FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Decentralised Data Provenance Based on the Blockchain

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering und Internet Computing

eingereicht von

Ing. Svetoslav Videnov, BSc
Matrikelnummer 1025844

an der Fakultät für Informatik
der Technischen Universität Wien

Betreuung: Dr.-Ing. Stefan Schulte, BSc, Dipl.-Oec.

Wien, 1. März 2019

Svetoslav Videnov	Stefan Schulte

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Decentralised Data Provenance Based on the Blockchain

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering and Internet Computing

by

Ing. Svetoslav Videnov, BSc
Registration Number 1025844

to the Faculty of Informatics

at the TU Wien

Advisor: Dr.-Ing. Stefan Schulte, BSc, Dipl.-Oec.

Vienna, 1st March, 2019

Svetoslav Videnov	Stefan Schulte

Erklärung zur Verfassung der Arbeit

Ing. Svetoslav Videnov, BSc
St.-Antonius-Str. 40, 6890 Lustenau, Austria

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 1. März 2019

Svetoslav Videnov

Danksagung

An erster Stelle möchte ich meinem Betreuer Herrn Dr.-Ing. Stefan Schulte danken für seine unermüdliche Geduld mit meiner Arbeit, seiner humorvollen Aufnahme meiner Zeitplanung, und vor allem seiner stetigen Bereitschaft mich fachlich und auch wissenschaftlich zu unterstützen. Ohne seine engagierte Betreuung wäre diese Arbeit nicht in der vorliegenden Qualität zustande gekommen.

Weiters möchte ich allen meinen Arbeitskollegen, und insbesondere Michael Borkowski, für anregende, fachliche und nicht fachliche, Diskussionen und einen tiefgehenden Einblick in das wissenschaftliche Arbeiten, danken. Ihre Bereitschaft, zu jeder Zeit Erfahrungen und Empfehlungen mit mir zu teilen war eine wahre Hilfe für meine Zeit am Institut.

An dieser Stelle möchte ich auch meinen besonderen Dank an die Damen aus dem Sekretariat der DSG aussprechen. Renate Weiss, Margret Steinbuch und Christine Kamper waren allzeit bereit mir bei organisatorischen Problemen zu helfen um diese möglichst rasch aus dem Weg zu räumen.

Des Weiteren möchte ich mich bei der Internet Privatstiftung Austria bedanken, die mir im Rahmen ihres Förderprogramms Netidee ein Stipendium für meine Masterarbeit verliehen haben. Mit ihren Stipendien bieten sie eine wichtige Hilfestellung für die OpenSource-Gemeinschaft in Österreich.

Zu guter Letzt möchte ich auch meiner Familie, meinen Freunden und meiner Partnerin danken. Allen voran meinen Eltern, die mein Studium in erster Linie ermöglicht haben und sich all die Jahre von der Dauer nicht beirren haben lassen. Meinen Freunden und Schwester die mich in regelmäßigen Abständen dazu gebracht haben, Dinge aus den Augen von *Nicht-Informatikern* zu betrachten und meiner Partnerin, Agnes Fastenbauer, die vor allem in den finalen Wochen meiner Arbeit für ein ungestörtes Umfeld gesorgt hat, in dem ich in Ruhe arbeiten konnte.

Acknowledgements

First, I would like to express my gratitude to my supervisor, Dr.-Ing. Stefan Schulte, for his seemingly endless patience and his humorous taking of my time frames. But, foremost I want to thank him for his technical and scientific support. Without his expertise, this thesis would not be present in its current quality.

Next, I want to thank my colleagues, especially Michael Borkowski, for interesting and funny discussions which often but not always revolved around the work we were doing. They made it possible for me to get a glimpse at what it means to do scientific work and never fell short to support me with their experience and expertise during my time at the institute.

I also want to give my special thanks to the gorgeous ladies from the office administration. Renate Weiss, Margret Steinbuch and Christine Kamper were always there when organizational obstacles needed to be taken care of.

Furthermore, I want to thank the Internet Foundation Austria who supported my work with a grant as part of their Netidee funding programme. They are providing important support for the open source community in Austria.

Last but not least, I want to thank my family, friends and my partner. First and foremost my gratitude goes to my family who enabled my studies in the first place and were never concerned with the long time it took. Then I want to thank my friends and my sister for helping me regularly to see things from the perspective of a non-information scientist. Finally, I want to thank my partner, Agnes Fastenbauer. Especially in the final weeks of this work, she created a very productive environment for me to be able to finish my work.

Kurzfassung

Das Fachgebiet der Daten-Provenienz beschäftigt sich mit der Herkunft von Daten. Es erlaubt uns, Vertrauen in Daten aufzubauen. Dafür müssen die Provenienz-Daten selber zuverlässig sein und gewisse Sicherheitsstandards erfüllen. Das zu erreichen erweist sich als schwierig, da Daten-Provenienz oft sehr Domänen-abhängig ist, wodurch eine starke Fragmentierung des Fachgebietes entstanden ist.

Diese Fragmentierung macht es schwierig, einheitliche Sicherheitsstandards zu implementieren. Deswegen haben sich auch für die Sicherheitsanforderungen Domänen-spezifische Lösungen entwickelt. Außerdem erschwert die Fragmentierung die Zusammenarbeit zwischen den verschiedenen Domänen. Dies wiederum erschwert es, die Herkunft von Daten nachvollziehbar zu machen was das eigentliche Ziel von Daten-Provenienz ist.

In dieser Arbeit stellen wir eine Blockchain-basierte Lösung vor um Domänen-unabhängige Suchgebiete für Daten-Provenienz zu erstellen. Dies erlaubt es uns, auch Sicherheitseigenschaften der Blockchain auf eine einheitliche Art und Weise auf Daten-Provenienz zu übertragen.

Mithilfe unserer Lösung können wir die Zusammenarbeit zwischem Domänen ermöglichen, ohne diesen einheitliche Provenienz-Eigenschaften aufzwingen zu müssen. Dies erlaubt es, die Herkunft von Daten über Domänengrenzen hinaus nachvollziehbar zu machen und ein vollständigeres Bild der Datenherkunft zu erstellen. Gleichzeitig kreieren wir eine Platform auf der Domänen-unabhängige Sicherheitsanforderungen einheitlich umgesetzt werden können.

Abstract

Data provenance allows to reproduce what has happened to data during its lifecycle, i.e., it allows to build trust in data and decisions. To achieve this, provenance data has to be reliable itself and to fulfill certain security requirements. This proves to be a difficult problem since data provenance often has many domain-specific properties. Addressing these domain specificities has led to a fragmentation of the field of data provenance.

This fragmentation is a problem since it makes it hard to implement common solutions for security requirements. Instead, the field of secure provenance often focuses on providing domain-specific solutions. Furthermore, this fragmentation makes collaboration between domains difficult, hindering the overall goal of making the history of data products reproducible.

In this thesis, we present a blockchain-based approach for creating a domain-independent search space for data provenance. At the same time, we are able to utilize this search space to map strong blockchain-based security properties, in a domain-independent way, to the field of data provenance.

With our solution, we can enable collaboration between domains without enforcing common provenance properties to the domains. This would allow us to track data products across domain borders and create more complete provenance views while providing a platform for common security properties.

Contents

CHAPTER 1
Introduction

In complex, loosely-coupled systems, as depicted in Figure 1.1, it is often hard to reproduce how a certain decision was made or how certain data was generated. In our example, the input data depicted in green passes through a hybrid system of human experts and Web services and produces some output data, depicted in violet. It is hard to build trust into this output data since it is not easily reproducible what happened to the input data and what influence the human experts had on it. Neither is the path which the data took during its lifetime easily traceable. This is a simple example from the domain of Web services [1]. Another example from the domain of hybrid systems could be, why did a certain person get the organ donation and not another [2], and a socio-political example could be the trustful disclosure of decisions for example in automated cars. To solve these issues, data provenance, i.e., a type of metadata, can be used to provide reliable information about those processes, decisions, and outcomes (depicted in orange). By collecting provenance information about the process and the changes that occurred to the input data, we can build trust in the output data. However, one major disadvantage of data provenance solutions is that you have to be able to trust the provenance data itself. This leads back to the initial problem of how to ensure trust in the data that is provided. To solve this, many solutions fall back to centralized approaches [3] or cryptographic techniques [4] but even so, there is some third party involved and trusted not to alter any information.

One way of providing this trust without a trusted third party could come out of the field of cryptocurrencies, i.e., the usage of blockchain technology. The blockchain is simply put a distributed ledger [5]. It allows for storing information publicly and distributed in an unchangeable manner. By utilizing the blockchain, we can move part of the trust issue away from traditional provenance solutions into a technology which is by design envisioned to create trust between parties. This would strengthen the trust in the provenance data which as a consequence would strengthen the trust in the original data product. Finally, this allows for easy and reliable reproduction of data and decisions.

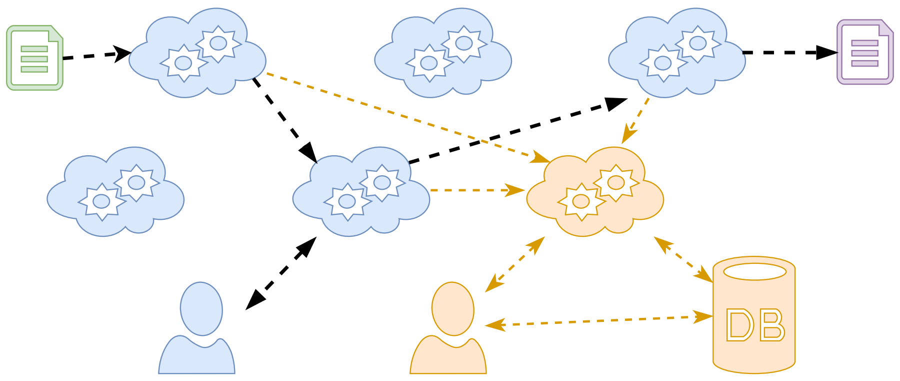


Figure 1.1: An example use case.

1.1 Data Provenance

Many different solutions address the problem of data provenance. They vary in different important aspects, e.g., the domain, granularity, provenance models. It has to be noted that there are major differences between collecting provenance of a process being executed on a personal computer, database queries, or a workflow being executed based on Web services [6]. Database provenance, on the one hand, aims at providing the ability to reconstruct how tables and databases as a whole got to exist in the first place and which other databases or tables are the roots [7]. Provenance in Web services, on the other hand, tries to provide information about which services were used in a certain workflow and what they did to the data [2]. For example, Groth et al. [8] identify four main categories of provenance: fine-granularity provenance, domain-specific provenance, provenance in databases, and middleware-based provenance. These categories are not mutually exclusive and also not the only categorization of data provenance as we will see.

Besides the general properties of data provenance, e.g., the chosen provenance model and granularity, there is also the topic of securing data provenance. When it comes to the security of data provenance, we can identify five main properties, as presented in the literature [4]:

Confidentiality: Confidentiality deals with securing sensible information which can be contained in provenance data.

Integrity: Integrity deals with securing provenance data from alterations during the entire data lifecycle.

Unforgeability: Unforgeability deals with securing the provenance data against forgery.

Non-Repudiation: Non-Repudiation deals with making provenance data undeniable, by users who recorded the data.

Availability: Availability deals with the total time that provenance data is accessible.

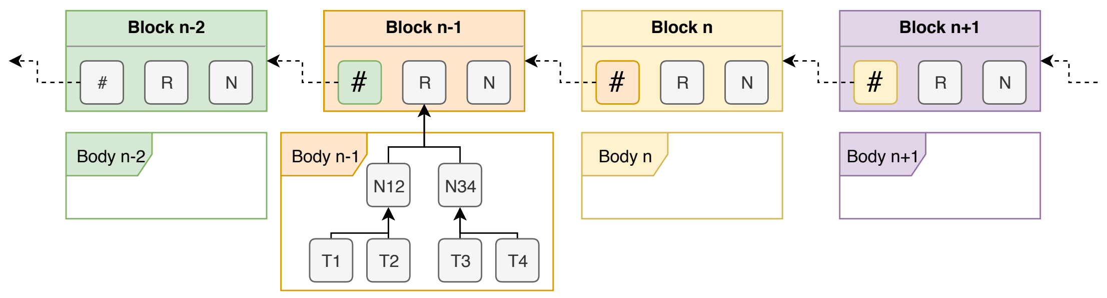


Figure 1.2: Simplified view of the blockchain.

Securing provenance data proves to be a difficult topic that is often left to be handled independently of the general challenges provided by data provenance. For example, the W3C PROV recommendation [9] which defines a provenance model discusses the topic briefly [10] but does not propose any specific solutions. Other work [4] then again focuses solely on securing data provenance in a specific domain but does not consider interoperability between domains.

1.2 Blockchain

The Bitcoin protocol overcomes the need for a centralized trusted store by means of the blockchain [5]. It operates in a secure manner without the need for a centralized, trusted entity, e.g., a bank. This opens up the question of how this technology can be used to provide decentralized security for data provenance.

More formally put, the blockchain is a linked list, as shown in Figure 1.2. Hereby, every block links to the block before by saving the previous blocks' hash [11]. Since the link is created by using the previous blocks' hash once linked a block cannot be changed anymore since this would break the chain. Furthermore, every block contains in its body a set of transactions [11], which can be compared to the single rows in a ledger. The result can be simplified as a digital ledger in which data can be written but not edited or deleted.

We mentioned that the blockchain operates in a secure manner without the need of a trusted entity. More concretely, it uses decentralized proof algorithms to establish a distributed consensus between all the participants in the network [11]. Put in a very simplified manner, as shown in Figure 1.3, instead of giving all the important information to one trusted entity, e.g., a bank, we can give it to everyone and use a consensus mechanism to achieve a common view of the state of this information. The advantage of this approach is that no one single entity is now able to manipulate the data anymore.

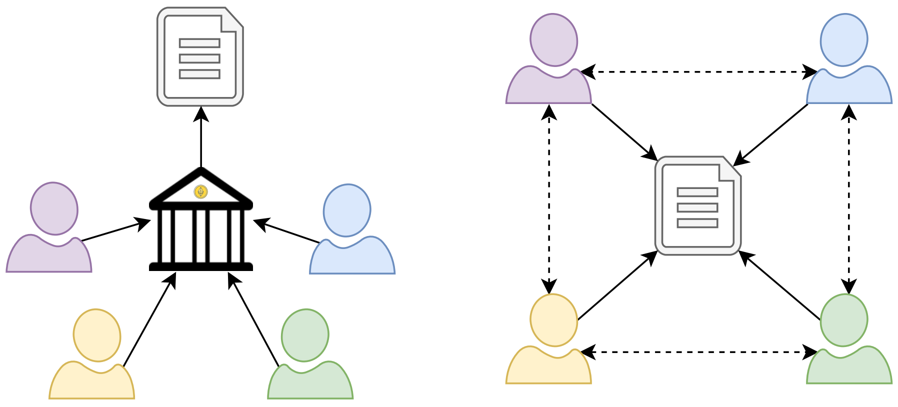


Figure 1.3: Simplified idea of distributed consensus.

1.3 Solution Approaches

The strong security properties as provided by the blockchain are interesting for the application with provenance data. Especially integrity and availability are naturally supported by the blockchain. This is also one of the reasons why a lot of work regarding the combination of data provenance and the blockchain has started to appear recently. Many contributions combine the two by using the blockchain as a data store for the provenance data, e.g., Liang et al. [12]. As we will see in later chapters, this has the disadvantage of compromising data confidentiality. This disadvantage gets addressed by some of the related work by employing different kinds of encryption techniques to secure the provenance data, e.g., Neisse et al. [13]. However, it has also another disadvantage. Most of the chains are not build for storing huge amounts of data and it can become very fast, very expensive. As we will see, there are also other possibilities how to reap the advantages of the blockchain without having to store the data directly on the blockchain, for example by off-chaining [14].

