pahl	Trustworthy orchestration of container based edge computing using permissioned blockchain	1
[100]	Isaac Abiodun et al.	Data provenance for cloud forensic investigations, security, challenges, solutions and future perspectives	0
[101]	Kaaniche et al.	Prov-Trust: Towards a trustworthysgx-based data provenance system	0
[102]	Kak et al.	Privacy improvement architecture for IoT	0
[55]	Kirstein	A decentralized provenance network for linked open data	1
[103]	Komamizu et al.	Analyzing Japanese law history through modeling multi-versioned entity	0
[63]	Lautert et al.	A fog architecture for privacy-preserving data provenance using blockchains	1
[104]	Li et al.	Blockchain technology for vector geographic provenance information organization and verification	0
[29]	Liang et al.	ProvChain: A Blockchain-Based Data Provenance Architecture in Cloud Environment	1
[105]	Linoy et al.	EtherProv: Provenance-Aware Detection, Analysis, and Mitigation of Ethereum Smart Contract Security Issues	0
[62]	Margheri et al.	Decentralised provenance for healthcare data	1
[60]	Markovic et al.	Recording Provenance of Food Delivery Using IoT, Semantics and Business Blockchain Networks	1
[59]	Markovic et al.	Integrating Internet of Things, Provenance, and Blockchain to Enhance Trust in Last Mile Food Deliveries	1
[69]	Ramachandran and Kantarcioglu	SmartProvenance: A Distributed, Blockchain Based DataProvenance System	1

[64]	Sigwart et al.	A secure and extensible blockchain-based data provenance framework for the Internet of Things	1
[61]	Song et al.	An Improved Data Provenance Framework Integrating Blockchain and PROV Model	1
[67]	Sun et al.	BSTProv: Blockchain-Based Secure and Trustworthy Data Provenance Sharing	1
[70]	Tunstad et al.	HyperProv: Decentralized resilient data provenance at the edge with blockchains	1
[106]	Zayas et al.	An Integrated Blockchain Approach for Provenance of Rotorcraft Maintenance Data	0
[107]	Zhang and Mao	Blockchain-based decentralized data provenance method	0
[108]	Zhang et al.	Trusted Query Method for Data Provenance Based on Blockchain	0
[65]	Zhang et al.	Research on Consistency Tracing Technology of Dispatching Control Model Data based on Blockchain	1

C. PROV Document of a Workflow


```
{
  "bundle": {
    "rceId:a6bc7097-ff9a-4817-af57-fd7d286a56c7": {
      "agent": {
        "rceId:RCE": {
          "prov:type": {
            "$": "software",
            "type": "prov:QUALIFIED_NAME"
          },
          "version": {
            "$": "10.2.1",
            "type": "xsd:string"
          }
        },
        "rceId:Steven": {
          "prov:type": {
            "$": "user",
            "type": "prov:QUALIFIED_NAME"
          },
          "username": {
            "$": "Human-readable name of user not supplied",
            "type": "xsd:string"
          }
        }
      },
      "activity": {
        "rceId:a14cc9d9-b54e-4bae-abfa-2af271c15c7c": {
          "prov:startTime": "2023-03-01T18:58:44.210+03:00",
          "label": {
            "$": "runWorkflowNode",
            "type": "xsd:string"
          },
          "prov:endTime": "2023-03-01T18:58:44.264+03:00"
        },
        "var:activity_ID_runTool": {
          "label": {
            "$": "runTool",
            "type": "xsd:string"
          }
        }
      },
      "prefix": {
        "xsd": "http://www.w3.org/2001/XMLSchema#",
        "pre_0": "rce2prov",
```

```
      "default": "rce2prov",
      "var": "http://openprovenance.org/var#",
      "prov": "http://www.w3.org/ns/prov#",
      "rceId": "https://namespace.dlr.de/prov/rce#"
    },
    "wasStartedBy": {
      "_:wSB24": {
        "prov:trigger": "rceId:RCE",
        "prov:activity": "var:activity_ID_runTool"
      }
    },
    "wasDerivedFrom": {
      "_:wDF39": {
        "prov:generatedEntity": "var:entity_ID_outputVersion",
        "prov:usedEntity": "rceId:5edd9fd5-9d7f-4814-90f4-e9ad99a7b591"
      },
      "_:wDF38": {
        "prov:generatedEntity": "var:entity_ID_outputVersion",
        "prov:usedEntity": "rceId:f84881aa-3967-4135-8beb-197c3273e60b"
      }
    },
    "used": {
      "_:u109": {
        "prov:entity": "rceId:e430a509-f919-4bf0-a5cf-6b7fe234302b",
        "prov:role": {
          "$": "placeholderAssignment",
          "type": "prov:QUALIFIED_NAME"
        },
        "prov:activity": "rceId:a14cc9d9-b54e-4bae-abfa-2af271c15c7c"
      },
      "_:u111": {
        "prov:entity": "rceId:f84881aa-3967-4135-8beb-197c3273e60b",
        "prov:activity": "rceId:a14cc9d9-b54e-4bae-abfa-2af271c15c7c"
      }
    },
    "hadMember": {
      "_:hM105": {
        "prov:entity": [
          "var:entity_ID_outputVersion"
        ],
        "prov:collection": "rceId:b6aa9a2b-1442-441e-b8e1-6958affd3c02"
      },
      "_:hM103": {
        "prov:entity": [
          "rceId:f84881aa-3967-4135-8beb-197c3273e60b"
        ],
        "prov:collection": "rceId:93541678-dda1-4ea7-83d6-fb36c7850e89"
      },
      "_:hM104": {
        "prov:entity": [
          "rceId:5edd9fd5-9d7f-4814-90f4-e9ad99a7b591"
        ],
        "prov:collection": "rceId:93541678-dda1-4ea7-83d6-fb36c7850e89"
      }
    },
    "wasGeneratedBy": {
      "_:wGB109": {
        "prov:entity": "var:entity_ID_outputVersion",
        "prov:activity": "rceId:a14cc9d9-b54e-4bae-abfa-2af271c15c7c"
      },
      "_:wGB108": {
        "prov:entity": "rceId:b6aa9a2b-1442-441e-b8e1-6958affd3c02",
        "prov:activity": "rceId:a14cc9d9-b54e-4bae-abfa-2af271c15c7c"
      },
      "_:wGB110": {
        "prov:entity": "var:bundle_ID_runToolBundle",
        "prov:activity": "rceId:a14cc9d9-b54e-4bae-abfa-2af271c15c7c"
      }
    },
    "actedOnBehalfOf": {
      "_:aOBO25": {
        "prov:responsible": "rceId:Steven",
        "prov:delegate": "rceId:RCE"
```

```
        }
      },
      "wasInformedBy": {
        "_:Infm42": {
          "prov:informed": "rceId:a14cc9d9-b54e-4bae-abfa-2af271c15c7c",
          "prov:informant": "var:activity_ID_runTool"
        }
      },
      "wasAssociatedWith": {
        "_:wAW26": {
          "prov:agent": "rceId:RCE",
          "prov:activity": "rceId:a14cc9d9-b54e-4bae-abfa-2af271c15c7c"
        }
      },
      "wasAttributedTo": {
        "_:wAT61": {
          "prov:entity": "var:bundle_ID_runToolBundle",
          "prov:agent": "rceId:RCE"
        }
      },
      "entity": {
        "rceId:d39f72a6-1a74-48e0-a0be-9b4d857fcaa3": {
          "prov:location": {
            "$": "\"Default instance started by \"Steven\" on STB-NB23\" [151↩
                dcce9b3c3d405f47686e55e98fe74:0]",
            "type": "xsd:string"
          },
          "name": {
            "$": "Design of Experiments",
            "type": "xsd:string"
          },
          "executionInformation": {
            "$": "NodeConfiguration(\"{}\")",
            "type": "xsd:string"
          }
        },
        "rceId:7773c180-b567-4850-8dbc-a2f255237b49": {
          "datatype": {
            "$": "Float",
            "type": "xsd:string"
          },
          "handling": {
            "$": "somehandling",
            "type": "xsd:string"
          },
          "label": {
            "$": "input",
            "type": "xsd:string"
          }
    }},
      "specializationOf": {
        "_:sO99": {
          "prov:specificEntity": "rceId:f84881aa-3967-4135-8beb-197c3273e60b",
          "prov:generalEntity": "rceId:fe3b0194-7f02-44fb-aa35-1e7eaf8519ec"
        },
        "_:sO100": {
          "prov:specificEntity": "rceId:f84881aa-3967-4135-8beb-197c3273e60b",
          "prov:generalEntity": "rceId:7773c180-b567-4850-8dbc-a2f255237b49"
        }
      }
    }
  }
}
```