With our solution we present an approach of utilizing the blockchain to provide a common search space for different kinds of provenance solutions. We realize that different domains have different needs towards their provenance solutions and do not try to force them into one specific solution. Instead, we leave the provenance-related choices open for the specific domains and focus on creating a non-restricting way of utilizing blockchain properties across domains. Hereby we also consider the different strategies which can be employed to integrate blockchain with data provenance. Furthermore, we see the blockchains' disadvantage not being able to store big amounts of data as a natural enforcement to provide only functionality necessary for creating the common search space. Meanwhile leaving complex cross-domain and -model integration issues open to be solved by the client side. This allows us to provide a robust and flexible blockchain-based backend for cross-domain provenance data.

1.4 Thesis Structure

This thesis is structured in six chapters and one appendix:

Chapter 1: Introduction
The current chapter, the introduction, which has the purpose of introducing the general topic, the related technologies, and the document structure.

Chapter 2: Background
The background chapter contains a more detailed introduction to data provenance and blockchains. We present a specific provenance model to ease the introduction into the topic, talk about different kinds of data provenance, and also about secure data provenance. In the blockchain part of the chapter, we first introduce cryptographic tools necessary to understand the technical concepts behind the blockchain. Then, we introduce the details of the blockchain technology using one specific chain and in the end, we also present the Ethereum-chain. Furthermore, we also discuss the concepts of trust networks in the background chapter. At the end of the chapter, we have a detailed related work discussion presenting related work from the domain of secure data provenance and from the domain of blockchain-based data provenance.

Chapter 3: Design
In our design chapter, we present in detail the searchability and the duplication issue and our approach to solve these two issues. We have a detailed discussion about how data provenance can be incorporated with the blockchain and how the huge variety of possible solutions is one of the reasons for the searchability issue. Afterwards, we present the duplication issue before we introduce provenance networks as a solution to both issues.

Chapter 4: Implementation
In the implementation chapter, we present our prototype which we implemented to realize provenance networks. We go into the architectural and technical details of each of the subprojects we implemented as part of our prototype.

Chapter 5: Evaluation
In the evaluation chapter, we evaluate the concrete costs of using our solution to store provenance data in the Ethereum chain. Furthermore, we use a scenario-based evaluation to present the abilities of our solution to search for provenance and to discover provenance duplication attempts. We then also present a short evaluation of required search times to search provenance networks of different sizes. Finally, we have a qualified discussion about how our solution supports general and security-related properties from the domain of data provenance.

Chapter 6: Conclusion
In the conclusion chapter, we summarize our solution, our findings, and give an outlook on future research topics.

Appendix A: Evaluation Results

In this appendix, we list the detailed outputs of our scenario-based evaluation and also the contract addresses of our entire provenance test network.

CHAPTER 2

Background

In this chapter, we start by introducing what data provenance is, then we continue with discussing blockchain technology. Following this, we introduce trust concepts and finally, we conclude this chapter by presenting related work.

2.1 Data Provenance

In this section, we will introduce data provenance. We will explain what data provenance is, talk about different kinds of provenance, security considerations, and introduce the data provenance model recommended by the W3C.

2.1.1 Introduction

The Collins online dictionary defines provenance [15] as follows:

Definition 2.1. *The provenance of something is the place that it comes from or that it originally came from.*

The authors of [6] define data provenance as follows:

Definition 2.2. *Data provenance, one kind of meta data, pertains to the derivation history of a data product starting from its original sources.*

The authors of [8] define it as follows:

Definition 2.3. *The provenance of a piece of data is the process that led to that piece of data.*

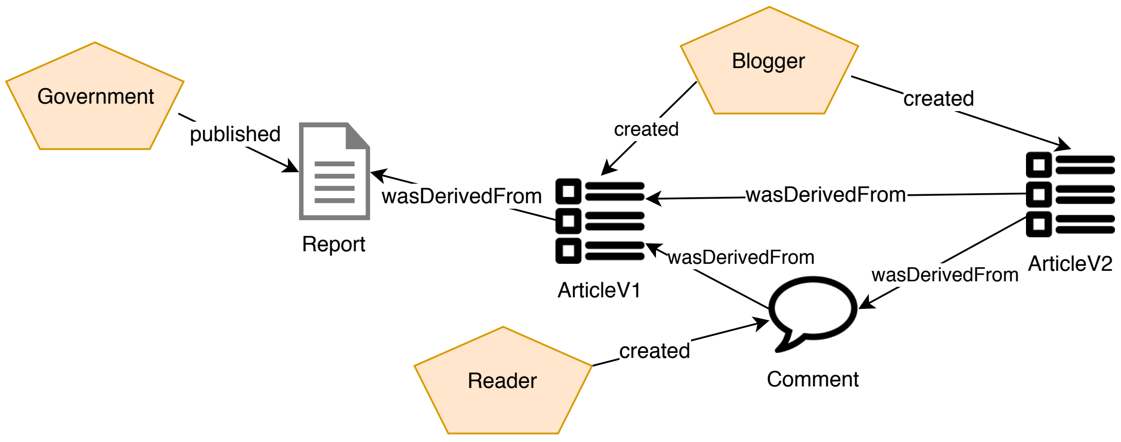


Figure 2.1: Simplified provenance view of a blog article.

All three definitions have one thing in common, they refer to where something originates. Whether this is a physical object or data is hereby of no importance. In the case of data provenance, the definitions of both [8] and [6] bring a notion of *how* into the definition. This means that when it comes to data, not only lineage is of importance, but also what alterations were made to the data.

Let us have a look at a simplified example. Given some government data, for example, a report about the environmental footprint of the country, some blogger picks up this document and writes a blog article summarizing some aspects of the report. Later, a reader comments on the blog article that there is a mistake in the summary and the blogger fixes this mistake. Any reader following this incident will have a blog article which is correct but will miss on the bigger picture. A natural reaction could be not to trust the blog entry since the comment states that there are errors in it. Thus these readers are missing the context on who influenced the data, when the data was influenced, and how the data was influenced.

An additional aspect to note in this example is that although the report is attributed to the government as a whole, it was likely created by the department of environment. This detail will be of interest later on and is not immediately visible for the blogger or the readers.

Thus, in this example, we have three different agents: the government, the blogger, and the reader. Our example also contains three different data products: the government report, the blog article, and the reader's comment. These three data products are accompanied by the activities of the different agents, being: publishing, creating, and editing.

As shown in Figure 2.1, provenance data allows us to understand the timeline and the dependencies between the different data products. We are able to see when the blogger created the original blog article and which government report it was based on. We

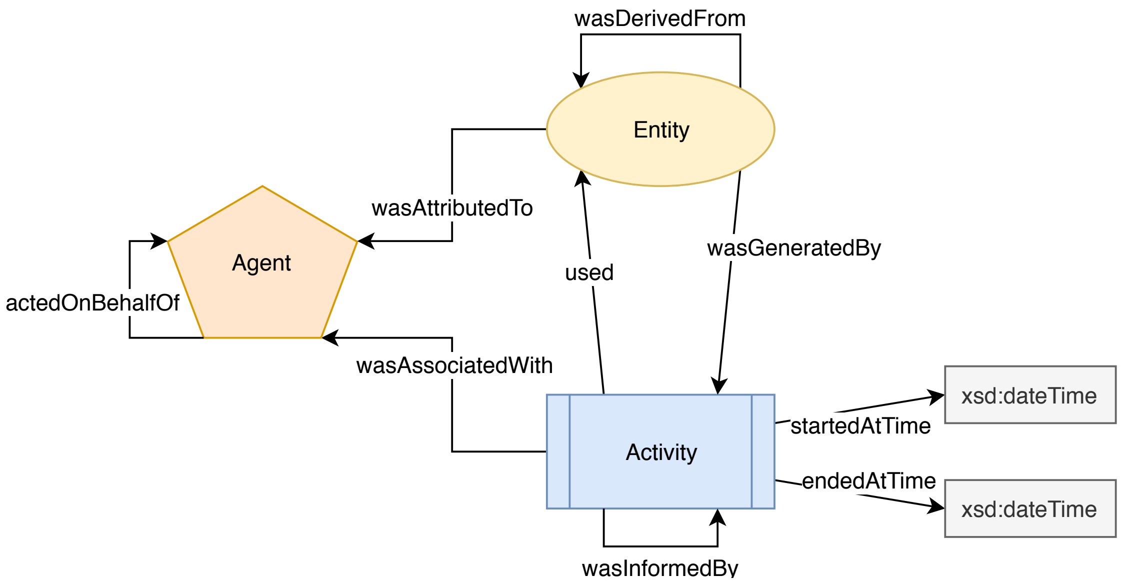


Figure 2.2: Core structures of W3C PROV-DM.

furthermore see that a reader created a comment based upon the blog article and the government data, and finally, we see how the blogger derived a new version of the blog article based on the old article and the comment of the reader.

As we can observe, in this simplified example, provenance data is metadata which describes the lineage of some data product regarding the questions who, when, and how.

Since it is rather hard to reason about provenance data without any common tools, we will introduce a provenance model to help us discuss certain properties of provenance data. A provenance model is a data model that defines some common ground on how provenance data is structured, how it describes the provenance of a data product and how it is represented. In fact, we already use a very simplified version of a model in Figure 2.1. A more detailed explanation of this model follows in Section 2.1.2.

2.1.2 W3C PROV

As we have seen in Section 2.1.1, we need a provenance model to be able to properly describe provenance information and to be able to argue about it in a unified and simplified way. For this, the World Wide Web Consortium (W3C) provides a family of documents under the umbrella term PROV [9]. This set of documents also includes a provenance model recommendation, the PROV-DM [16] recommendation.

2.1.2.1 Core Concepts

This provenance model consists, in its core, of three types and seven relations, some of which we have already met in a simplified form in Section 2.1.1. In Figure 2.2, we see the types of the PROV-DM model and their relations to each other. In the following, we give a brief description of the core types as defined in PROV-DM [16].

- **Entity:** An entity represents a thing about which we want to record provenance. This thing can be physical, virtual, conceptual, real or even imaginary. In our example, the government report, the blog articles, and the blog comment are entities. In an even broader sense, we can describe all data products we want to record the provenance of as entities.
- **Activity:** An activity represents an action that can happen over time and that interacts with entities in some form. In our example, publishing the report, creating the blog article and the comment, as well as editing the article are activities. In a broader sense, actions that create, modify, process, destroy, or otherwise consume data products are activities.
- **Agent:** An agent is someone who is in some form responsible for a certain activity that is taking place. An agent does not have to be a human being, it can also be a thing, an organization, a code library, etc. In our example, we have three agents of importance: the government, the blogger, and a reader.

The form of the different core types as seen in Figure 2.2 is recommended by the W3C PROV document family, however, the graphical visualization is not a defined notation like the one in the PROV-N document [17] but merely a tool to make reasoning easier. The PROV documents [16], however, recommend to use graphical elements as described in the W3C [18]. Next, we will also introduce the basic meaning of the relations between the types as defined in PROV-DM [16].

- **Generation:** Generation represents the creation of a new entity by an activity. In our example, the article entity was generated by the create activity. Generation is named *wasGeneratedBy* as shown in Figure 2.2.
- **Usage:** Usage marks the beginning of utilization of an entity by an activity. In our example, the edit activity used the first version of the article entity and the comment entity to achieve the generation of a new article entity. Usage is named *used* as shown in Figure 2.2.
- **Communication:** Communication is when one activity uses an unspecified entity which was generated by another activity. The comment entity, for example, was generated by a create activity and then used by the edit activity, thus representing a communication between the create and edit activities. Usage is named *wasInformedBy* as shown in Figure 2.2.
- **Derivation:** Derivation is when one entity gets transformed into another. It can also denote the influence that using one entity toke upon a newly generated entity.

In our example, the first version of the article was derived from the government report, the second version, however, was derived from the first version of the article and the comment of the reader. Derivation is named *wasDerivedFrom* as shown in Figure 2.2.

- **Attribution:** Attribution refers an entity to an agent, meaning that the entity can be attributed to the agent. In our example, the report is attributed to the government, since the government is responsible for creating it. Attribution is named *wasAttributedTo* as shown in Figure 2.2.
- **Association:** Association represents the responsibility of an agent for a certain activity. In our example, as briefly mentioned, the creation of the report would be associated with the department of environment, since it is responsible for the correctness of the government report. Association is named *wasAssociatedWith* as shown in Figure 2.2.
- **Delegation:** Delegation is when one agent assigns another one some responsibility and authority for a certain activity. In this case, the assigning agent still holds some of the responsibility for the assigned activity. In our example, the department of environment created the report on behalf of the government. The government is still responsible for the report since the report is representing the government's effort, however, for the factual correctness, the department of environment is responsible. Delegation is named *actedOnBehalfOf* as shown in Figure 2.2.

In Figure 2.3, we show our example from Section 2.1.1 completely modeled using the W3C PROV model. To make distinguishing between the different relations easier, they have been modeled in different colors. As we see, although we used a rather simple example to introduce data provenance, it results in a rather complex model of dependencies between the different entities, actors and activities. However, this model now allows for complex deductions about how the data product, ArticleV2, came to be, on which previous data it was based on, and who influenced it.

Besides the core specification explained above, the W3C PROV document family defines also a lot of additional concepts, a few of which we will discuss below.

2.1.2.2 Extensibility

Since the PROV-DM model is designed to be domain- and technology-independent [16], it defines mechanisms to ensure extensibility and to provide for the necessary adaptability towards specific domains and use cases.

Extended structures: PROV-DM supports sub-typing of core types, expanding relations, and also optional identification. The first, sub-typing, allows users to easily create domain-specific versions of the core types. The second, expanding relations, is a mechanism that allows to open up the core relations in a way so that

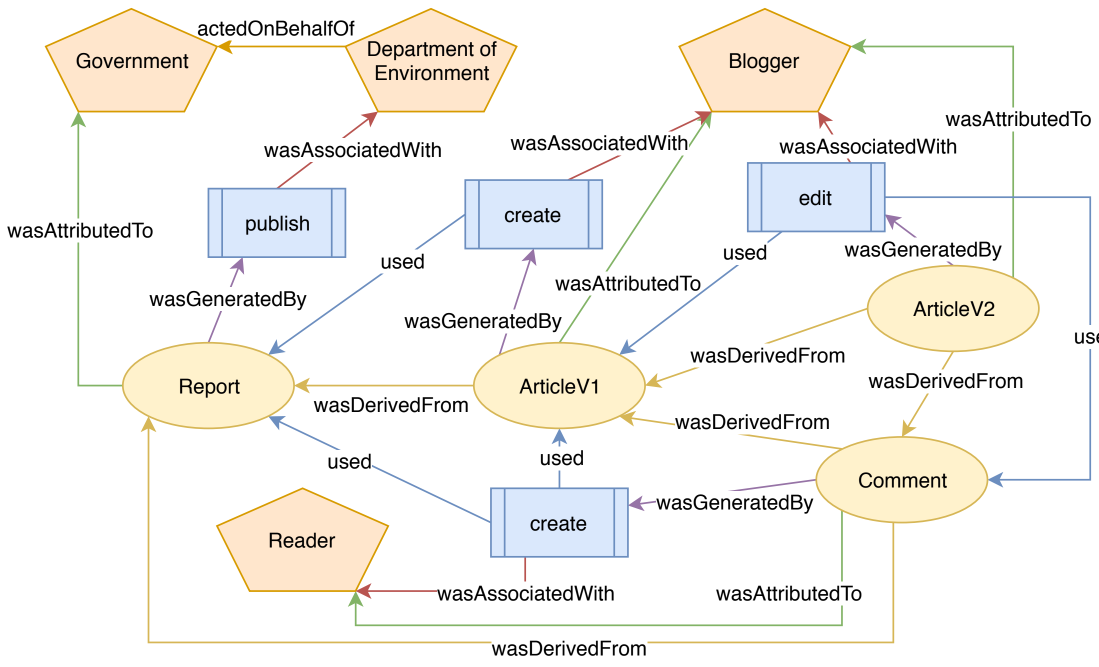


Figure 2.3: Core structures of W3C PROV-DM.

they can be used to express n-ary relations. And the third, optional identification, allows for introducing ids when there is the need to distinguish between two different occurrences of the same relation.

Extensibility Points: The extensibility points define three reserved attributes, *prov:type*, *prov:role* and *prov:location*, which allow for domain-specific adaptation of the model and facilitate the support of extended structures. Furthermore, PROV-DM does not prohibit the creation of application-specific attributes which can be used to further describe the specific domain.

2.1.2.3 Provenance of Provenance

An important concept in PROV [16] is the *bundle*, which is a named set of provenance records. The bundle itself is an entity, thus we can simply record provenance about it. We are now talking about metadata of metadata which may seem like an unnecessary level of abstraction at first. However, considering our example, the government report is one entity which is attributed to the government. Given that we have access, we could request the provenance information of the report. The resulting records may have multiple entities, representing sub-documents, graphics, statistical analysis, etc. There will also be a lot of actors, humans, and computers, that performed different activities. All of these records can be bundled into a provenance bundle attributed to the department of environment, since it is their provenance data and they were responsible for recording and maintaining it. The report itself is attributed to the government, however, the provenance information about the report would be attributed to the department of environment since they are responsible for the provenance information itself although not directly for the report as a whole. This way, bundling helps us to argue about a whole set of provenance records at once.

2.1.2.4 Accessing Provenance

Accessing provenance is managed by a technical note [10] within the PROV document family. The technical note contains some important concepts that will help us to reason about some details in the following chapters.

- **Resource:** We can think of anything that can be represented by a URI as a resource. Important to note is that from here on forward, any kind of data product will be denoted as a resource. Everything that we can record data provenance about will be called a resource in the following chapters.
- **Provenance record:** As has already been established, a provenance record is some provenance information about a resource.
- **Provenance query service:** A provenance query service is a service that allows querying for provenance records given some filter criteria.
- **Target URI:** A target URI denotes any kind of resource. Quite important in case of provenance is that this URI has not to be dereferenceable.
- **Provenance URI:** A provenance URI denotes a certain provenance record. It is not defined how much provenance information will be returned when dereferencing the provenance URI.
- **Service URI:** A service URI denotes a provenance query service.
- **Pingback URI:** A pingback URI denotes a service that can receive references about additional provenance records for a resource.

The W3C note defines two major ways of accessing provenance. Either by *direct access* through dereferencing a provenance URI or indirectly by querying a *query service*. Furthermore, the W3C note defines two major ways of how to retrieve the provenance URI or the query service location. Either by incorporating the link into the resource itself, for example by a *link*-element in case of an HTML resource. Or, by adding necessary metadata during the retrieval of the resource, for example by introducing additional HTTP headers. This will be of interest later in Chapter 4.

2.1.3 Kinds of Data Provenance

Now that we have introduced the concept of data provenance and also a data provenance model, we will talk briefly about different kinds and granularities of provenance. The authors of [8] distinguish between four different kinds of provenance, *fine granularity provenance*, *domain-specific provenance*, *database provenance* and *middleware-based provenance*. However, the same authors distinguish in [19] between work in the *database*, *workflow*, and *semantic web communities*. The authors of [20], for example, distinguish between *data provenance*, *workflow provenance*, and *cloud provenance*, although their categorization is rather broad.

As we can see, the different authors have slightly different views on the kinds of provenance, but we can roughly summarize them as follows.

- **Database provenance:** Database provenance focuses on three major points [19], how to explain the results of database queries, how to model the evolution of databases over time, and how to manage and query provenance from other sources and domains.
- **Workflow provenance:** Workflow provenance is always based on the execution of a workflow process by a workflow engine. The two major points [19] are on provenance about the execution itself and on provenance about the data product of the execution.
- **Distributed provenance:** Under distributed provenance, we collect all kinds of provenance regarding the semantic web community [19], grid provenance [8], cloud provenance [20] etc. Here, we are talking about systems that are highly distributed and can contain many different services or parts. Similar to workflow provenance, we can be either interested in the data product which we are generating or manipulating in this distributed environment, or in the services that participated in the fulfillment of a certain task.

A very exhaustive taxonomy is presented by the authors of [6]. Although quite old, many of the other categorizations can still be broken down to this taxonomy, as shown in Figure 2.4.

It is out of the scope of this work to go into the complete details of this taxonomy, however, if we go back to our blog example from Section 2.1.2, we can see in Figure 2.4, depicted in yellow, the different categorizations that are fulfilled by our example. Our provenance example qualifies as informational since it gives a broad overview of the lineage of our articles. It is data-oriented since the provenance information is about the data products rather than the process of how they came to be. It is annotation-based because we are using the W3C PROV model in our example and this particular model is annotation-based [17]. In this case, the provenance information represents overhead data on top of the data products. Finally, we are providing it in a visual graph, for reasoning.

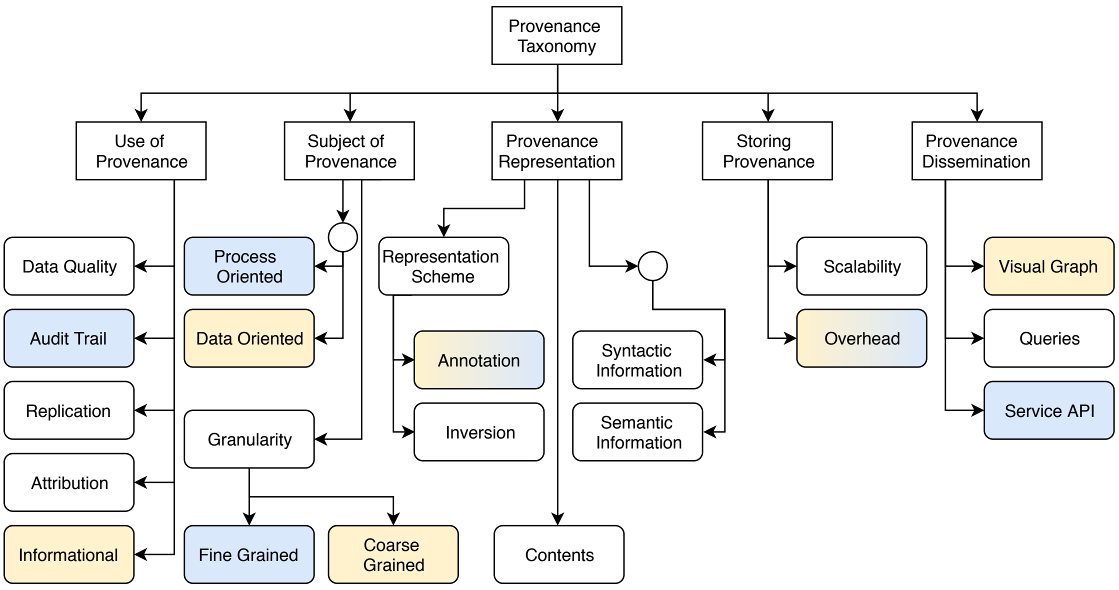


Figure 2.4: Provenance Taxonomy as presented in [6].

If we consider the government report from our example, and once again query the government service for provenance information, it is easy to imagine that the result will be more process-oriented provenance information, depicted in Figure 2.4 in blue. It would probably consider the process of how the report was created with the aim to achieve auditability of the process itself and the used sources. The granularity would be finer to allow for a more detailed audit of how the input source was manipulated by the report writers. In our example, we could receive the provenance information through a service API, which we would query. The rest of the categories depend on the actually used provenance model. Since in this example we use the W3C PROV model they would stay the same.

One point that we will need for further argumentation is the granularity of provenance. The granularity of provenance [6] is very domain- and use case-specific. Take for example a picture that we want to enhance. On the one hand, if we are working on a new algorithm for enhancing pictures every single pixel could become a provenance resource, meaning that we would record provenance information about every single pixel to properly capture the changes done by us. On the other hand, if we have a simple document cloud it can be enough to record the provenance about which algorithm was used on which picture. This means that we can have a few provenance records like in our blog example up to millions of records describing only a small part of a much bigger data product. This has a huge impact on how we process and store provenance since it can easily happen that the provenance information itself is, in the end, bigger than the data product which is described by the provenance information.

2.1.4 Securing Data Provenance

Securing data provenance is a quite complex topic. In one of the W3C technical notes [10] there is a short section discussing some of the main security considerations when it comes to provenance data. However, the W3C does not provide any suggestions on how to properly deal with these topics. A few issues mentioned there regard the need to provide for the integrity of provenance data, that provenance data should be stored securely and in a tamper-proof way, and that when retrieving the provenance data location from a source document, it should be ensured that the source is trustable and is indeed the original document.

These considerations can be generalized into the following five properties [4], as wildly adopted by the literature:

Confidentiality: Provenance data can contain sensitive information that should not be accessible by anyone. Confidentiality describes how well the provenance data is secured against unauthorized access.

Integrity: Since provenance data is used to create trust in other data objects, it is essential that the provenance data itself cannot be modified. Thus integrity describes how well provenance data is secured against modifications by adversaries but also by mistakes. This attribute is not only concerned with how provenance data gets stored but also how it is secured during transportation and processing. This includes also querying provenance data.

Unforgeability: This attribute is about how tightly data provenance is coupled with the original data product. An adversary should not be able to forge data provenance with an existing data product or forge a data product with existing data provenance without being detected.

Non-Repudiation: Ideally, data provenance should be undeniable. For example, once a doctor has taken the decision which patient gets the organ donation and the provenance about this decision was recorded, the doctor should later not be able to deny that he took the decision.

Availability: The provenance data should be easily available. This is usually put under different privacy and security constraints but consider our previous example with the organ donation. An auditor checking the hospital and its doctors' decisions should have access to the provenance data at all time without any problems.

2.2 Blockchain

In this section, we will discuss cryptographic basics before we explain what a cryptocurrency is in general. Afterwards, we will explain the concepts of the blockchain based on the example of Bitcoin, the cryptocurrency that introduced the technology first. Finally, we will discuss some of the differences between Bitcoin and the Ethereum chain.

2.2.1 The Basics of Cryptography

Before we can talk in detail about how the blockchain works, we have to talk about a few necessary cryptographic tools that are used by the blockchain.

2.2.1.1 Cryptographic Hash Function

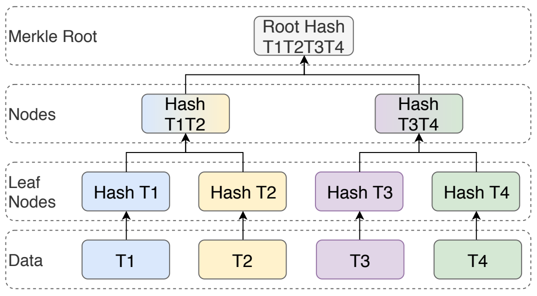
A hash function [21] is a function that takes some input data of arbitrary length and returns some data of fixed length as output. More precisely, a generic hash function has the following three properties [21]:

- The input can be an arbitrary string of arbitrary size.
- The output is a string of fixed size.
- The function can compute the output from the input in a reasonable amount of time. More precisely, an n-bit input string has an $O(n)$ runtime for computing the output string.

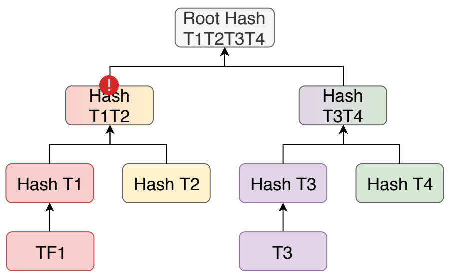
The issue with generic hash functions is that they are not collision-resistant [21]. This means that two different input strings could result in the same output string. For instance, a function that maps any input text to the numbers one to twenty-six based on the first letter of the text would satisfy all of the properties above [21]. Many of those texts, however, would get mapped to the same number. As we will see later in this chapter, collisions are bad for our use case with blockchains. This is why we need cryptographic hash functions, which have the following additional properties [21]:

Collision resistance: As has already been mentioned above, collisions are when a hash function returns the same output for different inputs. More formally [21], given two input values x and y, it is feasible to find two values where $x \neq y$ but $H(x) = H(y)$. The issue with this is that since the input space is much bigger then the output space, collisions are in theory possible. However, if it takes an infeasibly long time to find a collision, then an algorithm is said to be collision-resistant.

Hiding: The second property of cryptographic hash functions is the inability to compute the input value of the hash function out of the output value. More formally [21], if we have the output of a cryptographic hash function $y = H(x)$ it is infeasible to compute the input value x. This is an issue, especially for restricted input spaces. Take for example a dice, which has six numbers. By simply creating a table where we list the output value of each possible input value for a hash function, we can reliably tell which number was hashed by simply looking it up in our table. Such hash tables are also called rainbow tables. To avoid this problem, the definition of hiding has to be extended. Given an output value $y = H(x \parallel r)$ and a secret, randomly chosen value r, it is infeasible to find the input value x.



(a) Structure of a Merkle tree.



(b) Verifying branches in a Merkle tree.

Figure 2.5: Merkle tree and verification.

Puzzle friendliness: The third property is about how hard it is to hit a specific output value. More formally [21], if we have an output value y of an n-bit hash function and a randomly chosen value k, then it is not feasible to find a value x such that $y = H(x \parallel k)$ in time much less than 2^n. A good example is a search puzzle. Given are an id, a hash function and a target set Y. A valid solution to the puzzle is each value x where $H(x \parallel id) \in Y$ holds. The difficulty of the puzzle is dictated by the size of the target set Y and the only way to solve it is by guessing random values x.

Cryptographic hashes have a very interesting side effect, they can be used as pointers [21]. A so-called hash pointer is nothing else then a normal pointer with the additional ability to protect the integrity of the data it is pointing to. Hash pointers can be used instead of normal pointers in arbitrary data structures like trees and lists. One form of trees using hash pointers are Merkle Trees, which we will talk about next, and one form of a by hash pointers linked list is the blockchain itself as we will see in Section 2.2.3.

2.2.1.2 Merkle Tree

A Merkle tree is a cryptographic data structure that allows for separating content from verifiability [21]. The data structure is built upon leaf nodes and nodes [22]. For the sake of simplicity, we assume it is a binary tree. Each leaf node is the hash of some data and each intermediary node is the hash of its two child nodes [22]. Given some data, in our case transactions as can be seen in Figure 2.5a, we create leaf nodes by hashing the transactions. We then create intermediary nodes by hashing over the leaf nodes and so one until there is a single root node left, also called the Merkle root. If the data in any node would change or the order of the data would change, the root hash would also change [22]. This allows us for verifying that some piece of data is or is not in the tree without having to possess or store the complete data.

In the example shown in Figure 2.5b, we want to verify that two transactions are part of the block. One of them is a correct transaction, T3, and one is a fake transaction, TF1. By retrieving the correct Merkle tree and then recomputing the respective branches, we can easily see that T3 is indeed part of the tree and that TF1 is not part of the tree without the need to possess or retrieve all other transactions. Compared to the original data, T1-T4 as shown in Figure 2.5a, a Merkle tree needs much less space, time to be distributed over a network, and in case of a cryptographic hash algorithm provides us with feasible verifiability if some data has been processed or not [21].

2.2.1.3 Asymmetric Cryptography

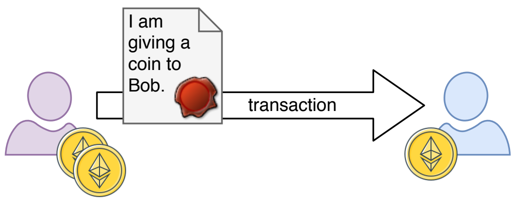
Asymmetric cryptography, which is also known as public/private-key cryptography, is a cryptographic approach [23] where you generate a key pair consisting of a public and a private key. The private key, as the name suggests, has to be guarded by the owner. The public key, however, can be distributed freely [21]. The basic idea is that everyone can encrypt messages or data with the public key of some other user and only the intended user, who owns the private key, will then be able to decrypt that message.