Listing 1: PROV Document of a Workflow

D. Implementation

D.1 Smart Contract

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.8.0;

import "OpenZeppelin/openzeppelin-contracts@4.8.2/contracts/access/AccessControl.sol";

contract Prov is AccessControl {
    bytes32 private constant CREATER_ROLE = keccak256("CREATER_ROLE");
    bytes32 private constant VIEWER_ROLE = keccak256("VIEWER_ROLE");
    bytes32 private constant CREATER_ADMIN_ROLE =
        keccak256("CREATER_ADMIN_ROLE");
    bytes32 private constant VIEWER_ADMIN_ROLE = keccak256("VIEWER_ADMIN_ROLE");

    struct Workflow {
        string hash;
        uint256[] nodes;
    }

    //arrays for list of prov graphs
    Workflow[] private workflows;
    string[] private nodes;

    constructor(address creater_admin, address viewer_admin) AccessControl() {
        _grantRole(CREATER_ADMIN_ROLE, creater_admin);
        _grantRole(CREATER_ROLE, creater_admin);
        _grantRole(VIEWER_ADMIN_ROLE, viewer_admin);
        _grantRole(VIEWER_ROLE, viewer_admin);
    }

    function grantViewerAdminRole(
        address account
    ) public onlyRole(VIEWER_ADMIN_ROLE) {
        _grantRole(VIEWER_ADMIN_ROLE, account);
        _grantRole(VIEWER_ROLE, account);
    }

    function grantCreaterAdminRole(
        address account
    ) public onlyRole(CREATER_ADMIN_ROLE) {
        _grantRole(CREATER_ADMIN_ROLE, account);
        _grantRole(CREATER_ROLE, account);
    }

    function grantViewerRole(
        address account
    ) public onlyRole(VIEWER_ADMIN_ROLE) {
        _grantRole(VIEWER_ROLE, account);
    }

    function grantCreaterRole(
        address account
    ) public onlyRole(CREATER_ADMIN_ROLE) {
        _grantRole(CREATER_ROLE, account);
    }

    function hasAccess() internal view returns (bool) {
        return (hasRole(CREATER_ADMIN_ROLE, msg.sender) ||
            hasRole(CREATER_ROLE, msg.sender) ||
            hasRole(VIEWER_ROLE, msg.sender) ||
            hasRole(VIEWER_ADMIN_ROLE, msg.sender));
    }

    function get_workflows() public view returns (Workflow[] memory) {
        require(hasAccess(), "Access Denied");
        return workflows;
    }

    function get_nodes() public view returns (string[] memory) {
```

```
68          require(hasAccess(), "Access Denied");
69          return nodes;
70      }
71
72      function check_workflow(string memory prov) public view returns (bool) {
73          require(hasAccess(), "Access Denied");
74          for (uint i = 0; i < workflows.length; i++) {
75              if (
76                  keccak256(abi.encodePacked(workflows[i].hash)) ==
77                  keccak256(abi.encodePacked(prov))
78              ) {
79                  return true;
80              }
81          }
82          return false;
83      }
84
85      function check_nodes(
86          string[] memory node_list
87      ) public view returns (bool[] memory) {
88          require(hasAccess(), "Access Denied");
89          bool[] memory result = new bool[](node_list.length);
90          for (uint i = 0; i < node_list.length; i++) {
91              result[i] = false;
92              for (uint j = 0; j < nodes.length; j++) {
93                  if (
94                      keccak256(abi.encodePacked(nodes[j])) ==
95                      keccak256(abi.encodePacked(node_list[i]))
96                  ) {
97                      result[i] = true;
98                      break;
99                  }
100             }
101         }
102         return result;
103     }
104
105     function check_workflow_and_nodes(
106         string memory hashed_workflow,
107         string[] memory hashed_nodes
108     ) public view returns (bool[] memory, uint256) {
109         require(hasAccess(), "Access Denied");
110         int index = find_workflow(hashed_workflow);
111         bool[] memory node_check_result = new bool[](hashed_nodes.length);
112
113         // workflow not existing
114         if (index == -1) {
115             return (node_check_result, 0);
116         }
117
118         uint u_index = uint256(index);
119
120         uint256[] memory wf_nodes = workflows[u_index].nodes;
121         uint256 found_nodes = 0;
122         for (uint i = 0; i < hashed_nodes.length; i++) {
123             for (uint j = 0; j < wf_nodes.length; j++) {
124                 if (
125                     keccak256(abi.encodePacked(hashed_nodes[i])) ==
126                     keccak256(abi.encodePacked(nodes[wf_nodes[j]]))
127                 ) {
128                     node_check_result[i] = true;
129                     found_nodes++;
130                     break;
131                 }
132             }
133         }
134
135         return (node_check_result, wf_nodes.length - found_nodes);
136     }
137
138     function find_workflow(string memory prov) private view returns (int) {
139         require(hasAccess(), "Access Denied");
140         for (uint i = 0; i < workflows.length; i++) {
```

```
141             if (
142                 keccak256(abi.encodePacked(workflows[i].hash)) ==
143                 keccak256(abi.encodePacked(prov))
144             ) {
145                 return int(i);
146             }
147         }
148         return -1;
149     }
150 
151     function add_workflow_and_nodes(
152         string memory hashed_workflow,
153         string[] memory hashed_nodes
154     ) public onlyRole(CREATER_ROLE) {
155         uint256 node_id = nodes.length;
156         uint256[] memory node_list = new uint256[](hashed_nodes.length);
157 
158         for (uint i = 0; i < hashed_nodes.length; i++) {
159             nodes.push(hashed_nodes[i]);
160             node_list[i] = node_id;
161             node_id++;
162         }
163         workflows.push(Workflow(hashed_workflow, node_list));
164     }
165 
166 
167 }
```

Listing 2: Smart Contract `prov.sol`

D.2 Class Diagram of Smart Contract

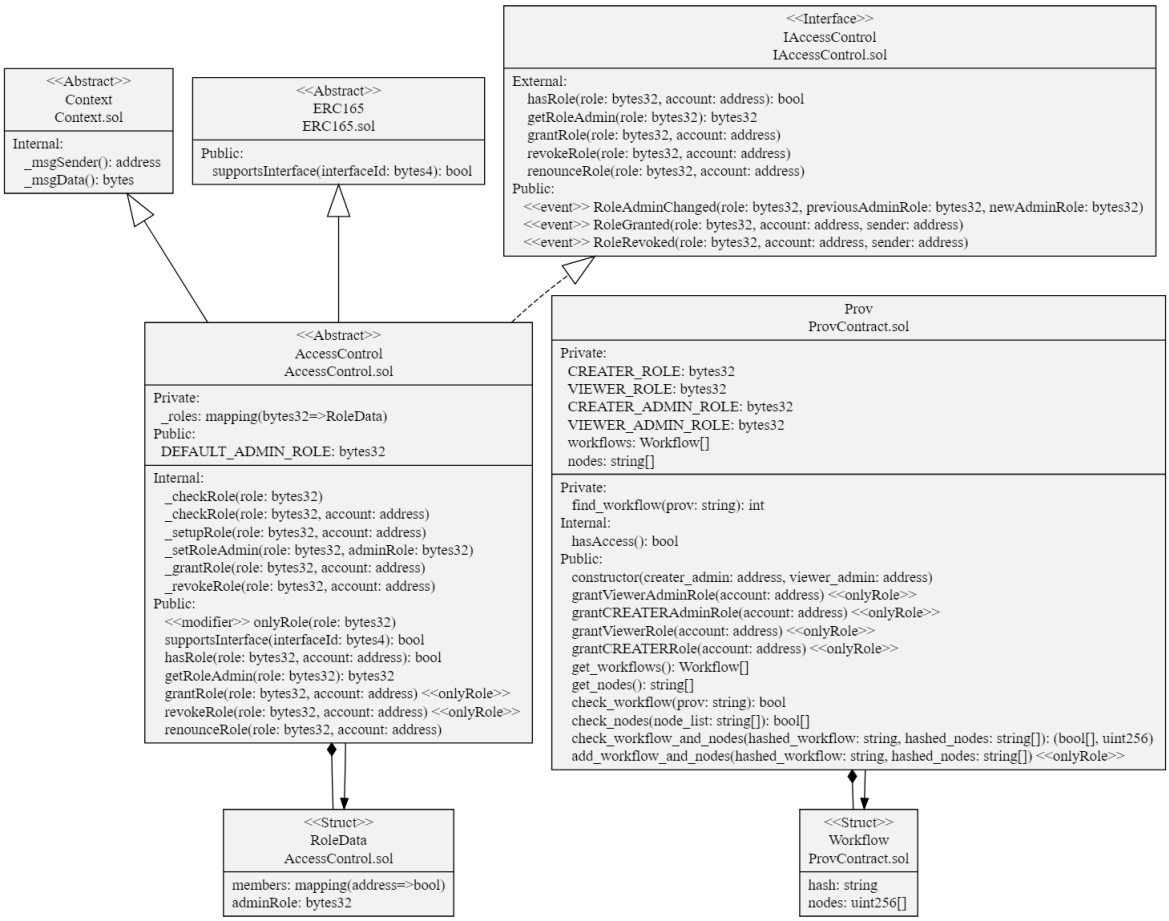


Figure 16: Class Diagram of the Smart Contract `prov.sol`

D.3 GUI Upload Page