Beside data encryption, asymmetric cryptography has also some other use cases. A particularly important one, for us, is the use case of digital signatures. A digital signature is a way to prove that a specific message originated from a specific user. This is done by encrypting the message that is to be signed with the own private key and attaching this signature to the message. The receiver then can decrypt the signature with the public key of the sender and compare it to the message. If it matches, the receiver knows automatically two things. First, the message is from the expected sender since only this sender owns the corresponding private key. Second, the message was not tampered with by a third party. For our use case, as we will see, the first property of a digital signature is of specific interest [21]. Furthermore, there are two interesting things to note. First, usually, the message gets hashed [23] before it is encrypted with the private key. This helps to save space and time during the process of encryption and transfer. Second, although it is called a digital signature and often compared to a biometric signature, the concept of the digital signature is much closer to the concept of a seal [24], since the signature strictly speaking only proofs that you have access to the private key. However, it does not have any biometric information incorporated to proof that it is indeed you who is signing.

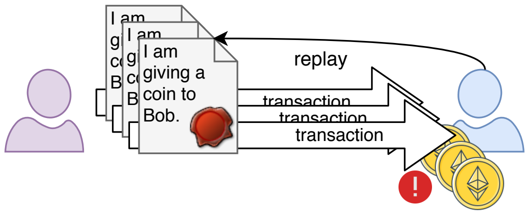
2.2.2 The Idea Behind Cryptocurrencies

The complete concepts behind the blockchain where first introduced by Satoshi Nakamoto back in 2008 [11]. The blockchain is the technology behind the cryptocurrency Bitcoin and allows to have a completely decentralized cryptocurrency. To better understand the concepts behind the technology, we will first shortly discuss what a cryptocurrency is. The idea behind a cryptocurrency is to create digital money that does not depend on a trusted third party. Let us have a look at a simplified example [5].

Imagine Alice wants to give a digital coin of some currency to Bob. To do that, Alice could write a simple digital contract that states that she gives one coin to Bob, and sign it. When executed, this digital contract would deduct one coin from Alice's account and

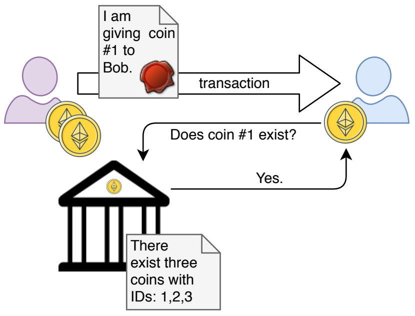


(a) Simple transaction.

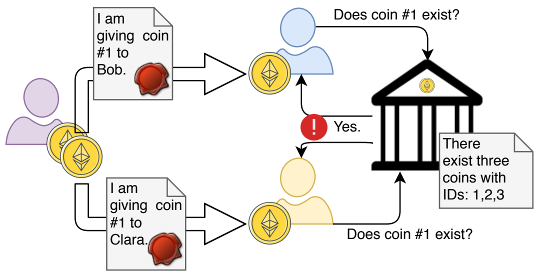


(b) Replayed transaction.

Figure 2.6: A simple transaction and the thread of replay.


I am giving coin #1 to Bob.
transaction
Does coin #1 exist?
Yes.
There exist three coins with IDs: 1,2,3

(a) Bank issuing coins.


I am giving coin #1 to Bob.
Does coin #1 exist?
Yes.
There exist three coins with IDs: 1,2,3
I am giving coin #1 to Clara.
Does coin #1 exist?

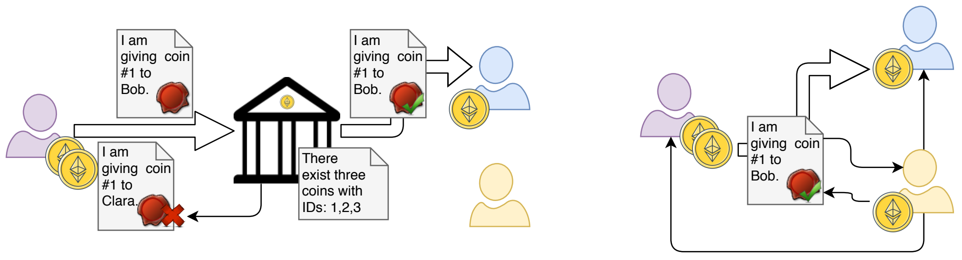
(b) Alice double spending.

Figure 2.7: Centralized issuing of coins and the thread of double spending.

add one to Bob's. The execution of this contract can also be called a transaction. Alice needs to sign with her digital signature to ensure that nobody can spend her money by simply writing a contract that gives it away.

So whenever Alice executes this contract, Bob will get one of her coins and Alice cannot take it back, see Figure 2.6a. This, however, leads to another problem, the so-called problem of replaying contracts [5]. If Bob starts replaying Alice's contract, he would be easily able to steal coins from Alices account, as can be seen in Figure 2.6b.

To avoid this, there are two different possibilities. Either we introduce a third party that authenticates Alice, for example by a shared secret. Alice would first authenticate towards the third party and then tell the third party that she wants to transfer funds to Bob. If Bob tries to replay that transaction, the third party would ignore it since Bob does not know Alice's secret and cannot be authenticated as Alice, which is how banks work today. But we do not want to trust a bank with our funds. Which brings us to the second possibility, we can make the coins distinguishable. If every coin can be clearly identified, the contract could also list the identity of the coin. This way, if Bob replays the contract, he cannot steal coins because he already owns the coin with that identity. To make this work, we need again a trusted third party that issues the coins. Otherwise, everybody could create new coins. Let us call this third party a crypto


I am giving coin #1 to Bob.
I am giving coin #1 to Clara.
There exist three coins with IDs: 1,2,3
I am giving coin #1 to Bob.
I am giving coin #1 to Bob.

(a) Crypto bank guarding transactions. (b) Everybody guarding everything.

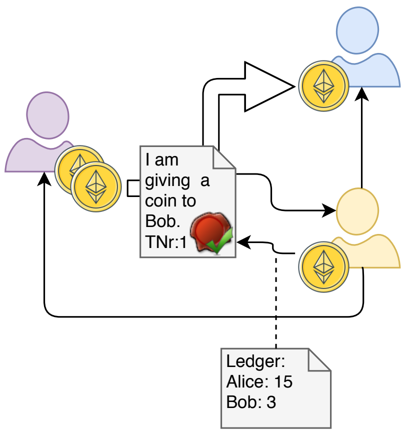
Figure 2.8: Centralized guard vs decentralized guards.

bank. This crypto bank would have the task to issue coins with unique identities, see Figure 2.7a, and verify their existence. The difference to a normal bank is that it does not have to know which or how many coins Alice owns but only if a coin exists. However, there is still another problem left. Alice could decide to write two different contracts. The first stating that Bob gets the coin and the second stating that Clara gets the coin. Since both get a valid contract, both would think that they have the coin. In this case, Alice would have successfully double spend the coin, as can be seen in Figure 2.7b.

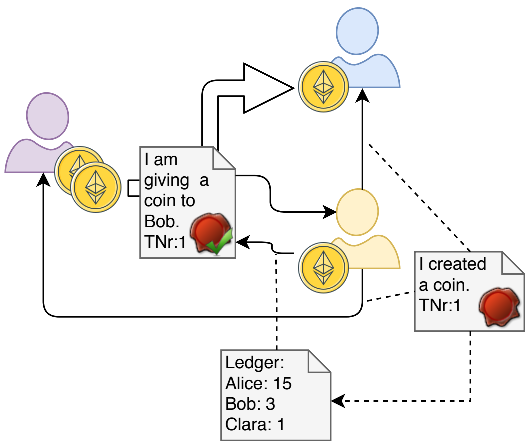
To avoid this, our crypto bank needs to be further extended to track who owns which coins. If Alice tries to double spend her coin, the crypto bank would stop the second transaction from succeeding, see Figure 2.8a. Now the issue is that we have again to trust a third party to behave correctly and non-maliciously since it is basically controlling the amount of coins that exist and the ledger that states who owns which coins.

This can also be solved differently by giving the critical information to everyone instead of to a trusted third party. Everyone would be responsible for keeping a ledger with the information who owns which coins. When Alice wants to give Bob a coin, she would give the corresponding contract to everyone, Bob and Clara. Clara would then be able to confirm to Bob that she has seen his contract and that Alice does not try to double spend, as can be seen in Figure 2.8b. If later Alice tries to double spend by trying to give Clara the same coin, Clara could simply decline the payment since she already knows that the coin has been spent before.

Our cryptocurrency would in theory work, however, we have one issue left: we do not want coins to be individually identifiable. This would make it a lot harder to split coins. Prices would need to be always a whole multiple of a coin. So to solve this final issue we can make the transactions identifiable instead of the coins. If Alice adds a simple number stating which transaction that is to here contract and increments this number for each new transaction, we would achieve the same result. Remember we introduced identifiable coins because of the replay attack. Since Bob cannot change the number of the transaction he cannot replay it because Clara would not accept the replayed transaction, since, in her ledger, she already has a transaction from Alice with that number. At the same time, Clara would check in her ledger if Alice has enough money in



(a) Transaction numbers instead of coin ids.



(b) Creating new coins.

Figure 2.9: Finalized simple cryptocurrency.

her account to prevent her from double spending it, as can be seen in Figure 2.9a. This only works because we have now a network of participants who verify transactions and have a common view on the state of the network.

Furthermore, verifying transactions could also be used to solve issuing new coins. Instead of having the crypto bank issue coins, it could be defined that whenever one of the participants verifies a transaction for the others, the participant is allowed to issue new coins. Clara, for example, who witnessed the transaction between Alice and Bob would now be allowed to create a new coin and in the process of confirming the transaction, she would also tell Bob and Alice about this new coin, as can be seen in Figure 2.9b. This way, the participants would be able to pay themselves for doing the work of verifying transactions of other participants.

We successfully have created a simplified cryptocurrency which shows us in a simplified way how Bitcoin operates. To make this work between huge amounts of participants, Bitcoin uses the blockchain as the underlying technology.

2.2.3 The Basics of Blockchain

In the previous section, we introduced the basic concepts behind the blockchain based on a simplified example. In this section, we will have a closer look behind the constructs and mechanisms enabling the blockchain.

2.2.3.1 Transactions

Until now, we did not go into detail what a transaction actually is. We just know from Section 2.2.2 that transactions are used to transfer funds and to keep track of what is happening in the network. A transaction is a construct [5] consisting of a transaction

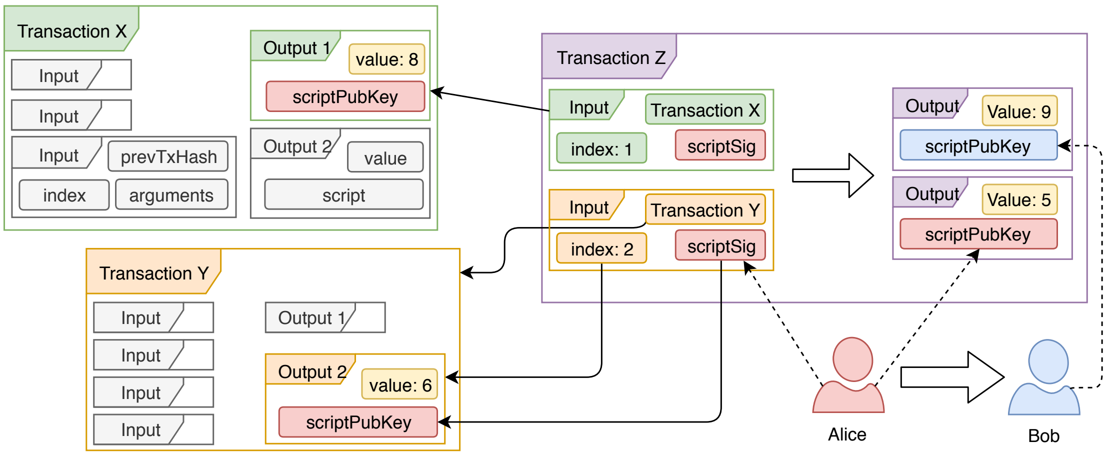


Figure 2.10: The structure of transactions.

hash which is also called the transaction id, a set of inputs, and a set of outputs. Every output consists of the following elements.

Value: The amount of coins that are transferred.

Output Script: A script that defines who is allowed to spend this money. A future transaction that tries to spend the coins in this output has to provide input parameters for this script so that it evaluates to true. The simplest script is just one that verifies that a certain user approves the transaction by verifying the signature of that user which has to be given as an input parameter. But this general approach of using scripts allows also for more complex scenarios.

Every input has the following values:

Previous Transaction ID: This id is pointing to a previous transaction on the blockchain.

Index: The index is specifying which output of this previous transaction is of interest. This output has to be an unspent output. Otherwise, the transaction will be discarded as invalid, since it is an attempt at double spending the same coins.

Arguments: The necessary arguments for the script which is stored in the output we are pointing to. In case of a simple transfer script, this will be just the signature of the user owning the coins.

When Alice wants to transfer some funds to Bob, she has to reference as many previous outputs in her inputs list until she reaches at least the amount she wants to transfer or more [5]. The *unspent transaction outputs*, UTXOs [25], that Alice has to gather are the way how Bitcoin keeps track of a user's total balance. Then, she has to specify the

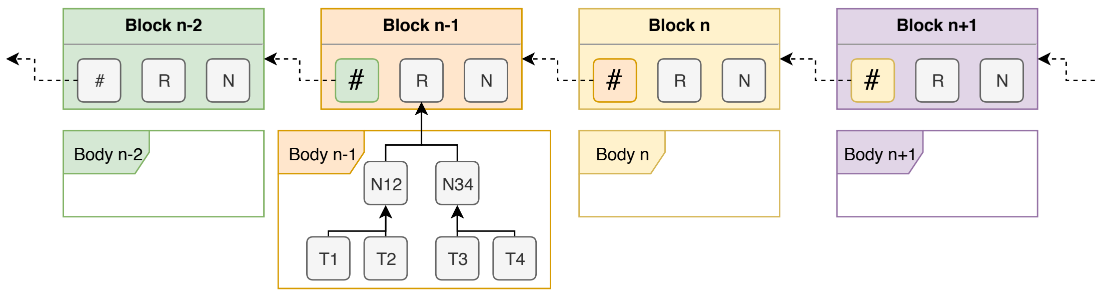


Figure 2.11: Simplified view of the blockchain.

output script. This can also be seen as a contract between Bob and Alice. If Bob is willing to fulfill the requirements of the script, he can have the funds. We will speak later more about contracts. Finally, Alice has to specify the target address in her script. This address has to be one where Bob owns the private key to, otherwise, he will not be able to retrieve the coins. When referencing an output from a previous transaction, Alice has to use up all the funds in that output [5] since it is not allowed to reference an output twice. This means that it can happen that Alice was forced to add more funds to the inputs than she wants to transfer to Bob. In this case, she can simply create a second output where she references an address that is controlled by herself. This can be compared to change in the cash world. Figure 2.10 depicts this example with all of its aspects.

A user can generate an address from a public key [5]. Respectively behind every address is a public/private-key pair that can authenticate transactions for that address. Furthermore, users do not have to manage their addresses and unused transaction outputs themselves. There are so-called wallet applications that collect the outputs and display the accumulated value of owned coins to the users. They will also create the necessary input and output lists for the users.

2.2.3.2 Blocks

In the simplified example in Section 2.2.2, one important point was that every participant has to be informed about every transaction and also about every creation of new coins. To make this possible, Bitcoin uses the blockchain, which can be simplified thought of as a distributed ledger [5]. Everybody can write into that ledger and read from it but no one can modify or delete content in the ledger. A more technical analogy would be a linked list [5] as briefly mentioned in Section 2.2.1. This linked list consists of so-called blocks where each block references the block before and contains a set of transactions, as shown in Figure 2.11. To be more specific, every block consists of a block header and a block body [11]. The header contains certain fields needed for managing the blockchain [11] and the block body contains the set of transactions [11]. This way, a user can verify multiple transactions at once instead of having to verify each transaction by itself. Remember how Clara verified the transaction between Alice and Bob in our example. Instead of doing this for each transaction separately, Clara can verify a set of transactions, possibly from different users, at once by putting them into a block.

Below we will explain shortly the different fields contained by the block header:

BlockHash: The block hash is the cryptographic hash generated from the current block header. In the case of Bitcoin, it is generated by a SHA-256 hash function.

PrevBlockHash: This field contains the block hash of the previous block. This is how out of the different blocks a linked list is created, as shown in Figure 2.11. This is also why any block already in the list cannot be changed later on. It would be necessary to change also every block following. We will discuss the security aspects also later in this chapter.

Nonce: The nonce is part of the security algorithm used by Bitcoins blockchain. It basically is a value that helps to make the hash of the current block look in a specific way. We will explain the details later in this chapter. For now, it is only important to know that it is part of the block header.

Time: The time field is the current timestamp in seconds since 1970-01-01T00:00 UTC.

MerkleRoot: The Merkle root is the root node of a Merkle tree as described in Section 2.2.1. The children of this Merkle tree are the transactions which are stored in the block body. This way it is not necessary to add all transactions to the block hash computation. As we have seen in Section 2.2.1, this allows us to verify that a specific transaction is part of this block without the need to retrieve all transactions.

2.2.3.3 Mining

The process of confirming transactions is called *mining*. The parties taking part in mining are called *miners*. To be able to confirm a block of transactions, a miner has first to solve a cryptographic puzzle [11]. This is a necessary step to provide security for the blockchain. The idea is that each x minutes a new block is created. To achieve this, miners constantly keep adding new transactions to the block they are currently mining and try to guess the block nonce so that the block hash is, in the end, lower or equal to a certain target value. Since the only way of solving this puzzle is by guessing values and calculating hashes, as discussed in Section 2.2.1.1, the security of the blockchain does not depend on the number of miners but on their computational power [5]. The difficulty of the puzzle, the target value, scales with the amount of computational power that is provided by the miners so that the block production stays roughly at the same rate. This way of confirming blocks is called *proof of work* [11] since it requires machines to perform calculations. By now, there exist also other algorithms by which one can prove a new block like *proof of stake* [26]. *Proof of work* is still used by major blockchains like Bitcoin [11] and Ethereum [27].

Once a miner has found a valid nonce that creates a valid block, the block gets distributed to all other nodes who then append the valid block to their local blockchain and the process of mining continues with the next block. Since the process of mining itself is expensive, as it costs real energy, the miners are getting two kinds of rewards for their troubles.

Coin creation: The first kind of reward is a certain amount of new coins [11]. A miner who successfully verified a block is allowed to add one special transaction to that block which transfers a certain amount of newly generated coins to an address chosen by the miner. This is also the way how many cryptocurrencies manage how many new coins and in which time frame new coins get introduced into the network.

Transaction fees: The second kind of reward are transaction fees [11]. Transactions may have a positive delta between inputs and outputs. This delta is commonly known as the transaction fee and the miners are allowed to spend these coins as they wish. Usually, a transaction offering a higher transaction fee has higher chances to be accepted by the next block compared to a transaction offering lower fees. The transaction fee gets set by the user who is creating the transaction but the miner decides which transactions are added to the next block being mined. This way we get a self-regulating transaction fee market.

2.2.3.4 Blockchain Security

Miners are the backbone of the security [5] of the blockchain. In the process of transaction verification, they are checking each transaction for correctness and only accept correct transactions. Furthermore, each miner also has to check the blocks generated by other miners, meaning that when one miner finds a valid nonce and distributes the block to all other miners, each of the other miners validates the freshly distributed block and decides if it is valid or not. Only if the majority of the miners decides that one block is valid it can be said that this block is now accepted in the blockchain.

Since the whole process of mining and distributing blocks is highly decentralized, it can of course also happen that for a short time there are multiple blocks concurrently racing for being accepted as head of the blockchain [5]. This state is averted with the rule of the longest branch [11]. Given two racing branches, one will have at some point more miners supporting it than the other branch. This also means that more mining power is behind one of the branches which ultimately leads to one branch becoming longer than the other. At this point, all miners who were supporting the losing branch will drop the losing branch and start supporting the longest branch.

This basically means that you have to have more than 50% honest miners to have a secure blockchain. This condition is known as the 51% attack [5]. If attackers manage to control 51% of the computing power behind a blockchain, they are basically able to rewrite the whole history of that blockchain by simply mining on a discontinued branch.

They then would eventually overtake the main branch and thereby force also the honest miners to accept the rewritten history as correct.

Such an attack would quite probably be discovered [5] before succeeding in rewriting major parts of the history which would lead to massive value drop of the currency behind the blockchain since the users would pull their investments. In the end, the attackers would have successfully destroyed the currency but not won any real value. This complex mix of socio-economic factors [5] behind the blockchain are the main security idea of the chain itself. The assumption is based upon the idea that as long as there is actual value in the blockchain, attackers would not be interested in destroying the blockchain itself [5] since this would destroy the value of the cryptocurrency and they would lose the ability to capitalize on the attack. Furthermore, there are positive incentives in place and self-regulating algorithms to incentives miners to behave correctly and continue confirming valid transactions.

Although the security of the chain as a whole is thereby quite well ensured and the network is subject of severe auditing to recognize big attacks on it, there still exist a bunch of smaller attack vectors [5]. Among others, some scientists have proven that given certain conditions it can be enough to control around one third [5] of the network's computational power to make specific transactions pass or not pass. Another group showed that it is possible to make use of the positive incentives to enforce blacklisting [5] of certain users. We will not go into further detail about these attacks since it is out of the scope of this work, however, it is important to know that there are a bunch of attack vectors besides of destroying the whole cryptocurrency.

2.2.4 Ethereum

In the last few sections, we have explained the idea behind the blockchain and its most important aspects based on the specific cryptocurrency Bitcoin. However, besides Bitcoin, there have evolved many other cryptocurrencies focusing on improving specific aspects of Bitcoin which were not solved by it to the full satisfaction of the community. One of these other chains is Ethereum [27]. Ethereum focuses on smart contracts. As we have already briefly mentioned when talking about transactions in Section 2.2.3.1, Bitcoin allows to specify scripts which again allow for creating more complex rules for receiving the money than just proofing ownership of a certain key. As mentioned before, we can also view these scripts as contracts or smart contracts because the code specifies the conditions under which someone can access the funds and it is not changeable after deployment. However, Bitcoin has a major limitation, since the language that it supports in those scripts is not Turing-complete [5]. This limitation is a natural one, since every miner has to execute the code in every transaction and having loops inside those scripts would make it impossible for miners to recognize wrong or malicious code. This issue is based upon the halting problem [28] which basically states given an arbitrary program and an input it is not possible to state if the program will terminate on that input. To circumvent this issue, the Bitcoin blockchain does not offer a more complex script language.

This is where Ethereum builds upon. Ethereum has a Turing-complete script language, although it is arguable that Ethereum itself is not Turing-complete, we will discuss this latter in this section in more detail. Furthermore, Ethereum actually aims to be an application engine build upon a blockchain and not simply a cryptocurrency. In the following, we will describe a few of the core features and differences [25] of Ethereum with respect to Bitcoin.

Virtual Machine: As has already been briefly mentioned, the main idea of Ethereum is to be an application platform. To achieve this, Ethereum implements a virtual machine (VM) that is capable of working with Ethereum's own opcodes, the EVM. Transaction code gets executed by the miners on these EVMs.

Gas and Fees: Like Bitcoin, Ethereum has also to deal with the halting problem. However, since Ethereum has decided to provide a Turing-complete language, the developers had to come up with a different approach for solving this. That is why they introduced the notion of gas [27]. Basically, the user has to pay for each line of code that gets executed by the miners. Users in Ethereum have to specify as part of their transaction the upper limit of gas their transaction will use. All remaining gas will be refunded to the user by the miner and if the transaction execution exceeds the provided amount of gas, it gets terminated and all changes are reverted. This way even if somebody submits a faulty program, with an endless loop, for example, the miners do not have to worry about it since they are getting paid for each iteration of the loop and once the gas runs out they can stop the execution of the transaction code. Beside for code execution, users have also to pay for storage access and for permanent storage on the Ethereum chain. Since every piece of information that is stored permanently on the chain makes the chain grow and every miner has to provide physical storage for this information, the fees for storing big amounts of data have to be covered by the users and can be quite high. The whole concept of gas prices and storage fees is set up in a way to scale with market demand and supply, however, it is out of scope to go into further details in this thesis.

Accounts: As has already been discussed, Bitcoin uses an UTXO approach for managing the balance of users. Ethereum, on the other hand, has decided to use an account-based approach. In the case of the account-based approach, the state of the chain stores a list of user accounts with the corresponding balances. This has some benefits but also some disadvantages compared to the UTXO approach. One of the main disadvantages is a lower degree of privacy. One of the main advantages is that it saves a lot of storage on the miner's side and it is simpler to reason about by developers.

Besides these core concepts as presented above, Ethereum has also a few technical differences that we will not go into further detail. One example is that they are using a *Merkle Patricia Tree* [27] instead of a *Merkle Tree* allowing for tree operations to be performed in logarithmic time.

2.2.4.1 Ethereum and Turing-Completness

As we have mentioned earlier, the language that is run by the EVM is Turing-complete [27] since it allows for arbitrary computations. However, the Ethereum chain itself is not. The Ethereum chain is designed in a way that transaction do always halt as we have mentioned above. Besides the transaction gas limit that we have mentioned above, there is a second mechanism, that restricts execution time namely the block gas limit. The sum of the gas limits of all transactions in one block is not allowed to surpass the block gas limit. Thus, a single transaction may at maximum provide gas up to the block gas limit for its computations. However, this behavior is comparable to any normal computer. A memory-intensive program will run until it runs out of memory at which point a modern operating system would kill it with an out-of-memory exception to avoid total machine failure. To solve the issue, the user of that computer could increase the available memory and try again to execute the program. We have the same behavior in the EVM itself. By increasing the provided gas, we can try to run the program again. By increasing the total allowed block gas limit, we can allow even more complex programs to be executed. This argument can even be extended. Say we take the EVM code and alter it to not require any gas to run instructions, without changing the instruction set that it can run. We then have private EVM which can run the exact same programs that can run on the Ethereum network without the need for gas. With this altered EVM and the language provided by Ethereum, we could do anything that we can do on an arbitrary computer.

2.3 Trust Networks and Trust Propagation

The scientific community likes to visualize trust between entities in so-called trust networks [29] or trust graphs. These are weighted directed graphs (V, E), where the vertices, V, represent the entities and the edges, E, the trust between two vertices. The vertices are directed because trust is not bidirectional. Take scientific work, for example, a blogger may trust a set of papers to write a blog entry, however, scientists writing papers will not necessarily trust blog entries to write scientific papers.

One example from the semantic web community is the *FOAF* [30], *friend of a friend*, schema. The idea behind this schema is to allow homepages to expose, in a machine-readable language, information about entities they know in the FOAF universe. This way a graph is formed which later was enriched [31] by trust assertions with the weights from one, absolute distrust, to nine, absolute trust. These weights could be given to entities on that network for different categories allowing, for example, to have full trust into one skill of this entity while expressing some reservations towards another skill of the same entity. For example, you could express that you have full trust in your computer science professor about computer science but only very limited when it comes to language skills.

If it comes to how trust is propagated from one entity to another, there exist many different propagation algorithms and metrics. One well-accepted categorization [32] of the different metrics is the usage of the following three dimensions:

Group vs Scalar: This refers to the way in which trust relations are evaluated. Scalar metrics, in general, compute trust between two entities while group metrics, in general, compute the trust for sets of entities.

Centralized vs Distributed: Centralized metrics rely on one machine which needs access to all trust information to compute the trust for all entities. Distributed metrics rely on algorithms where every entity in the network helps to compute a part of the network.

Global vs Local: The difference hereby refers to how many entities of the trust graph are used to compute the trust of the network or they can take only a part of the network into the computation, thus taking local bias into account.

In the following, we will describe four basic strategies [33] to propagate trust between entities:

Atomic propagation: Atomic propagation is the most intuitive one. Given three entities $a, b, c \in V$, if a trusts b and b trusts c then one can assume that a can trust c to some degree, see Figure 2.12a.

Transpose trust: The idea here is that given two entities $a, b \in V$ where b trusts a, this should imply some amount of trust from a to b, see Figure 2.12b.

Co-citation: Given four entities $a, b, c, d \in V$ where a trusts c, d and b trusts d then this implies that b can also trust c to a certain degree, see Figure 2.12c.

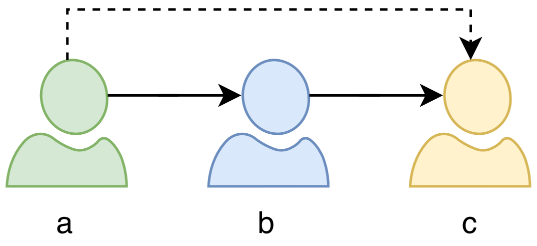
Trust coupling: Given four entities $a, b, c, d \in V$ where a trusts b and the entities b, c trust d, then this should imply some trust from a towards c, see Figure 2.12d.

2.4 Related Work

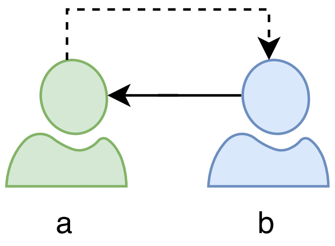
In this section, we will first introduce some interesting properties by which we can categorize related work. Afterwards, we will discuss some of the efforts of the scientific community towards secure data provenance and blockchain-based data provenance.

2.4.1 Related Work Categorization

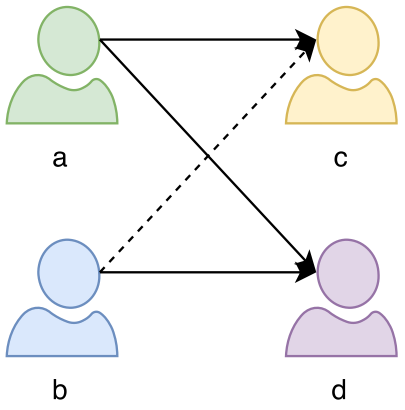
To make comparing the related work easier, we will extrapolate some interesting properties from the different topics involved. Among these properties, we will have some provenance-based properties influenced by the taxonomy [6] presented in Section 2.1, some blockchain-based properties, and some secure-data-provenance-based properties from Section 2.1.4.



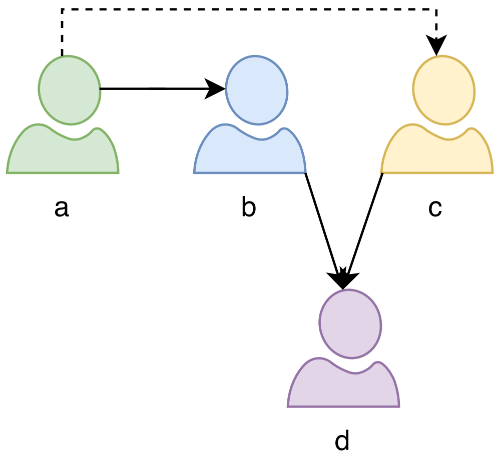
(a) Atomic propagation.



(b) Transpose trust.



(c) Co-citation.



(d) Trust coupling.

Figure 2.12: Trust propagation strategies.

Granularity: We can express granularity as *fine*- or *coarse*-grained. With this attribute, we express which level of granularity the authors envisioned for their contribution. If the user can choose the granularity, we will denote it as *custom*.

Chain: We will distinguish for which chain the contribution was developed. *Ethereum* based solutions are usually public chains, however, the *Ethereum* chain can also be deployed as a private chain. The same goes for *BigchainDB*. *Hyperledger Fabric* is a permissioned chain and *Tierion* is a third party API that allows access to Bitcoin and Ethereum.

Model: We will distinguish between existing and self-defined, *custom*, data models for expressing data provenance. Furthermore, we will take into account that not all of the solutions presented in the related work are indeed depending on a specific provenance model. These solutions will be marked as *model-agnostic*.