```python
from pathlib import Path
from prov.model import ProvDocument
import lxml.etree as ET
from dotenv import load_dotenv
import jcs
from hashlib import sha256

# from tkinter import *
# Explicit imports to satisfy Flake8
from tkinter import (
    Tk,
    Canvas,
    Entry,
    Text,
    Button,
    PhotoImage,
    filedialog,
    simpledialog,
)
from tkinter.messagebox import showerror, showinfo

import json
from Web3Provider import Web3Provider

OUTPUT_PATH = Path(__file__).parent
ASSETS_PATH = OUTPUT_PATH / Path(
    r"C:\Users\Steven\Google Drive\Meine Ablage\Masterarbeit\Code\tkinter\build\assets\↩
        frame0"
```

```
)


def relative_to_assets(path: str) -> Path:
    return ASSETS_PATH / Path(path)


class UploadPage(Tk):
    def __init__(self, *args, **kwargs):
        Tk.__init__(self)
        self.title("Docu Sphere")

        self.geometry("450x600")
        self.configure(bg="#202020")

        self.itemlist = []

        self.canvas = Canvas(
            self,
            bg="#202020",
            height=600,
            width=450,
            bd=0,
            highlightthickness=0,
            relief="ridge",
        )
        self.canvas.place(x=0, y=0)

        self.canvas.create_text(
            225.0,
            75.0,
            anchor="center",
            text="Upload Documents",
            fill="#FFFFFF",
            font=("Helvetica", 14, "bold"),
        )

        self.button_image_5 = PhotoImage(file=relative_to_assets("button_5.png"))
        self.button_1 = Button(
            self.canvas,
            image=self.button_image_5,
            borderwidth=0,
            highlightthickness=0,
            command=self.check_files,
            relief="flat",
        )
        self.button_1["state"] = "disabled"

        self.button_1.place(x=16.0, y=176.0, width=414.0, height=47.0)

        self.entry_image_1 = PhotoImage(file=relative_to_assets("entry_1.png"))
        entry_bg_1 = self.canvas.create_image(210.0, 138.0, image=self.entry_image_1)
        self.entry_1 = Entry(bd=0, bg="#2D2D2D", fg="#000716", highlightthickness=0)
        self.entry_1.place(x=37.0, y=118.0, width=346.0, height=38.0)

        self.image_image_1 = PhotoImage(file=relative_to_assets("image_1.png"))
        image_1 = self.canvas.create_image(224.5, 137.5, image=self.image_image_1)

        self.button_image_2 = PhotoImage(file=relative_to_assets("button_2.png"))
        self.button_2 = Button(
            image=self.button_image_2,
            borderwidth=0,
            highlightthickness=0,
            command=self.select_files,
            relief="flat",
        )

        self.button_2.place(
            x=402.0, y=130.39999389648438, width=16.0, height=15.20001220703125
        )

        self.canvas.create_text(
            25.0,
```

```
            98.0,
            anchor="nw",
            text="Input Directory",
            fill="#FFFFFF",
            font=("Roboto Medium", 14 * -1),
        )

        self.image_image_2 = PhotoImage(file=relative_to_assets("image_2.png"))
        image_2 = self.canvas.create_image(225.0, 37.0, image=self.image_image_2)

        self.button_image_3 = PhotoImage(file=relative_to_assets("clear.png"))
        self.button_3 = Button(
            image=self.button_image_3,
            borderwidth=0,
            highlightthickness=0,
            command=self.clear_files,
            relief="flat",
        )
        self.button_3.place(x=20.0, y=21.0, width=30.0, height=30.0)

        self.button_image_4 = PhotoImage(file=relative_to_assets("button_4.png"))
        self.button_4 = Button(
            image=self.button_image_4,
            borderwidth=0,
            highlightthickness=0,
            command=lambda: print("button_4 clicked"),
            relief="flat",
        )
        self.button_4.place(x=400.0, y=21.0, width=30.0, height=30.0)

        self.resizable(False, False)

    def clear_files(self):
        for item_id in self.itemlist:
            self.canvas.delete(item_id)
        self.itemlist = []

        self.button_image_1 = PhotoImage(file=relative_to_assets("button_5.png"))
        self.button_1.configure(image=self.button_image_1, command=self.upload_files)
        self.button_1.place(x=16.0, y=176.0, width=414.0, height=47.0)
        self.button_1["state"] = "disabled"

    def select_files(self):
        filez = filedialog.askopenfilenames(parent=self, title="Choose a file")

        pad = len(self.itemlist) * 20.0
        for file in list(filez):

            # if file is stored already, skip it
            if self.is_file_already_in_itemlist(file):
                continue

            # check if file is json
            if file[-4:] != "json":
                showerror(title="Error", message=f"{file} is not a valid json file")
                continue

            self.button_1.place(x=16.0, y=196.0 + pad, width=414.0, height=47.0)

            self.itemlist.append(
                self.canvas.create_text(
                    25.0,
                    176.0 + pad,
                    anchor="nw",
                    text=file,
                    fill="#FFFFFF",
                    font=("Roboto Medium", 14 * -1),
                )
            )
            pad += 20.0

        if len(self.itemlist) > 0:
            self.button_1.config(image=self.button_image_5, command=self.check_files)
```

```
174             self.button_1["state"] = "normal"
175
176     def is_file_already_in_itemlist(self, file):
177         for id in self.itemlist:
178             if self.canvas.itemcget(id, "text") == file:
179                 return True
180         return False
181
182     def check_files(self):
183         components_in_workflow = 0
184         uploaded_components = 0
185         for file in self.itemlist:
186             path = self.canvas.itemcget(file, "text")
187             try:
188                 json = ProvDocument.deserialize(source=path, format="json")
189             except:
190                 showerror(
191                     title="error",
192                     message=f"{path} is not a valid provenance document",
193                 )
194             return
195
196             xml = json.serialize(format="xml")
197             root = ET.fromstring(xml.encode("utf-8"))
198
199             # check if number of components in workflow is equal to number of uploaded ↩
                    nodes
200             if "runWorkflow" in path and "Node" not in path:
201                 # file is a workflow
202                 if components_in_workflow != 0:
203                     showerror(
204                         title="error",
205                         message="Dont upload more then one workflow at the same time!",
206                     )
207                     return
208                 for element in root.getchildren()[0].getchildren():
209                     if "hadMember" in element.tag:
210                         components_in_workflow += 1
211             elif "runWorkflowNode" in path:
212                 # file is a node
213                 uploaded_components += 1
214
215             else:
216                 showerror(
217                     title="error",
218                     message=f"{path} is not named like a workflow or node",
219                 )
220
221         if components_in_workflow == 0:
222             showerror(title="error", message="No workflow was selected")
223         elif components_in_workflow == uploaded_components:
224             showinfo(
225                 title="Check succesful",
226                 message="Check successful. Files can be uploaded.",
227             )
228             self.button_image_1 = PhotoImage(file=relative_to_assets("button_1.png"))
229             self.button_1.configure(
230                 image=self.button_image_1, command=self.upload_files
231             )
232         else:
233             showerror(
234                 title="error",
235                 message="Number of selected Nodes is not maching number of nodes in the↩
                        provided workflow",
236             )
237
238     def upload_files(self):
239         private_key = simpledialog.askstring(title="Key", prompt="Insert Private Key")
240         workflow, nodes = "", []
241         for item_id in self.itemlist:
242             path = self.canvas.itemcget(item_id, "text")
243             file_reader = open(path)
244             json_file = jcs.canonicalize(json.load(file_reader), utf8=False)
```

```
            hash = sha256(json_file.encode("utf-8")).hexdigest()
            if "runWorkflow" in path and "Node" not in path:
                workflow = hash
            else:
                nodes.append(hash)
        try:
            Web3Provider.upload_nodes_and_workflows(private_key, workflow, nodes)
        except Exception as e:
            showerror("Error!", e)
        else:
            showinfo("Upload successful!", "Done! Documents uploaded succesfully.")