Storage: We will distinguish where the solutions envision storing the actual provenance data. More specifically, we will distinguish between solutions which store their provenance in the blockchain, *on-chain*, solutions that store their provenance in a non-blockchain-based data store, *off-chain*, and solutions that allow for a *custom* approach, where data can be on the chain but also off the chain.

Integrity: We will distinguish between *high*, *mediocre*, and *low* integrity. *High* integrity will be for solutions that take into account storing as well as transportation of the provenance data. *Mediocre* integrity will describe solutions that secure either the storage or the transportation. And *low* will be for solutions that have no integrity considerations.

Confidentiality: For this property, we will distinguish between *high*, *mediocre*, and *low* confidentiality. Depending on what tradeoffs the different solutions consider and if they have confidentiality considerations at all. Furthermore, we will have the value *custom* for solutions that allow the users to decide which level of security to enforce.

Availability: Availability will be also measured as *high*, *mediocre*, and *low* for solutions that have a clear approach to the issue. *High* availability, for example, is achieved when a public chain is used to store data since it can be assumed that such a chain will not have any downtimes. Respectively less downtime-secure designs might get a lower availability rating. And we will use *custom* for solutions where availability is use case-specific.

Non-Repudiation: For non-repudiation, we will distinguish between four different cases: *supported*, *not available*, *partial*, and *custom*. *Partial* will be used for solutions that provide non-repudiation under some trust assumptions. *Custom* will be used for solutions where it depends on the actual key distribution in the use case. For example, if the keys are held by the actual actors, non-repudiation is supported but if they are held by any automated auxiliary actors, non-repudiation is not provided.

Unforgeability: For unforgeability, we will have the same options as for non-repudiation with the same meaning.

2.4.2 Secure Data Provenance

First, we will take a look at blockchain-independent related work from the domain of secure data provenance. We will do this based on the five security properties we defined in Section 2.1.4. We are not discussing work from the non-secure data provenance domain since these contributions often make very simplifying assumptions about security or simply leave it as an open topic since it would be out of scope for their work. The W3C PROV [10] recommendation itself is a very good example. Although it mentions important security issues that have to be taken into account, it does not provide any suggestions to how to solve them. Another good example is the work of *Bower et al.* [34] where the provenance data gets saved as plain text in a file and folder structure on the user's computer. Other work, partially even from the domain of secure data provenance often assumes a trusted infrastructure.

Many approaches tend to solve confidentiality by using different kinds of state-of-the-art encryption techniques, like by *Asghar et al.* [4], *Hasan et al.* [35], [36], and *Lu et al.* [3]. The exact cryptographic methods depend on which assumptions the authors took about which parties should have full access to the provenance data, for example, users and auditors. There are also solutions that allow for simple querying [4] of the encrypted data without the need to decrypt it. A different approach is taken by *Tan et al.* [37]. They consider access control mechanisms to achieve confidentiality, by grouping the provenance data into different types of information with varying levels of sensitivity and then applying role-based access control mechanisms to restrict access to the provenance data. This approach works only with a trusted storage provider.

Cryptographic techniques are also preferred to solve the integrity issue for provenance data. More concretely, signatures are often used to ensure the integrity of the data, e.g., by *Hasan et al.* [35], [36] and *Lu et al.* [3]. This is a simple technique where the hash of the provenance data gets cryptographically signed by the actor. Since it is assumed that the private key of the actor is secure, anyone can verify with the public key of the actor that the provenance data is indeed unchanged and as it was created by the actor.

Signatures are also used to achieve non-repudiation. This also works because it is assumed that the private key of an actor is secure. Thus, when the provenance information of a certain action is signed with the private key of the actor who performed that action, it allows us to argue that the actor indeed did perform that specific action. This strategy is used by *Asghar et al.* [4] to secure cloud provenance as well as by *Lu et al.* [3]. The latter also assumes that the signature will be checked by the cloud service provider (CSP) before storing the provenance.

To provide availability, much of the related work is proposing the usage of CSP in one form or another. A major difference is to which degree the CSPs are trusted. *Asghar et al.* [4] are proposing a solution where the CSP cannot access the data but only stores it. *Lu et al.* [3], on the other hand, work under the assumption of trusted CSPs. *Jung et al.* [38] speak of e-Science grids which are available through the Internet, making them effectively a different kind of CSPs.

Unforgeability is often considered only one-sided. Provenance data that is secured by a hash with a digital signature is hard to forge since an adversary would need to gain access to the private key first. However, this does not necessarily mean that the original data is secured. *Asghar et al.* [4] themselves define unforgeability into both directions, however, solve it only by securing the provenance data. Encryption, as used to satisfy confidentiality, is another factor that strengthens the unforgeability of provenance data by making it unreadable to adversaries. With this in mind, also the works of *Lu et al.* [3] and *Jung et al.* [38] support unforgeability through signatures and encryption.

As can be seen, encryption combined with digital signatures are commonly used techniques in the related work to ensure the security of data provenance. The differences are often subtle and domain- or use case-specific. Very often, it is also assumed that the solution will operate on a trusted infrastructure, e.g., by *Hasan et al.* [35], [36] and *Jung et al.* [38].

2.4.3 Blockchain-based Data Provenance

In this section, we will have a closer look at related work form the domain of blockchain-based data provenance.

The authors of *DataProv* [39] focus on verifying the correctness of the submitted provenance data by utilizing the Ethereum chain. A peer voting mechanism is used to enforce the correctness of the committed provenance and they combine this with a monetary incentive towards not submitting wrong records. However, their incentive mechanism also provides an incentive for reviewing peers to simply reject any change independent of its correctness. Once accepted, they write the whole provenance record into the blockchain. In their evaluation, they do show the rough costs of the different operations they provide. They assume high-level provenance data in their work, however, the cost can stack up to be a massive issue in a high granularity context. Finally, they make use of the *Open Provenance Model*, a direct predecessor of W3C PROV, for recording file provenance without going into further details of the structure. Their integrity is protected through the blockchain for stored data and additionally through signatures. Besides integrity, the signatures also provide non-repudiation. Availability is provided by the blockchain. They do employ encryption to provide confidentiality, however only against external users. All users that are stakeholders have full access to the data, also to enable the voting. Unforgeability is given through the combination of encryption and signatures.

The authors of *ProvChain* [12] present a framework for capturing and storing provenance data for cloud applications. In their paper, they evaluate their framework based on the open source cloud ownCloud. They use file hooks to identify file access and record the provenance information related to those file operations with the help of Trieion[1], an API for storing data into Bitcoin or Ethereum. The granularity of the provenance which they record is based on files as a data product and file operations as provenance activities. The provenance gets also stored into a local provenance store and the integrity can be verified during auditing by querying transaction receipts. The provenance records stored to the blockchain are pseudo-anonymized by hashing usernames and not exposing any file contents. Like general blockchain research, however, shows [5], it is possible to revert this anonymization to a certain degree. By using a pure transaction data-based approach, they have some level of independence from the specific blockchain, however, using a third party for writing transactions to the chain introduces other risks and dependencies. Integrity is provided through the blockchain. As well as the availability of the provenance data. Confidentiality is only partially provided since encryption is introduced on the cloud storage provider side, thus the storage provider is seen as a trusted entity. Non-repudiation is also only partially given since there are access controls at the cloud provider side. Unforgeability is also given for the provenance data from the storage provider onwards.

In his Master's thesis, *Stoffers* [40] presents three different approaches to how the W3C PROV provenance model could be mapped into the blockchain. His first approach is

[1]https://tierion.com/

a document-based approach where the whole provenance information gets saved into one transaction. His other two approaches are a graph-based approach and a role-based approach. In the document-based approach, all the provenance data gets saved in the same transaction. In the graph-based approach, there exist accounts, which are basically public/private key pairs, for each element of the PROV model and relations between the elements are mapped as transactions between those accounts. The third approach, the role-based model, maps every agent from the PROV model to an account and all the other PROV elements, activities and entities, as well as the relations, are stored in the transactions. All three approaches are not based on the specific capabilities of one blockchain but can be used with any blockchain that supports a cryptocurrency. The author uses BigchainDB [41] for his prototype. BigchainDB aims to provide owners of digital assets with reliable proof of ownership. Integrity and availability are provided in all three approaches through the underlying blockchain. Confidentiality can only be provided for the document-based approach given the content has been encrypted before saving it to the blockchain. Non-repudiation depends on the underlying use case. If there is a known mapping between users and their public keys, the normal way of signing transactions provides non-repudiation out of the box. The proposed approaches do not provide any safeguards towards unforgeability.

Neisse et al. [13] present three models in their paper on how to track GDPR compliance-related data provenance by utilizing the Ethereum chain. They use the terminology, as provided by GDPR, of *Data Subject*, *Data Controller* and *Data Processor*, where the subject is typically a user, a controller is typically a service provider and a processor is typically some kind of data processing organization used by the controller. All three models are based on the idea that subject and controller establish a smart contract between them that holds a set of policies which define what operations are allowed by the subject to be performed with its data. The controller then has to check first if a certain operation is allowed with the data of a specific subject before performing it. The controller then uses a transaction to record the actual usage of the data. The main difference between the three different models they propose is based around provenance granularity. They suggest a subject-centric model where the subject defines the police contract for high granularity and high security scenarios. For low granularity and high scalability scenarios, they suggest a controller-centric approach where the controller provides the contract for all users and they can decide to accept it or not. The same principle is also used between subject and processor. The privacy of the data on the chain is ensured by using a combination of new contracts and nonces per controller and processor. The nonce represents a random value that is shared between subject and controller or processor. This random value is then used to anonymize data that is part of the provenance records in a way that only the subject can read the entries and map them across contracts. The authors, however, do not go into details how recording this provenance can be enforced on the side of controllers and processors. They do not use any specific data provenance model but they adopt the SecKit [42] approach to model data. Integrity, availability, and non-repudiation are provided through the native properties of the blockchain. They have provided an encryption-based obfuscation algorithm so that the data is not plain text on the blockchain, which brings confidentiality to some degree. The obfuscated data together with users' signatures on transactions provide for some degree of unforgeability.

The authors of TOVE [43] work towards a traceability ontology for supply chains by utilizing the Ethereum chain. They build upon a model very similar to the W3C PROV model with similar elements and even have also the notion of consuming and producing *TRU*s, *traceable resource units*. Activities in their model can consume a TRU to produce a different TRU. The complete provenance information can then be found in the blockchain with different transactions representing different activities and recording which TRU they consumed and which they produced. The resulting provenance trace is generated by reading the events thrown by the smart contract as a response to the different activities applied to it. Integrity and availability are provided through the native properties of the blockchain. Non-repudiation depends on the actual use case. If the use case accounts for matching users to signatures, then it is given through transaction signing.

The authors of *BlockPro* [44] focus in their solution solely on the secure integration of IoT devices into provenance-enabled environments by using *PUF*s, *physical unclonable functions*, and a gateway smart contract. They utilize the Ethereum chain for their prototype. Furthermore, they ensure that only registered, trusted IoT devices are allowed to store provenance data. In their setup, only the gatekeeper contract is allowed to perform a writing operation onto the storage contract itself. They do not go into details about what data model to use for the provenance data itself and also do not discuss the issue of high granularity provenance and its consequences for their system. From a security perspective, they provide data integrity and availability through the native properties of the blockchain. Integrity is additionally supported through MACs, message authentication codes, for transportation over the network. Furthermore, since they authenticate the IoT devices and each IoT device has its own private key, this solution supports non-repudiation for IoT devices. For the same reasons, unforgeability is supported in this solution.

Massi et al. [45] take a slightly different approach. They use a permissioned or private blockchain, the Hyperledger Fabric [46]. A permissioned blockchain [47] is one where all participating miners are well-known and authenticated. This way, the miners can enforce trust between each other without opening the blockchain for arbitrary users and attacks. This also allows for different proof algorithms that are not necessarily as resource-demanding as proof of work, for example. Furthermore, it allows to leverage the blockchain technology without necessarily having to create a cryptocurrency. However, a permissioned blockchain of course brings some administrative effort with it as well as less trust from outside actors since they may have fewer options to verify that the chain is properly maintained. Another advantage, which the authors leverage, of permissioned blockchains is that they can be purpose-driven, meaning that if the intent of the authors is to save huge amounts of data on it, it is less of a problem since all participating miners are participating to have access to the specific futures, in this case saving fine-grained provenance data on the chain. The authors also do extend the W3C PROV [9]

Table 2.1: Properties of the related work.

	Granularity	Chain	Model	Storage
DataProv [39]	coarse	Ethereum	OPM	on-chain
ProvChain [12]	coarse	Tierion	custom	on-chain
Stoffers [40]	custom	BigchainDB	W3C PROV	on-chain
TOVE [43]	coarse	Ethereum	custom	on-chain
BlockPro [44]	fine	Ethereum	model agn.	on-chain
Neisse et al. [13]	custom	Ethereum	model agn.	on-chain
Massi et al. [45]	custom	Hyp. Fabric	W3C PROV	on-chain

model as defined in the recommendation to fit their needs for the medical use case and integrate their solution into an existing medical platform that also provides the necessary authentication features. Since they are using a permissioned blockchain, integrity and availability strongly depend on the actual structure of their chain. If the chain is not sufficiently and evenly distributed among stakeholders, its security will be at stake. Confidentiality is partially supported by the existing medical platform, however, the state that moderate data leakage could be possible. Non-repudiation is solely based on the need for users to authenticate on the platform level. Unforgeability depends, in this solution, on the overall security of the permissioned blockchain, the degree of encryption that gets provided by the underlying medical platform, and on the user authentication on the platform level.

Janowicz et al. [48] are looking for how blockchain-based technologies can help the open science community. One of the areas they identify is about achieving reproducibility of scientific results and improved access to scientific data. They briefly talk about how the blockchain can help to link data sets to scientific workflows and help to generate data provenance that helps to reproduce the results.

As can be seen, the different solutions have a quite big variety of approaches towards combining blockchain-based technologies and data provenance approaches. Furthermore, solutions are often domain- or use case-focused and do not provide an easily generalizable approach towards combining these two technologies. In Table 2.1, we can see a direct comparison between the different contributions we discussed with respect to their non-security-related properties.

If we take a look at Table 2.2 we can see the security-related properties. Although many of the contributions are not specifically discussing security aspects, by simply using the blockchain as storage place they can get quite good availability and integrity properties.

In Chapter 5, we will use these two tables to compare and discuss the properties of our work with respect to the related work.

Table 2.2: Security properties of the related work.

	Integrity	Confid.	Avail.	Non-repud.	Unforg.
DataProv [39]	high	mediocre	high	supported	supported
ProvChain [12]	mediocre	mediocre	high	partial	partial
Stoffers [40]	mediocre	low	high	custom	not avail.
TOVE [43]	mediocre	low	high	custom	custom
BlockPro [44]	high	low	high	supported	supported
Neisse et al. [13]	mediocre	high	high	supported	supported
Massi et al. [45]	mediocre	mediocre	mediocre	partial	partial

CHAPTER 3

Design

In Chapter 2, we saw that provenance-aware systems can vary a lot depending on their specific domains and use cases. In this chapter, we will discuss two major issues that we discovered during our analysis of the background and related work, the searchability issue, and the duplication issue. Since the two issues are fairly unrelated, however, commonly solvable, we will split this chapter into three parts. The first part will present the searchability issue and present some of the reasons for its existence. The second part will then introduce the duplication issue which, as we will see, is part of the group of forgeability attacks. And the third part will present provenance networks, our contribution, to solve both issues and to provide a generalized way to approach blockchain-based data provenance.

3.1 The Searchability Issue

Especially blockchain-based provenance systems, as were presented in Section 2.4, are often very use case- and domain-specific and hard to generalize. This issue of generalizing data provenance systems is the result of the possibility to combine data provenance and the blockchain technology in a variety of different ways. Those do not only depend on the classical data provenance design choices which are often influenced by the choice of model and granularity, by the specific domain, and by the concrete use case. But, also on the blockchain-based design choices, like how to save data in the blockchain, how to query data in the blockchain, how to reference data in the blockchain and how to interpret data in the blockchain.