        self.clear_files()


if __name__ == "__main__":
    new = UploadPage()
    new.mainloop()
```

Listing 3: Upload Page of GUI `UploadPage.py`

D.4 WebProvider

```
import json
from web3 import Web3
from dotenv import load_dotenv
import os


class Web3Provider(object):
    load_dotenv()
    transaction_address = os.environ.get("contract_address")
    abi_path = os.environ.get("path_to_abi")

    w3 = Web3(
        Web3.HTTPProvider("http://127.0.0.1:7545", request_kwargs={"timeout": 600})
    )
    chain_id = 1337
    with open(abi_path, "r") as file:
        compiled_sol = json.load(file)

        # get abi
        abi = compiled_sol["abi"]

    deployed_prov_contract = w3.eth.contract(address=transaction_address, abi=abi)

    @classmethod
    def upload_nodes_and_workflows(cls, private_key, workflow, nodes=[]):
        address = cls.w3.eth.account.privateKeyToAccount(private_key).address
        nonce = cls.w3.eth.getTransactionCount(address)
        transaction = cls.deployed_prov_contract.functions.add_workflow_and_node(
            workflow, nodes
        ).buildTransaction(
            {
                "chainId": cls.chain_id,
                "from": address,
                "gasPrice": cls.w3.eth.gas_price,
                "nonce": nonce,
            }
        )
        sign_store_contact = cls.w3.eth.account.sign_transaction(
            transaction, private_key=private_key
        )
        print("Deploying Contract!")
        # Send the transaction
        send_store_contact = cls.w3.eth.send_raw_transaction(
            sign_store_contact.rawTransaction
        )
        print("Waiting for transaction to finish...")
        transaction_receipt = cls.w3.eth.wait_for_transaction_receipt(
```

```
48              send_store_contact
49          )
50          print(transaction_receipt)
51          print(f"Done! Contract deployed to {transaction_receipt.contractAddress}")
```

Listing 4: Webprovider Class `WebProvider.py`

E. Evaluation

```
1  import jcs
2  from hashlib import sha256
3  # loading workflows and nodes provenance documents
4  w_file = open('testfiles/runWorkflow6.json')
5  n1_file = open('testfiles/runWorkflowNode125.json')
6  n2_file = open('testfiles/runWorkflowNode126.json')
7
8  # canonicalizing documents
9  wf = jcs.canonicalize(json.load(w_file), utf8=False)
10 n1 = jcs.canonicalize(json.load(n1_file), utf8=False)
11 n2 = jcs.canonicalize(json.load(n2_file), utf8=False)
12
13 # hashing documents
14 wf_hash = sha256(wf.encode("utf-8")).hexdigest()
15 n1_hash = sha256(n1.encode("utf-8")).hexdigest()
16 n2_hash = sha256(n2.encode("utf-8")).hexdigest()
17
18 transaction = deployed_prov_contract.functions.add_workflow_and_nodes(wf_hash, [n1_hash↩
       , n2_hash]).buildTransaction({"chainId": chain_id, "from": dlr_admin_adress, "↩
       gasPrice": w3.eth.gas_price, "nonce": w3.eth.getTransactionCount(dlr_admin_adress)↩
       })
19 sign_store_contact = w3.eth.account.sign_transaction(
20     transaction, private_key=dlr_admin_private_key
21 )
22 send_store_contact = w3.eth.send_raw_transaction(sign_store_contact.rawTransaction)
23 transaction_receipt = w3.eth.wait_for_transaction_receipt(send_store_contact)
```

Listing 5: Upload of Nodes and Workflows

F. Data Extraction Table

Authors	Year	Goal	Area	Blockchain	Provenance Standard	Provenance Data Generation	Blockchain Storage
Dang and Anh	2020	Track Data Provenance	Open Data	Hyperledger Fabric	W3C PROV	1. Tool receives a synchronization request from Open Data Platform, when user changes data 2. Received context changes of the Objects properties result in provenance data	Data is stored in 5 Smart Contracts: Resources, Datasets, User, Portal, and Provenance, which combine the other 4 elements to actual Provenance information.
Bose et al	2019	Capture, store, explore and analyze provenance data	Software Provenance	Ethereum (TestRPC)	SWProcess Specification (Prov for Software Development)	Review Data is converted to SWProcess	1. Storing critical PROV Data and the Hash in Blockchain 2. Storing full PROV Data in Off-Chain DB 3. Suggestion: Cryptographic immutable distributed databases such as IPFS
Sun et al	2022	Sharing of Provenance Data	Provenance	Ethereum	W3C PROV		1 PROV graph is partitioned into several subgraphs (BFS-based Provenance Graph Partition) 2. Subgraphs are encrypted and uploaded onto the chain 3. users can then obtain a subset of provenance subgraphs and compose them into a new graph
Margheri et. al	2020	Track Data Provenance	Healthcare	Hyperledger Fabric	W3C PROV	Health Documents are considered as Entities of PROV document: - Non-CDA's are one PROV; - CDA's are getting split section wise into multiple PROVs	Key value pairs: <h(doc), Prov(doc)>, Where h is signature (hash) and Prov is in PROV format - Hash is calculated by Canonicalization - Whenever a Patients doc is modified, a new provenance tuple is stored

							- When Doc is a CDA, each section is stored and linked to master doc
Zhang et al	2021	Dispatching and control of Provenance Data	Power grids	Hyperledger Fabric	Modified PROV for Power-Grid		3 Smart Contracts 1. Data access: Put, Delete, Update and Query. Before/After each Operation Data Logging contract gets triggered 2. Data Logging: Create Provenance chain based on activity (init, delete, update, query). Each chain represents change of data 3. Data Provenance: Access data operation log and create a complete provenance chain.
Kirstein	2019	Defining requirement for a system that track and store provenance data	Linked Open Data		W3C Prov		
Lautert et al	2020	Track data provenance	Fog Computing	Tendermint	W3C Prov		Fog Architecture: Multiple Fogs own a private Blockchain each, that store Provenance. When it's time to disclose the data it is shared with clouds global blockchain
Sigwart et al	2020	Generic Framework to track provenance	Internet of Things (IoT)		Data Provenance model for IoT (olufowobi et al)		1. storage layer: low-level representation and storage of provenance data (o create, retrieve, update and delete)

							2. generic provenance layer: general-purpose provenance functionality 3. specific provenance layer: fine-tune the framework to requirements of specific use-case
Dang und Duong	2021	Keep track of data changes and share them with the open data platform	Open Data	Hyperledger Fabric	W3C PROV	1. Tool receives a synchronization request from Open Data Platform, when data is changed 2. Received context changes of the Objects properties result in provenance data	URL + Checksum of dataset is stored in Blockchain
Demichev	2021	Decentralized Data Management System (PROVHL)	Business Process Engineering	Hyperledger Fabric	Adapted W3C Prov: {Asset (entity); Operation/Transaction (Activity); Participant (Agent)}		Key value pairs of assets (e.g. Files), e.g. file-name, storageId, ownerID, creationDate etc..
Tunstadt et al	2019	general framework for tracking data provenance (HyperPROV)	Provenance	Hyperledger Fabric	Open Provenance Model		On-Chain: checksum, editors, operations, data ownership, and data pointers Off-Chain: Actual data
Coelho et al	2021	capture, store, analysis of provenance related data	Collaborative Research	Hyperledger Fabric	ProvONE (PROV for scientiftic workflows)		Off-Chain: The RESTful API web service, the Client, Wrapper, and Data layers On-Chain: Chaincode

Markovic et al	2020	Monitor and Track Food Provenance	Food Deliveries	Hyperledger Fabric	PROV-related ontology EP-PLAN (PROV extension for linking Plans)		Participants involved in preparing/delivering food, assets (the food) and transactions (delivery of assets) are stored on Blockchain. Prov Graph is exported as text and stored in Transaction attribute (complianceReport)
Liang et al	2017	Tracking of Cloud Operations while enhancing privacy and availability (ProvChain)	Cloud Data	Tierion API on Bitcoin	DataRecord with DateTime, Username, Filename, AffectedUser and Action, e.g. File Creation, File Modification, File Copy		DataRecord is published to Blockchain according to Chainpoint standard, which combine the hashes of the DataRecord Elements to a Merkle Tree
Markovic et al	2019	Managing and Reasoning about provenance compliance records	Food Delivery	Hyperledger Fabric	FS-PROV (Food Safety)		PROV Data is stored in Smart Contracts; Smart Contract functions are exposed via a RESTful API
Fadhel et al	2019	model Attacks towards IoT Devices	IoT Security	Hyperledger Fabric	PROV-N Modification		Store Documents in HF Keystore (key-value pairs)
Hogan and Helfert	2019	Can Distributed Ledger	Provenance		W3C PROV		Mapping of PROV Components to Blockchain Elements

		Technology apply the core primary PROV attributes					
Ramachandran und Kantarcioglu	2018	collection and verification of data provenance	Knowledge Management	Ethereum	Open Provenance Model (OPM) Triplet of agents, artifacts, and process, e.g. (user, file: old version, file: new version, process used for modifications).		Document Tracker, that log change event data and store included OPM data in a DB
Ioni und Pahl	2018	track identities and provenance of orchestration decisions of a business network	Edge Computing	Hyperledger Fabric		Relevant components get mapped to W3C Prov Elements - Assets (Data, Sensors, Devices) are considered as Entites - Actions like node joining, identity check, record provenance are Activitys - Initiators of Activites or Owners of Entities are the Agents	Prov Components are stored as classes in Smart Contracts

						Prov is automatically generated, which is shown for for DataAsset Hash of newly generated DataAsset is used to construct a DataAsset Object, which is linked to old DataAsset by derivedFrom attribute.	
Song et al	2020	Tracking and sharing of Provenance Data	Data Provenance		W3C Prov Standard		- Prov related data like person's identity signature, the time and type of operation are stored in the body of the blockchain - When prov data is changed, the digested hash in the header should change too, which make tamper-detection easy

References

[1] Manav Gupta. *Blockchain for dummies*. John Wiley & Sons, Inc., Hoboken, NJ, 2nd ibm limited edition edition, 2018. ISBN 978-1-119-54593-4. OCLC: 1066258248.

[2] PROV-Overview. An Overview of the PROV Family of Documents. Project report, April 2013. URL https://eprints.soton.ac.uk/356854/.

[3] W3C. World Wide Web Consortium (W3C). https://www.w3.org/. [Accessed 10-Apr-2023].

[4] Oxford English Dictionary. Definition of Provenance. https://www.oxfordlearnersdictionaries.com/definition/english/provenance?q=provenance. [Accessed 11-Apr-2023].

[5] Jeff Jarvis. The Importance of Provenance, Jun 2010. URL https://buzzmachine.com/2010/06/27/the-importance-of-provenance/.

[6] Luc Moreau, Juliana Freire, Joe Futrelle, Robert E. McGrath, Jim Myers, and Patrick Paulson. The Open Provenance Model: An Overview. In Juliana Freire, David Koop, and Luc Moreau, editors, *Provenance and Annotation of Data and Processes*, pages 323–326, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. ISBN 978-3-540-89965-5.

[7] Luc Moreau and Paolo Missier. PROV-DM: The PROV Data Model. URL https://www.w3.org/TR/2013/REC-prov-dm-20130430/.