This leads to many different systems and approaches to solving the provenance issue. With so many different systems, it becomes hard to keep track which actors have provenance records about which resources which is mainly due to different ways of modeling and querying these systems. Consider the Internet, for example, a lot of different actors publish a lot of information on different websites. It is nearly impossible to keep track of all the different sites and their contents. To solve this, search engines were invented. Those are able to crawl the Internet for all the sites that contain certain information. With a lot of different provenance systems, we have the same issue of not knowing who has provenance about what and in which form. Consider our short example from Section 2.1.1, where the blogger tracks the provenance of the blog articles with the W3C PROV model. The blogger would also like to write about the reliability of the government report and tries to query the provenance information. Even assuming the simplest possible scenario, that the provenance does not contain any critical data that has to be protected, the least that the blogger has to know is, that provenance exists in the first place, where to find it and how to query it. After that, the blogger has still to put up with issues of interpreting that provenance information and finding a way to merge it with his own model if necessary. The issue of finding the different actors that hold provenance for a specific resource is what we define as the *searchability issue*.

In the rest of this section, we will discuss why it is so hard to create a common solution to blockchain-based data provenance, based upon the design choices that exist when building such a system.

3.1.1 Storing Provenance Data in the Blockchain

The main choice we have to discuss is how to save the provenance records. Saving all the provenance records into the blockchain is the easiest approach since it does not include any third party provenance stores for actually saving the records. All of the related work, which we discussed in Section 2.4, persists provenance data directly in the respective blockchain. *Stoffers* [40] even defines three ways, *document-based*, *graph-based*, and *role-based*, of how one could go about saving W3C PROV modeled data into the blockchain. Although he does not use the Ethereum chain in his thesis all three provided strategies can also be implemented on the Ethereum chain. However, the graph-based and the role-based approach lack the capabilities to be properly secured, especially with regards to data confidentiality, as we discussed in Section 2.4. Solutions that use the Ethereum chain often prefer to use a *contract-based* strategy to save the provenance data into the chain. *Xu et al.* have in their taxonomy for blockchain-based systems [49] a more detailed comparison between the properties of document-based and contract-based strategies. Although not named as document-based and contract-based, they do distinguish between data items embedded in transactions, which is essentially the same as the document-based strategy, and data items embedded in smart contracts, which is what we call the contract-based strategy. Furthermore, we will for the rest of this work not distinguish those two strategies from each other, since they both involve saving all the data in the blockchain and are for our use case on the same side of the spectrum, the so-called *on-chain* strategies.

Xu et al. compare these on-chain strategies also to the as commonly known *off-chain* strategies. In their work, they suggest private and third-party clouds, as well as, peer-to-peer systems as such off-chain options to store data items. Since saving a lot of data on-chain, or using a lot of computational power on-chain, is very demanding for the

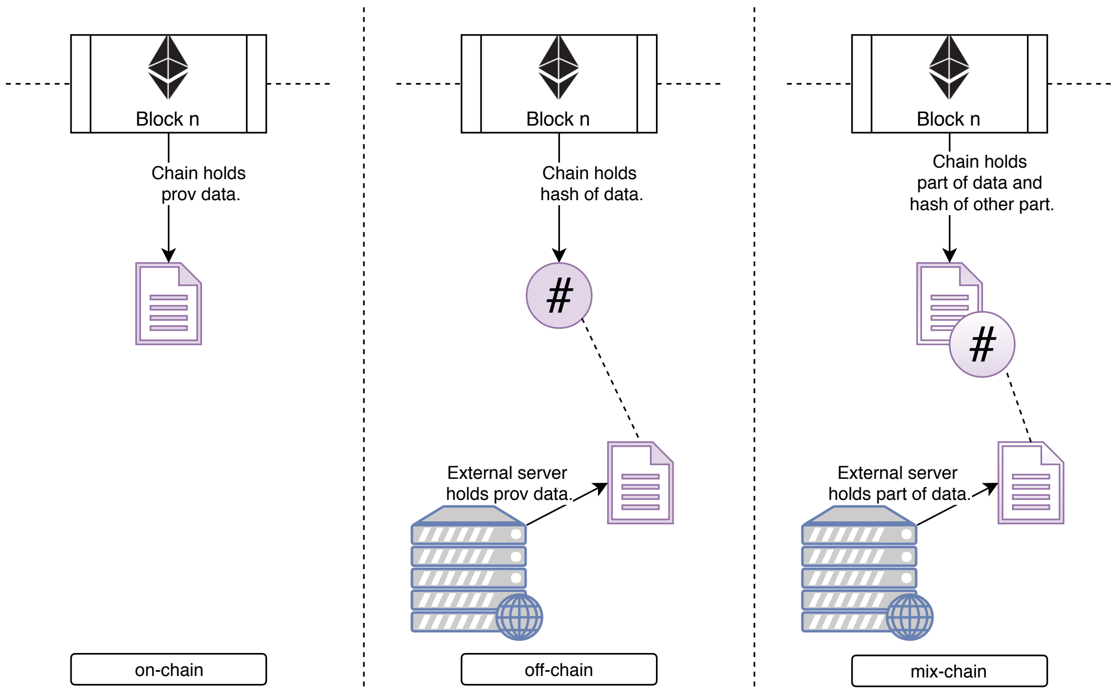

Block n
Chain holds prov data.
on-chain
Block n
Chain holds hash of data.
#
External server holds prov data.
off-chain
Block n
Chain holds part of data and hash of other part.
#
External server holds part of data.
mix-chain

Figure 3.1: Ways to to link provenance data to chain.

respective blockchain and thus very expensive it is often suggested to use off-chaining strategies to move big data chunks and computations of the actual blockchain but only provide security-related properties through the chain [14]. In general, we can say *off-chain* strategies are such that use the chain to prove some security properties of data or computations, but, store the actual data or perform the actual computations, on some different type of system, for example, a cloud service provider.

We can therefore define the notion of *on-chain*, *off-chain*, and *mix-chain* storage models, as can be seen in Figure 3.1, as follows:

on-chain
This means that the provenance records are saved in the blockchain itself. This does not yet define how the records are saved which can vary between the different blockchain implementations significantly. However, all the provenance information is saved on the chain and the client does not need to query any other stores. This is however also the most expensive strategy for saving blockchain-related data, as we will see in Chapter 5.

off-chain
In the off-chain storage model, the actual data is not stored in the chain itself but on an external server. How the data is actually stored on this server and if it is publicly available can vary between different use cases. In the chain, there is

only a hash pointer, as discussed in Section 2.2.1.1, linking to the actual data on the external server. Since a non-cryptographic hash pointer would not ensure the integrity of the linked data, it is highly recommended to use a cryptographic hash pointer, although our solution allows for both.

mix-chain
We define *mix-chain* as a mix of the other two storage approaches. In this case, part of the actual data is stored on the chain but we can have other parts that are stored on an external server and linked by hash pointers. The complete provenance data can only be found when querying both the on-chain data and the off-chain data by following the hash pointers.

Besides the storage model for provenance data in the blockchain, we can further distinguish between different storage states. The discussion about secure data provenance in Section 2.4.2 has shown us that encryption techniques are often used to secure provenance data towards some security properties. Some of the related work [13] used the approach of obfuscating the data, meaning only encrypting the critical parts of the data but leaving some structures un-encrypted. With respect to those approaches in the related work we will distinguish between the following three storage states:

plain text
This is the simplest storage state. The provenance data is saved exactly as it was recorded by the provenance-aware system. This is the easiest concept and often used in the related work, e.g., Stoffers [40] and Javaid et al. [44]. However, this state is only useful if all provenance data recorded is publicly disclosable or, in the case of off-chain storage, the storage provider can be trusted.

encrypted
In this state, the data is fully encrypted and confidentiality is ensured by the strength and strategy of the used encryption. The strategy can vary depending on the actual use case and the envisioned access privileges. For example, we can distinguish between encryption where only the recording actor has access to the decryption keys or encryption where a certain user group has access to the encryption keys.

obfuscated
Given some provenance data which is partially public and partially private, we can easily store the public data in plain text and embed the private data in an encrypted form. The disadvantage of this approach is that the provenance-recording service has to be well-programmed so that the public data does not unintentionally leak information about the critical part of the data. Since provenance data, depending on its granularity, can be highly descriptive about the overall state of a system it can be difficult to obfuscate the data in a way that no inference is possible.

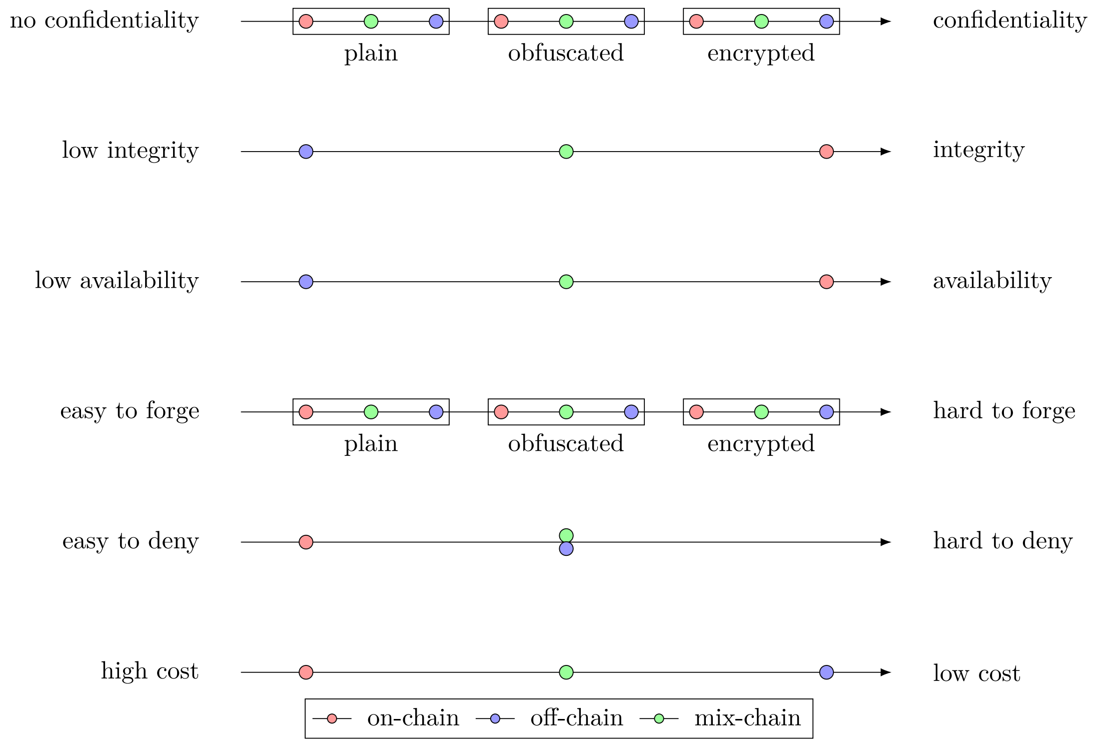


Figure 3.2: A comparison of the different properties.

The presented storage models can be freely combined with the different storage states, resulting in a set of possible combinations that have a different influence on non-functional properties. Next, we will discuss these non-functional properties. We will put special focus on how the security properties, from Section 2.1.4, are influenced. Furthermore, we will discuss the *cost* property which is also majorly influenced by the storage state.

Confidentiality

If we recall Section 2.1.4, confidentiality is about protecting sensitive data. Data provenance can not only contain sensitive data but also help to derive sensitive data. It can help to show up connections and dependencies between actors and data, which may be sensitive in nature without disclosing the actual data. To protect the provenance data, encryption and access controls are often employed. Obviously, this property is mainly influenced by the state of the data, encrypted data has better confidentiality then plain text data. And under the assumption that storage providers enforce access controls, off-chain data is slightly better protected than on-chain data, in the case of public chains. The reasoning behind this is also quite simple, on-chain data can be accessed by anyone at any time, whereas, off-chain data only by the pool of people that pass the access controls of the cloud storage provider. In the case of private chains, it depends on the actual use case. For example, within a big company many departments may be using the same private chain to secure provenance data, however, not every department should be able to read the data of the payroll department. In this use case, the company could again employ off-chaining to provide confidentiality based on the roles of the different users while maintaining only one chain.

Integrity

Integrity is about how well we can protect the data against intended or unintended manipulation, as discussed in Section 2.1.4. This is necessary to help build trust into the provenance data which again helps to build trust into the respective resource. Integrity is for on-chain data always higher than for off-chain data. Although we still have the hash in the chain to protect the off-chain data, we can only check the integrity with it, not ensure it. For example, we have a cloud provider who provides storage for provenance data and that data is secured by a hash on-chain. When we query the data we can easily confirm the integrity of the data, however, if something goes wrong at infrastructure level or the cloud service provider maliciously deletes the provenance data, it is lost and the hash cannot help us retrieve the data. Thus the integrity of the data would be compromised.

Availability

Availability describes the total amount of time the provenance data can be accessed when needed. As long as a specific blockchain exists, there is a network of miners supporting the data on it. Given this highly distributed nature, we can say that provenance data saved on-chain tends to have a higher availability then off-chain data relying on local infrastructure.

Unforgeability

As discussed in Section 2.1.4, unforgeability is about how hard it is to forge provenance records. It plays a huge difference if we consider an external adversary or an internal one. Methods like encryption and signatures can make it very hard for an external adversary to forge provenance data, however, not necessarily for an internal one, as we will see in more detail in Section 3.2. Involving the blockchain requires transaction signing by a private key, which we assume is well protected and improves defense against forgery by default. This applies for all storage models since to forge a signed hash we again require the private key. Furthermore, similar to confidentiality we argue that off-chain data is slightly better protected than on-chain data. This argument is based on the additional access control that exists. We assume that it is easier to forge data which you can read and analyze. If you can see the required granularity and provenance model it is easier to produce fake data of the same quality than when you have to guess all of these properties. And finally, encryption increases forgeability protection further due to preventing unauthorized access and reading of the data.

Non-Repudiation

As we discussed in Section 2.1.4, non-repudiation is about an actor not being able to deny his action once it was recorded. Again under the assumption that the private key of the respective actor is secure, by signing the transaction to the blockchain the actor acknowledges the responsibility for the action. However, this has a precondition that has to hold true. The public key of that actor has to be mappable to the respective actor. Many public blockchains are pseudo-anonymized by nature and allow for arbitrary creation of new key pairs. Thus systems involving a third party, like cloud service providers, are better suited to fulfill this property since they usually require user authentication and can perform the task of key mapping. The storage state has no influence on this property.

Cost

Most chains, especially public ones, require users to pay transactions fees to the miners, who keep the chains alive. These fees are often dependent on the size of the transaction. In Ethereum, for example, it depends on the amount of code that is executed and the size of the data that is to be stored or processed. These costs can get quite high and are usually by magnitude higher than the costs of cloud storage providers. A more detailed cost evaluation will be provided in Chapter 5. On private chains, these fees may be nonexisting, however, there are instead the costs of maintaining a chain. It is very use case- and setup-dependent how high these maintenance costs will be. They are a mix of hardware and operational costs like with common cloud infrastructure. Private chains have one advantage regarding the operational costs compared to public chains, it is easier to switch to a less resource demanding proof algorithm which can save a lot of operational costs. This is why we consider only the public chain case in Figure 3.2.

Figure 3.2 shows us the different non-functional properties in comparison to each other regarding storage model and storage state. There are many possibilities how those properties can be combined in meaningful ways to fulfill use case- or domain-specific requirements. This is one of the biggest issues why blockchain-based provenance solutions are hard to generalize.

3.1.2 Querying Provenance Data from the Blockchain

Let us consider the approach of querying provenance data as recommended in the W3C PROV documents [10]. We recall that provenance data can be either queried directly or by pointing to a provenance query server, as discussed in Section 2.1.2.4. In either way, by following the reference, embedded in the resource itself, we can obtain the provenance data that belongs to a certain resource and analyze it. Depending on where and how this data is stored, we can build a certain amount of trust into the provenance data and then use this trust to build trust into the original resource.