[8] Satya Sahoo Timothy Lebo and Deborah McGuinness. PROV-O: The PROV Ontology, 2013. URL https://www.w3.org/TR/2013/REC-prov-o-20130430/.

[9] Luc Moreau and Paolo Missier. PROV-N: The Provenance Notation, 2013. URL https://www.w3.org/TR/2013/REC-prov-n-20130430/.

[10] Amir Sezavar Keshavarz Danius T. Michaelides Huanjia Yang Trung Dong Huynh, Michael O. Jewell and Luc Moreau. The PROV-JSON Serialization, 2013. URL https://www.w3.org/Submission/prov-json/.

[11] Stian Soiland-Reyes. W3C PROV Data Model.svg - Wikimedia Commons. https://commons.wikimedia.org/wiki/File:W3C_PROV_Data_Model.svg. [Accessed 11-Apr-2023].

[12] Stuart Haber and W. Scott Stornetta. How to time-stamp a digital document. *Journal of Cryptology*, 3(2):99–111, January 1991. doi: 10.1007/bf00196791. URL https://doi.org/10.1007/bf00196791.

[13] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. May 2009. URL http://www.bitcoin.org/bitcoin.pdf.

[14] Liang Cai, Qilei Li, and Xiubo Liang. *Advanced Blockchain Technology: Frameworks and Enterprise-Level Practices*. Springer Nature Singapore, Singapore, 2022. ISBN 978-981-19359-5-4 978-981-19359-6-1. doi: 10.1007/978-981-19-3596-1. URL https://link.springer.com/10.1007/978-981-19-3596-1.

[15] Nil Kalyani. Phases of Evolution of Blockchain. https://www.geeksforgeeks.org/phases-of-evolution-of-blockchain/, 2022. [Accessed 13-Apr-2023].

[16] Pratyusa Mukherjee and Chittaranjan Pradhan. Blockchain 1.0 to Blockchain 4.0—The Evolutionary Transformation of Blockchain Technology. In Sandeep Kumar Panda, Ajay Kumar Jena, Santosh Kumar Swain, and Suresh Chandra Satapathy, editors, *Blockchain Technology: Applications and Challenges*, volume 203, pages 29–49. Springer International Publishing, Cham, 2021. ISBN 978-3-030-69394-7 978-3-030-69395-4. doi: 10.1007/978-3-030-69395-4_3. URL https://link.springer.com/10.1007/978-3-030-69395-4_3. Series Title: Intelligent Systems Reference Library.

[17] Usman W. Chohan. The double spending problem and cryptocurrencies. *SSRN Electronic Journal*, 2017. doi: 10.2139/ssrn.3090174. URL https://doi.org/10.2139/ssrn.3090174.

[18] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, jul 1982. ISSN 0164-0925. doi: 10.1145/357172.357176. URL https://doi.org/10.1145/357172.357176.

[19] Billy Markus and Jackson Palmer. Dogechain Whitepaper, 2013. URL https://dogechain.dog/DogechainWP.pdf.

[20] Charlie Lee. Litecoin (LTC) Whitepaper, Nov 2011. URL https://cryptoverze.com/litecoin-whitepaper/.

[21] Nicolas van Saberhagen. Monero Whitepaper, 2014. URL https://github.com/monero-project/research-lab/blob/master/whitepaper/whitepaper.pdf.

[22] Vitalik Buterin. Ethereum WhitePaper: A Next Generation Smart Contract & Decentralized Application Platform. 2013. URL https://github.com/ethereum/wiki/wiki/White-Paper.

[23] Serguei Popov. The Tangle. IOTA. *Whitepaper*, 1(3):30, 2018.

[24] Colin LeMahieu. Nano: A feeless distributed cryptocurrency network. *Nano [Online resource]. URL: https://nano. org/en/whitepaper (date of access: 24.03. 2018)*, 4, 2018.

[25] Qin Wang, Jiangshan Yu, Shiping Chen, and Yang Xiang. Sok: Diving into dag-based blockchain systems, 2020. URL https://arxiv.org/abs/2012.06128.

[26] Stefan Schmidt, Marten Jung, Thomas Schmidt, Ingo Sterzinger, Günter Schmidt, Moritz Gomm, Klaus Tschirschke, Tapio Reisinger, Fabian Schlarb, Daniel Benkenstein, et al. Unibright-the unified framework for blockchain based business integration. *White paper, April*, 2018.

[27] Avadesian Xu Thomas Pang Tim Yang Maolin Zheng Luke Zeng, Shawn Xin. Seele. https://seele.pro/, 2019. [Accessed 09-Apr-2023].

[28] Zibin Zheng, Shaoan Xie, Hongning Dai, Xiangping Chen, and Huaimin Wang. An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends. In *2017 IEEE International Congress on Big Data (BigData Congress)*, pages 557–564, Honolulu, HI, USA, June 2017. IEEE. ISBN 978-1-5386-1996-4. doi: 10.1109/BigDataCongress.2017.85. URL http://ieeexplore.ieee.org/document/8029379/.

[29] Ying-Chang Liang. Blockchain for Dynamic Spectrum Management. In *Dynamic Spectrum Management*, pages 121–146. Springer Singapore, Singapore, 2020. ISBN 9789811507755 9789811507762. doi: 10.1007/978-981-15-0776-2_5. URL http://link.springer.com/10.1007/978-981-15-0776-2_5. Series Title: Signals and Communication Technology.

[30] Imran Bashir. *Mastering blockchain: a deep dive into distributed ledgers, consensus protocols, smart contracts, DApps, cryptocurrencies, Ethereum, and more*. Expert insight. Packt, Birmingham Mumbai, third edition edition, 2020. ISBN 978-1-83921-319-9.

[31] Melanie Mitchell. Why AI is harder than we think. In *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, June 2021. doi: 10.1145/3449639.3465421. URL https://doi.org/10.1145/3449639.3465421.

[32] Ragib Hasan, Joseph Tucek, Paul Stanton, William Yurcik, Larry Brumbaugh, Jeff Rosendale, and Roelof Boonstra. The techniques and challenges of immutable storage with applications in multimedia. pages 41–52, San Jose, CA, January 2005. doi: 10.1117/12.588103. URL http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid=858272.

[33] Elizabeth Haubert, Joseph Tucek, Larry Brumbaugh, and William Yurcik. Tamper-resistant storage techniques for multimedia systems. pages 30–40, San Jose, CA, January 2005. doi: 10.1117/12.588020. URL http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid=858271.

[34] John P Conley et al. *Encryption, Hashing, PPK, and Blockchain: A Simple Introduction*. Vanderbilt University, Department of Economics, 2019.

[35] Wahome Macharia. Cryptographic Hash Functions. 05 2021. URL https://www.researchgate.net/publication/351837904_Cryptographic_Hash_Functions.

[36] Rohit Verma and Aman Kumar Sharma. Cryptography: Avalanche effect of AES and RSA. *International Journal of Scientific and Research Publications (IJSRP)*, 10(4):p10013, April 2020. doi: 10.29322/ijsrp.10.04.2020.p10013. URL https://doi.org/10.29322/ijsrp.10.04.2020.p10013.

[37] Wouter Penard and Tim van Werkhoven. On the secure hash algorithm family. *Cryptography in context*, pages 1–18, 2008.

[38] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

[39] Ali Sunyaev. Distributed ledger technology. In *Internet Computing*, pages 265–299. Springer International Publishing, 2020. doi: 10.1007/978-3-030-34957-8_9. URL https://doi.org/10.1007/978-3-030-34957-8_9.

[40] Types of blockchain: Public, private, or something in between: Foley amp; lardner llp. URL https://www.foley.com/en/insights/publications/2021/08/types-of-blockchain-public-private-between.

[41] Aras Bozkurt and Hasan Ucar. Blockchain technology as a bridging infrastructure among formal, non-formal, and informal learning processes. In *Blockchain Technology Applications in Education*, pages 1–15. IGI Global, 2020. doi: 10.4018/978-1-5225-9478-9.ch001. URL https://doi.org/10.4018/978-1-5225-9478-9.ch001.

[42] Gleidson Sobreira Leite, Adriano Bessa Albuquerque, and Plácido Rogerio Pinheiro. Process automation and blockchain in intelligence and investigation units: An approach. *Applied Sciences*, 10(11):3677, May 2020. doi: 10.3390/app10113677. URL https://doi.org/10.3390/app10113677.

[43] 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*, 14:352, 10 2018. doi: 10.1504/IJWGS.2018.095647.

[44] Andreas M. Antonopoulos. *Mastering Bitcoin: Programming the Open Blockchain*. O'Reilly Media, Inc., 2nd edition, 2017. ISBN 1491954388.

[45] Jimi S. Blockchain explained: how a 51blog.goodaudience.com. https://blog.goodaudience.com/what-is-a-51-attack-or-double-spend-attack-aa108db63474. [Accessed 09-Apr-2023].

[46] P. Rajitha Nair and D. Ramya Dorai. Evaluation of Performance and Security of Proof of Work and Proof of Stake using Blockchain. In *2021 Third International Conference on Intelligent Communication Technologies and Virtual Mobile Networks (ICICV)*, pages 279–283, Tirunelveli, India, February 2021. IEEE. ISBN 978-1-66541-960-4. doi: 10.1109/ICICV50876.2021.9388487. URL https://ieeexplore.ieee.org/document/9388487/.

[47] Dylan Yaga, Peter Mell, Nik Roby, and Karen Scarfone. Blockchain technology overview. Technical report, oct 2018. URL https://doi.org/10.6028%2Fnist.ir.8202.

[48] Solidity. Layout of a solidity source file, 2023. URL https://docs.soliditylang.org/en/v0.6.8/layout-of-source-files.html.

[49] Solidity. First application, 2023. URL https://solidity-by-example.org/first-app/.

[50] Corwin Smith. Transactions — ethereum.org — ethereum.org. https://ethereum.org/en/developers/docs/transactions/. [Accessed 12-Apr-2023].

[51] Barbara Ann Kitchenham and Stuart Charters. Guidelines for performing systematic literature reviews in software engineering. Technical Report EBSE 2007-001, Keele University and Durham University Joint Report, 07 2007. URL https://www.elsevier.com/__data/promis_misc/525444systematicreviewsguide.pdf.

[52] Tran Khanh Dang and Thu Duong Anh. A Pragmatic Blockchain Based Solution for Managing Provenance and Characteristics in the Open Data Context. In Tran Khanh Dang, Josef Küng, Makoto Takizawa, and Tai M. Chung, editors, *Future Data and Security Engineering*, volume 12466, pages 221–242. Springer International Publishing, Cham, 2020. ISBN 978-3-030-63923-5 978-3-030-63924-2. doi: 10.1007/978-3-030-63924-2_13. URL https://link.springer.com/10.1007/978-3-030-63924-2_13. Series Title: Lecture Notes in Computer Science.

[53] T.K. Dang and T.A. Duong. An effective and elastic blockchain-based provenance preserving solution for the open data. *International Journal of Web Information Systems*, 17 (5):480–515, 2021. doi: 10.1108/IJWIS-03-2021-0029. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85108595981&doi=10.1108%2fIJWIS-03-2021-0029&partnerID=40&md5=93aa7dd534ee37139c9e4ff15a482c7c.

[54] A. Demichev, A. Kryukov, and N. Prikhod'ko. Business Process Engineering for Data Storing and Processing in a Collaborative Distributed Environment Based on Provenance Metadata, Smart Contracts and Blockchain Technology. *Journal of Grid Computing*, 19 (1), 2021. doi: 10.1007/s10723-021-09544-4. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85100231592&doi=10.1007%2fs10723-021-09544-4&partnerID=40&md5=7b54f4c37b83305dd467e6b511e4fc05.

[55] F. Kirstein. A decentralized provenance network for linked open data. volume 2548, pages 104–115, 2019. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85081551411&partnerID=40&md5=f42a3ab59a0b883122a49018ccf0dc77.

[56] G. Hogan and M. Helfert. Transparent cloud privacy: Data provenance expression in blockchain. pages 430–436, 2019. doi: 10.5220/0007733404300436. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85067421602&doi=10.5220%2f0007733404300436&partnerID=40&md5=e3699248d3a464aae1f16b77a212e707.

[57] N.E. Ioini and C. Pahl. Trustworthy orchestration of container based edge computing using permissioned blockchain. pages 147–154, 2018. doi: 10.1109/IoTSMS.2018.8554470. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85055940674&doi=10.1109%2fIoTSMS.2018.8554470&partnerID=40&md5=30613f6e2ce675a5ca3b8a9d21efb65c.

[58] N. Fadhel, F. Lombardi, L. Aniello, A. Margheri, and V. Sassone. Towards a semantic modelling for threat analysis of IoT applications: A case study on transactive energy. volume 2019, 2019. doi: 10.1049/cp.2019.0147. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85081958560&doi=10.1049%2fcp.2019.0147&partnerID=40&md5=6c7a5ba6971229612b354d0e7cbdc677. Issue: CP756.

[59] M. Markovic, P. Edwards, and N. Jacobs. Recording Provenance of Food Delivery Using IoT, Semantics and Business Blockchain Networks. pages 116–118, 2019. doi: 10.1109/IOTSMS48152.2019.8939250. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85077958179&doi=10.1109%2fIOTSMS48152.2019.8939250&partnerID=40&md5=c9f457afb985c0e9678e22d9c458eb43.

[60] M. Markovic, N. Jacobs, K. Dryja, P. Edwards, and N.J.C. Strachan. Integrating Internet of Things, Provenance, and Blockchain to Enhance Trust in Last Mile Food Deliveries. *Frontiers in Sustainable Food Systems*, 4, 2020. doi: 10.3389/fsufs.2020.563424. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85097092142&doi=10.3389%2ffsufs.2020.563424&partnerID=40&md5=6ce8b9287e5c02c095063bddc351327d.

[61] Zhang Song, Li Yang, Li Gaoyang, Yu Han, Hao Baozhong, Song Jinwei, and Fan Jingang. An Improved Data Provenance Framework Integrating Blockchain and PROV Model. In *2020 International Conference on Computer Science and Management Technology (ICCSMT)*, pages 323–327, Shanghai, China, November 2020. IEEE. ISBN 978-1-72818-668-9. doi: 10.1109/ICCSMT51754.2020.00073. URL https://ieeexplore.ieee.org/document/9443825/.

[62] Andrea Margheri, Massimiliano Masi, Abdallah Miladi, Vladimiro Sassone, and Jason Rosenzweig. Decentralised provenance for healthcare data. *International Journal of Medical Informatics*, 141:104197, September 2020. ISSN 13865056. doi: 10.1016/j.ijmedinf.2020.104197. URL https://linkinghub.elsevier.com/retrieve/pii/S1386505619312031.

[63] F. Lautert, D.F. Pigatto, and L. Gomes. A fog architecture for privacy-preserving data provenance using blockchains. volume 2020-July, 2020. doi: 10.1109/ISCC50000.2020.9219724. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85094103918&doi=10.1109%2fISCC50000.2020.9219724&partnerID=40&md5=a8033b5d770c0e89a4e36e2c4772604c.

[64] M. Sigwart, M. Borkowski, M. Peise, S. Schulte, and S. Tai. A secure and extensible blockchain-based data provenance framework for the Internet of Things. *Personal and Ubiquitous Computing*, 2020. doi: 10.1007/s00779-020-01417-z. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85086591225&doi=10.1007%2fs00779-020-01417-z&partnerID=40&md5=38ba363caddeb05a2e693f2dd186b6cb.

[65] Zhoujie Zhang, Can Cui, Lei Tao, Jiaqi Wang, Dapeng Li, Shuzhou Wu, Qiong Feng, Qingbo Yang, Jian Chen, Jie Zhang, Peng Zhang, and Zhijun Zhang. Research on Consistency Tracing Technology of Dispatching Control Model Data based on Blockchain. In *2021 IEEE 5th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, pages 234–239, Chongqing, China, March 2021. IEEE. ISBN 978-1-72818-028-1. doi: 10.1109/IAEAC50856.2021.9390682. URL https://ieeexplore.ieee.org/document/9390682/.

[66] R.P. Jagadeesh Chandra Bose, Kanchanjot Kaur Phokela, Vikrant Kaulgud, and Sanjay Podder. BLINKER: A Blockchain-Enabled Framework for Software Provenance. In *2019 26th Asia-Pacific Software Engineering Conference (APSEC)*, pages 1–8, Putrajaya, Malaysia, December 2019. IEEE. ISBN 978-1-72814-648-5. doi: 10.1109/APSEC48747.2019.00010. URL https://ieeexplore.ieee.org/document/8945702/.

[67] Lian-Shan Sun, Xue Bai, Chao Zhang, Yang Li, Yong-Bin Zhang, and Wen-Qiang Guo. BSTProv: Blockchain-Based Secure and Trustworthy Data Provenance Sharing. *Electronics*, 11(9):1489, May 2022. ISSN 2079-9292. doi: 10.3390/electronics11091489. URL https://www.mdpi.com/2079-9292/11/9/1489.

[68] Xueping Liang, Sachin Shetty, Deepak Tosh, Charles Kamhoua, Kevin Kwiat, and Laurent Njilla. ProvChain: A Blockchain-Based Data Provenance Architecture in Cloud Environment with Enhanced Privacy and Availability. In *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 468–477, Madrid, Spain, May 2017. IEEE. ISBN 978-1-5090-6611-7. doi: 10.1109/CCGRID.2017.8. URL http://ieeexplore.ieee.org/document/7973733/.

[69] Aravind Ramachandran and Murat Kantarcioglu. SmartProvenance: A Distributed, Blockchain Based DataProvenance System. In *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*, pages 35–42, Tempe AZ USA, March 2018. ACM. ISBN 978-1-4503-5632-9. doi: 10.1145/3176258.3176333. URL https://dl.acm.org/doi/10.1145/3176258.3176333.

[70] P. Tunstad, A.M. Khan, and P.H. Ha. HyperProv: Decentralized resilient data provenance at the edge with blockchains. pages 3–4, 2019. doi: 10.1145/3366627.3368105. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85078900202&doi=10.1145%2f3366627.3368105&partnerID=40&md5=7820c2cd27faa8035909a0e4b888d116.

[71] R. Coelho, R. Braga, J.M.N. David, M. Dantas, V. Ströele, and F. Campos. Integrating blockchain for data sharing and collaboration support in scientific ecosystem platform. volume 2020-January, pages 264–273, 2021. doi: 10.24251/HICSS.2021.031. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85108325612&partnerID=40&md5=4b62e8ca837adcedc936cb6a2a8691e0.

[72] R. Coelho, R. Braga, J.M.N. David, V. Stroele, F. Campos, and M. Dantas. A Blockchain-Based Architecture for Trust in Collaborative Scientific Experimentation. *Journal of Grid Computing*, 20(4), 2022. doi: 10.1007/s10723-022-09626-x. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85139762810&doi=10.1007%2fs10723-022-09626-x&partnerID=40&md5=e8f8b3013b128486da2c5694695c441a.

[73] Klaus Pohl and Chris Rupp. *Basiswissen Requirements Engineering: Aus- und Weiterbildung nach IREB-Standard zum Certified Professional for Requirements Engineering Foundation Level*. dpunkt.verlag, Heidelberg, 5., überarbeitete und aktualisierte auflage edition, 2021. ISBN 978-3-86490-814-9.

[74] Klaus Pohl and Chris Rupp. *Requirements engineering fundamentals: a study guide for the certified professional for requirements engineering exam, foundation level, IREB compliant*. Rocky Nook, Santa Barbara, CA, second edition edition, 2015. ISBN 978-1-937538-77-4.

[75] Sophisten and Christine Rupp. *Requirements-Engineering: Die kleine RE-Fibel*. Hanser, Carl, München, 2015. ISBN 978-3-446-44450-8.

[76] IEEE Computer Society. IEEE Recommended Practice for Software Requirements Specifications. *IEEE Std 830-1998*, pages 1–40, 1998. doi: 10.1109/IEEESTD.1998.88286.

[77] James Robertson and Suzanne Robertson. Volere. *Requirements Specification Templates*, 2000.

[78] R. W. Hollmann, A. Schäfer, O. Bertram, and M. Rädel. Virtual testing of multifunctional moveable actuation systems. *CEAS Aeronautical Journal*, 13(4):979–988, October 2022. ISSN 1869-5582, 1869-5590. doi: 10.1007/s13272-022-00602-5. URL https://link.springer.com/10.1007/s13272-022-00602-5.

[79] Deutsches Zentrum für Luft-und Raumfahrt e.V. (DLR). RCE Environment. https://github.com/rcenvironment/rce, 2023. [Accessed 24-Feb-2023].

[80] Deutsches Zentrum für Luft-und Raumfahrt e.V. (DLR). RCE Environment. Official Website, 2023. URL https://rcenvironment.de/.

[81] Deutsches Zentrum für Luft-und Raumfahrt e.V. (DLR). RCE Environment Screenshots. https://rcenvironment.de/pages/screenshots.html, 2023. [Accessed 13-Apr-2023].

[82] Mark von Rosing, Stephen White, Fred Cummins, and Henk de Man. Business Process Model and Notation—BPMN. In *The Complete Business Process Handbook*, pages 433–457. Elsevier, 2015. ISBN 978-0-12-799959-3. doi: 10.1016/B978-0-12-799959-3.00021-5. URL https://linkinghub.elsevier.com/retrieve/pii/B9780127999593000215.

[83] Statista. Vertrauen in die öffentliche Verwaltung in Deutschland 2022 — Statista — de.statista.com. https://de.statista.com/statistik/daten/studie/795828/umfrage/umfrage-in-deutschland-zum-vertrauen-in-die-oeffentliche-verwaltung/. [Accessed 13-Apr-2023].

[84] Cyberphone. JSON-Canonicalization Scheme (JCS), 2019. URL https://github.com/cyberphone/json-canonicalization.

[85] Friederike Kleinfercher, Sandra Vengadasalam, and James Lawton. Bloxberg - The Blockchain for Science. Whitepaper 2.0. Technical report, Bloxberg, 01 2022.

[86] OpenEthereum. Authority Round (Aura) Documentation. https://openethereum.github.io/Aura.html. [Accessed 13-Apr-2023].

[87] OpenZeppelin. AccessControl.sol. https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/AccessControl.sol, 2023. [Accessed 12-Apr-2023].

[88] OpenZeppelin. Ownable.sol. https://github.com/OpenZeppelin/openzeppelin-contracts/blob/release-v2.5.0/contracts/ownership/Ownable.sol, 2023. [Accessed 12-Apr-2023].

[89] TechHQ. Whitelist.sol. https://github.com/HQ20/contracts/blob/v0.0.2/contracts/access/Whitelist.sol, 2023. [Accessed 12-Apr-2023].

[90] TruffleSuite. Ethereum Workspace Overview. https://trufflesuite.com/docs/ganache/concepts/ethereum-workspace/overview/. [Accessed 13-Apr-2023].

[91] Brownie. The Configuration File. URL https://eth-brownie.readthedocs.io/en/stable/config.html.

[92] Web3.py. Documentation. https://web3py.readthedocs.io/en/stable/, 2023. [Accessed 13-Apr-2023].

[93] Steven Kocadag. Evaluation Testfiles. https://drive.google.com/drive/folders/1pWvaS9T-yxnp8cXsUjbMy39GdmC9kDnw?usp=share_link, 2023. [Accessed 13-Apr-2023].

[94] ExtendsClass. Random JSON Data Generator. https://extendsclass.com/json-generator.html, 2020. [Accessed 13-Apr-2023].

[95] F. Amin, R. Abbasi, A. Mateen, M. Ali Abid, and S. Khan. A Step toward Next-Generation Advancements in the Internet of Things Technologies. *Sensors*, 22(20), 2022. doi: 10.3390/s22208072. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85140651210&doi=10.3390%2fs22208072&partnerID=40&md5=b16034f306ecc37293fa1c1ef100c7e5.

[96] K. Boland, P. Fafalios, A. Tchechmedjiev, K. Todorov, and S. Dietze. Modeling and contextualizing claims. volume 2599, 2019. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85093850014&partnerID=40&md5=d0674c42db2a5fbedef89a760d48da94.

[97] Zhiqiang Geng, Yuan Cao, Jun Li, and Yongming Han. Novel blockchain transaction provenance model with graph attention mechanism. *Expert Systems with Applications*, 209:118411, December 2022. ISSN 0957-4174. doi: 10.1016/j.eswa.2022.118411. URL https://www.sciencedirect.com/science/article/pii/S0957417422015172.

[98] N. Gouru and N. Vadlamani. DistProv-Data Provenance in Distributed Cloud for Secure Transfer of Digital Assets with Ethereum Blockchain using ZKP. *International Journal of Open Source Software and Processes*, 10(3):1–18, 2019. doi: 10.4018/IJOSSP.2019070101. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85072635165&doi=10.4018%2fIJOSSP.2019070101&partnerID=40&md5=5b8b5e672884802ea6a33c65b978beba.

[99] Mahmud Hasan, Kemafor Ogan, and Binil Starly. Hybrid Blockchain Architecture for Cloud Manufacturing-as-a-service (CMaaS) Platforms with Improved Data Storage and Transaction Efficiency. *49th SME North American Manufacturing Research Conference (NAMRC 49, 2021)*, 53:594–605, January 2021. ISSN 2351-9789. doi: 10.1016/j.promfg.2021.06.060. URL https://www.sciencedirect.com/science/article/pii/S2351978921000706.

[100] Oludare Isaac Abiodun, Moatsum Alawida, Abiodun Esther Omolara, and Abdulatif Alabdulatif. Data provenance for cloud forensic investigations, security, challenges, solutions and future perspectives: A survey. *Journal of King Saud University - Computer and Information Sciences*, October 2022. ISSN 1319-1578. doi: 10.1016/j.jksuci.2022.10.018. URL https://www.sciencedirect.com/science/article/pii/S131915782200369X.

[101] N. Kaaniche, S. Belguith, M. Laurent, A. Gehani, and G. Russello. Prov-Trust: Towards a trustworthy sgx-based data provenance system. volume 3, pages 225–237, 2020. doi: 10.5220/0009889302250237. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85111138820&doi=10.5220%2f0009889302250237&partnerID=40&md5=4f223efed3ebec4e32f0645d28e6490f.

[102] E. Kak, R. Orji, J. Pry, K. Sofranko, R.K. Lomotey, and R. Deters. Privacy improvement architecture for IoT. pages 148–155, 2018. doi: 10.1109/ICIOT.2018.00028. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85055619938&doi=10.1109%2fICIOT.2018.00028&partnerID=40&md5=b80d7c2aa5a4be19df06fda98aebf3d3.

[103] T. Komamizu, Y. Uchida, Y. Ogawa, and K. Toyama. Analyzing Japanese law history through modeling multi-versioned entity. volume 2599, 2019. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85093833630&partnerID=40&md5=d9139f630fe8fb2fb510b2fcda6fef66.

[104] H. Li, P. Yue, L. Jiang, M. Zhang, and Z. Liang. Blockchain technology for vector geographic provenance information organization and verification. *Cehui Xuebao/Acta Geodaetica et Cartographica Sinica*, 50(6):823–832, 2021. doi: 10.11947/j.AGCS.2021.20200168. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85109074412&doi=10.11947%2fj.AGCS.2021.20200168&partnerID=40&md5=9db70528b49b7a6bb31a8341b30ba9ad.

[105] Shlomi Linoy, Suprio Ray, and Natalia Stakhanova. EtherProv: Provenance-Aware Detection, Analysis, and Mitigation of Ethereum Smart Contract Security Issues. In *2021 IEEE International Conference on Blockchain (Blockchain)*, pages 1–10, Melbourne, Australia, December 2021. IEEE. ISBN 978-1-66541-760-0. doi: 10.1109/Blockchain53845.2021.00014. URL https://ieeexplore.ieee.org/document/9680507/.

[106] J.R. Zayas, E. O'Neill, M.A. Seale, A. Ruvinsky, and O. Eslinger. An Integrated Blockchain Approach for Provenance of Rotorcraft Maintenance Data. 2020. doi: 10.1109/AERO47225.2020.9172700. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85092548167&doi=10.1109%2fAERO47225.2020.9172700&partnerID=40&md5=a6082d23e228cdeaa237404664cc54a2.

[107] G. Zhang and Y. Mao. Blockchain-based decentralized data provenance method. *Nanjing Youdian Daxue Xuebao (Ziran Kexue Ban)/Journal of Nanjing University of Posts and Telecommunications (Natural Science)*, 39(2):91–98, 2019. doi: 10.14132/j.cnki.1673-5439.2019.02.014. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85068585930&doi=10.14132%2fj.cnki.1673-5439.2019.02.014&partnerID=40&md5=2e70c2e67c62b1a45db81068b383bd1d.

[108] X. Zhang, J. Feng, Z. Yin, and J. Lin. Trusted Query Method for Data Provenance Based on Blockchain. *Yingyong Kexue Xuebao/Journal of Applied Sciences*, 39(1):42–54, 2021. doi: 10.3969/j.issn.0255-8297.2021.01.004. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85101347011&doi=10.3969%2fj.issn.0255-8297.2021.01.004&partnerID=40&md5=daef31f345b13cf391d0eaea8ec74753.

List of Figures

1 PROV-DM Overview of Core Types and Relations. Sources: [7] [11] 4

2 Phases of Evolution of Blockchain. Source: [15] 7
3 Sequence of Blocks of a Blockchain. Source: [29] 10
4 Public, Private and Hybrid Blockchains. Source: [40] 11
5 Smart Contract of a Counter Application `counter.sol`. Source: [49] 17

6 VPH Process 29
7 Graphical User Interface of RCE. Source: [81] 29
9 Mixed Documentation Model: Use Case Diagram and Requirements 34

10 Architecture of the System 36

12 Ganache User Interface. Source: [90] 51
13 Blockchain after Smart Contract Deployment 54
14 Header and Body of Block 2 54
15 Error Message: Invalid PROV Documents 57

16 Class Diagram of the Smart Contract `prov.sol` 71

List of Tables

1 Comparison between Public, Hybrid and Private Blockchains. Own representation based on: [41] [29] [43] [42] 13

2 Search Results of Queries among Databases 19
3 Overview of PROV Storage Strategies 23
4 Comparison of Storage Methods: On-Chain vs Off-Chain 24
5 Comparison of Storage Methods: Document-based and Element-based 25

6 List of Interview Questions 27

7 Ganache Test Environment Accounts 52
8 Evaluation Scenarios for Check Functions 55

9 PROV-DM Types and Relations. Own Representation based on: [7] 63

List of Listings

7.1 Access Control related Attributes of the Contract . . . 43
7.2 Methods for granting Roles . . . 44
7.3 Definition of Workflow and Node contract attributes . . . 44
7.4 Definition of `add_workflow_and_nodes` function . . . 45
7.5 Definition of `check_workflow` and `check_nodes` functions . . . 45
7.6 Definition of `check_workflow_and_nodes` function . . . 46
7.7 Interaction with the Contract . . . 47
7.8 Definition of `upload_files` method . . . 49

8.1 Deployment of Smart Contract . . . 53
8.2 Obtaining stored Workflows and Nodes . . . 55
8.3 Testing Check Functions of the Smart Contract . . . 55
8.4 Add Users to both Parties . . . 56
8.5 Determination of a Documents Uploader and Time . . . 58

1 PROV Document of a Workflow . . . 65
2 Smart Contract `prov.sol` . . . 68
3 Upload Page of GUI `UploadPage.py` . . . 71
4 Webprovider Class `WebProvider.py` . . . 75
5 Upload of Nodes and Workflows . . . 76