Now, if we consider a similar approach in the blockchain we again, in order to stay conform with the W3C PROV recommendations, can either reference the provenance information directly by pointing to the transaction where it is stored or we can reference a smart contract which acts like a provenance query server. In the W3C PROV recommendations,

Listing 3.1 EIP 831: URI Format for Ethereum

```
request = "ethereum" ':' [ prefix '=' ] payload
prefix = STRING
payload = STRING
;STRING is a URL-encoded unicode string of arbitrary length
```

Listing 3.2 Part of EIP 681: URI Format for Ethereum *pay* prefix.

```
payload = targetAddress [ '@' chainID ] [ '/' functionName
 ↪ ] [ '?' parameters ]
chainID = 1*DIGIT
functionName = STRING
parameters = parameter *( "\&" parameter)
parameter = key '=' value
```

these references are done by using either a provenance-URI or a service-URI. In the blockchain, however, we identify transactions and smart contracts by addresses. In the Ethereum community, there is an Ethereum improvement proposal that deals with this matter, EIP 831 [50]. It proposes a general structure for Ethereum-based URIs, as shown in Listing 3.1. The *prefix* is hereby a short identifier that defines the use case, for example *pay* for payments, and the *payload* contains the content for the specified prefix. This generalized Ethereum URI structure can be extended for specific use cases to carry the required payload. For example, in EIP 681 [51], the Ethereum community proposes an extension defining the *pay*-use case. We will not go into the full details of this extension but only briefly discuss the interesting parts for our use case, as shown in Listing 3.2. The *targetAddress* is simply an Ethereum address, for example, a referenced contract. The *chainID* specifies if the address is to be looked upon the main chain or on one of the test networks. With *functionName*, users are able to specify which function to call on the specified contract. *Parameters* allow specifying the necessary parameters to call this function on the contract.

This means that we can create URIs that can point to a specific address in the Ethereum chain and also where we would expect to find provenance records for a certain resource. However, these URIs have still the issue that they need clients that are able to interpret them. For example, if you take a standard Web URL to an arbitrary page and execute it, your operating system will know which client program to use to open that URL, which is most commonly a Web browser. However, if you open an Ethereum URI in your Web browser, the browser will not know what to do with it. Thus, to open an Ethereum URI there will always be the need for some specialized client which is capable of interpreting that URI. Furthermore, the EIP 831 [50] is purposely kept open and extendable to allow specific use cases to adopt the URI structure in a way that allows the use cases to cover their needs. This means that it is likely that different client implementation may choose different URI specifications to fit their needs. Furthermore, for a client to be able to access its data on the chain, it is not necessary to implement any URI standard at all. This combined with the fact that the EIP 831 [50] was proposed in 2018, leads to many different solutions simply implementing their own use cases without considering interoperability, as seen in Section 2.4.

This behavior is further promoted by clients often being required to provide functionality that processes or enriches the data which is stored in the blockchain. Take a classical Web page. When the browser requests the page, the server will either process the data as required and provide only a view or it will deliver a client application that can be run by the browser, which is able to process the data client side. If we look at the broader picture, then the Web browser is the actual client. It has to be installed on the local machine and provides the capabilities the end user needs to be able to load a specific Web page. In case of blockchain-based applications, we have the same issue. We need client applications to be installed in order to use the blockchain-based application. However, there is one huge difference, a blockchain-based application is not able to deliver its client-side code simply by itself. This is due to the fact that writing data to the blockchain is expensive, however, querying the data is free and unrestricted. Since the client-side code is simply a huge amount of data that would need storing in the blockchain, it is better off-chained. This also holds for complicated computations and data transformations, as in general discussed by Eberhardt et al. [14].

This lead to the community trying to utilize other existing technologies to create client applications for the blockchain which not only provide connectivity to the blockchain but also off-chaining of computation and data. This makes it particularly difficult to write generalized clients. However, since processing the stored data is a client-side task, every client application can do it using its own resources. This means how fast a client can process any given provenance data from the chain solely depends on the client's own capabilities, which allows for automatic load balancing.

In the following, we will define the main categories of clients. All clients have one thing in common for them to be able to communicate with the blockchain they have to connect to a blockchain node. This can be either a node that is running on the same machine as the client and synchronizing with the blockchain or by a remote connection to some node in the chain that allows remote connections. Clients that are connecting to Ethereum to provide some functionality are commonly known as DApps. DApp is standing for decentralized application [52] and is commonly associated with clients for Ethereum smart contracts.

Local Clients

As a local client, we categorize all clients that users have to install on their local devices to be able to use them. A classical example is the email client Thunderbird for example, is a local client that connects to a remote email server to provide its functionality. The same would be done by local blockchain clients. They would connect to a local or remote blockchain node to access the blockchain and provide their functionality.

A special case of a local client is the Mist browser, which is developed by the Ethereum team. The Mist browser aims to allow users to access different DApps on their computers. The Mist browser is comparable to a common Web browser and the DApps to Web pages that can be loaded. Combined with the ability of the Mist browser to act as an Ethereum wallet, it also allows users not only to query data in the chain but also to send requests and transactions to smart contracts. The idea for providing the necessary client-side code is to query external servers like a normal browser and to load the actual DApp interface from there.

Proxy Clients

As proxy clients, we categorize all clients that are hosted somewhere on the Internet. They function as clients towards the blockchain and as a server towards their own clients, which would usually access their functionality through the Web browser. The necessity for such proxy clients can arise for different reasons.

Caching Depending on the amount of provenance data that is saved or that needs to be queried, it may be necessary to implement an intermediary server to allow for faster replies, preprocessing provenance data and indexing. Since data which is once written to the chain cannot be deleted anymore, it may be desirable to have such an intermediary server acting as a client to allow caching of provenance data that is interesting for a specific user. For example, take the information technologies department of TU Wien. This department may be only interested in provenance data concerning its own department so instead of every client having to query through all the provenance data of TU Wien, each time access is needed it could simply keep a cache of the relevant information or of the addresses pointing to relevant information and thus allow clients to query faster.

Low-power clients Such an intermediary server could be needed to provide access to the data for clients that are not strong enough to run a full wallet application or which do not possess the processing power to retrieve results in sufficient time themselves.

Provide API For some use cases, it could be necessary to provide an access API similar to already existing provenance storage solution to allow for easier interoperability. In such cases, an intermediary server would again be the right choice to abstract the blockchain backing it.

Extend functionality A proxy client could also take care of merging data and authenticating users where necessary. For example, if we have a mix-chain solution, as discussed in Section 3.1.1, the proxy client could be responsible to merge the on-chain provenance data with the off-chain data before providing the complete data for the end user.

A special case of clients are single page applications [53]. Single page applications describe a special set of Web applications that are self-contained. Once delivered to

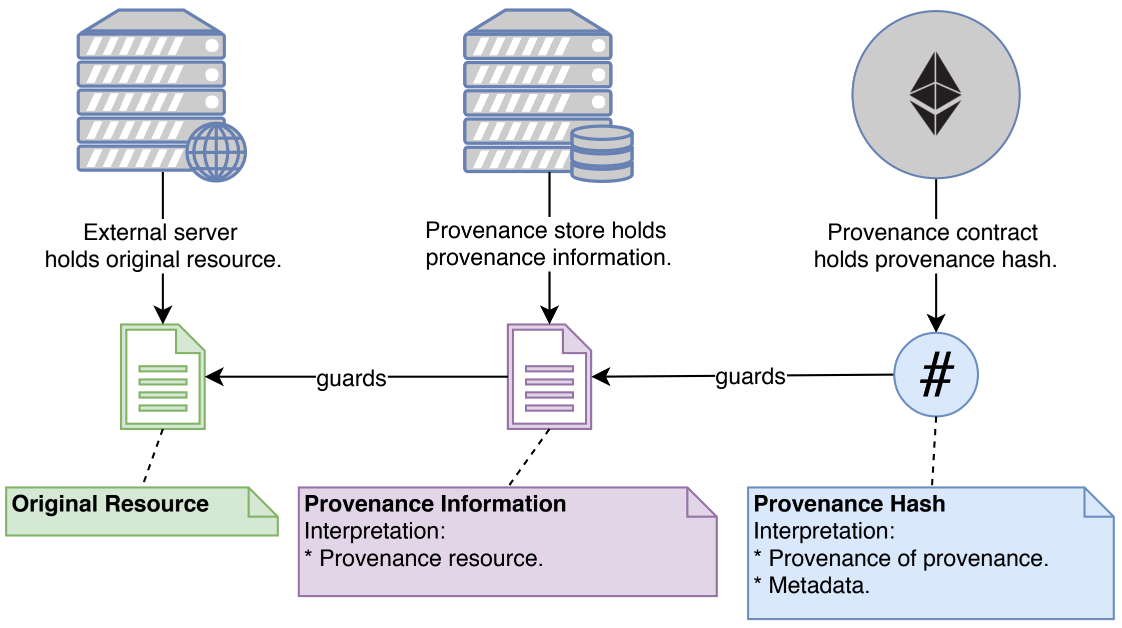


Figure 3.3: Off-Chain storage model.

the client browser, such an application has only to communicate with the server when it wants to update the database or query additional information, however, not for its own functionality. A big advantage of such applications is that they can be provided by static page servers. Furthermore, there exist plugins for modern Web browsers that allow to connect to Ethereum blockchains and provide this connection to the current Web page. These plugins combined with such a single page application allows us to create Ethereum clients that need a remote server only to be delivered to the end user's machine but then can provide their functionality as if they were a local client. Such a single page application combined with the browser plugin that enables the connection to the Ethereum chain behaves a lot like the Mist browser with the huge difference that clients already have a browser installed on their devices.

3.1.3 Interpreting Provenance Information

Until now, we have applied a top-down view on how data provenance and the blockchain can work together. We considered the big picture and talked about how we can save and query provenance data to and from the blockchain. However, we did not yet concern ourselves with the bottom-up view. More concretely, we did not think about the details of the problem, like what the meaning is of the information saved to the blockchain. In Section 3.1.1, we identified three different ways of how provenance data can be saved to the blockchain. Take the case of the on-chain model, for example, we receive after querying the blockchain for a specific resource plain provenance information about that resource, like with any non-blockchain-based provenance store. However, when querying the off-chain model for a resource, the result will be a provenance hash. The issue with this is, that this provenance hash represents the proof that a certain amount of provenance records is correct. In other words, this hash is protecting some provenance information and not the original resource, as shown in Figure 3.3. Since the hash is not provenance information about the original resource, it can either be provenance information of the provenance information, or it can be auxiliary metadata for the provenance system which we are using. In the rest of this section, we will discuss these two interpretations.

3.1.3.1 Provenance Information

If the hash is seen as provenance information it is, as mentioned, not the provenance of the original recourse but the provenance of the provenance of the resource as shown in Figure 3.3. In this section, we will discuss the provenance model-based way versus the provenance model-independent way of handling this second level provenance. The provenance model-based way works by extending the specifically used provenance model to be able to handle this additional information, which we will discuss assuming the usage of the W3C PROV model. The provenance model-independent way works by threating the original provenance information as a resource and applying the originally used model on this resource.

Extension: Since an extension-based solution requires a specific model, we will assume the usage of the W3C PROV model for the sake of this discussion.

The W3C PROV recommendation allows to easily track the provenance of provenance data with the construct of bundles. The construct of bundles is however also intended to be used for use cases like recording which actor recorded a certain set of provenance records. This provenance about the recording agent would also need to be protected by the hash. This could be solved by bundling over the bundle or by implicitly including this information about the bundle into the hash. Left aside that bundling a bundle is not allowed in the W3C PROV recommendation, either way, would make some inference necessary. Either what exactly is covered by the hash or who the author of the bundling bundle is.

Another construct provided by the W3C PROV recommendation is the document. A document is a house-keeping tool within the W3C PROV recommendation. It can carry bundles and provenance expressions. However, it does not have any notion of identifiers thus we are not able to reliably identify a document with a certain set of provenance records. This means a query service that gets queried for one resource two times could return two different documents with the contained data changing between the query requests.

As you can see, we would need to extend the W3C provenance recommendation by a new type of housekeeping construct. This construct, we can call it *package*, needs to be able to carry provenance records like a document and be identifiable like a bundle, making it an entity itself. This way, we could properly express provenance information about provenance information. The obvious disadvantage is that clients and stores would need to be able to handle this new provenance extension. This is not a minor change since existing clients and stores would need to be able to read and interpret the extended model basically rendering all existing provenance clients and stores useless for a blockchain-based usage.

Resource: In the case of the resource-based solution, we could handle the provenance information itself as a resource. To make this possible, we would need stores that reliably return the same provenance records given a query. This could be done either by the way the store is constructed or by introducing certain information into the provenance records that can be used for querying, for example, versioning. This way, we would not need provenance models to incorporate identifiable structures. Identifying provenance records would be done implicitly by the provenance store itself. Once we are able to get for a certain URI always the same set of provenance records from a provenance store, this set of provenance records, strictly speaking, becomes a resource that can be identified by a URI and for which we can record provenance information. This way we can record provenance about provenance resources without the need to extend any provenance model but by simply using it. This way we do not need to render all existing provenance clients and stores useless.

Both cases have the neat advantage that the off-chain provenance contract can behave like an on-chain provenance contract since it is storing provenance records. That those provenance records are guarding provenance information and not the original resource is not important for the provenance contract. However, this is basically keeping meta information about the meta information which can be misused. More concretely, both concepts allow for deep provenance hierarchies, in the style of the provenance of the provenance of provenance and so on.

3.1.3.2 Auxiliary Metadata

Another way of interpreting the hash is as auxiliary metadata, again see Figure 3.3. This means instead of incorporating the information saved in the blockchain into existing provenance concepts, we keep them separated and a client in sense of Section 3.1.2 has to manage how to interpret the hash. For example, the metadata saved in the provenance contract could contain where the actual provenance records are saved, how to query them and of course the hash of those records. The client or intermediary server would then have to resolve this metadata and retrieve the actual provenance information. For the end user, this would be done seamlessly. The user would only have to point the client to the according provenance contract and the resource in question and would get provenance information that has been verified on the fly. The obvious disadvantage is that the user has to use a client that is capable of interpreting that metadata correctly. Another disadvantage is that the provenance store has to be able to return multiple times the same provenance records. Like in the case of handling the provenance records themselves as a resource we could do this either through storage capabilities or through extending

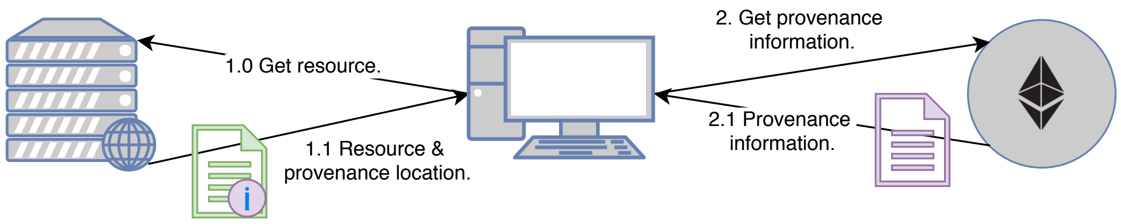

1.0 Get resource.
1.1 Resource & provenance location.
2. Get provenance information.
2.1 Provenance information.

Figure 3.4: Referencing on-chain storage model.

the details of the provenance information and thus allowing for exact provenance queries. The advantages are that we do not need to touch the provenance model definition itself and that the creation of highly customized solutions is possible. This can, however, be also a disadvantage, leading to different definitions how this metadata looks and how clients interpret it.

We have to note that the client being able to process this metadata is a different disadvantage than the client changes in the case of a provenance model extension as discussed previously. This is mainly due to not affecting classical provenance clients and stores but only affecting the specialized clients needed for reading the data from the blockchain, as described in Section 3.1.2.

3.1.4 Referencing Provenance Information

In Section 3.1.3, we talked a lot about how we can interpret the provenance hash but we did not discuss how to link the provenance hash to the respective provenance records. More precisely, when we obtain a provenance hash how do we know where to find the corresponding provenance records. But, also in the other direction, given we have some provenance records, how do we know that there is a hash guarding those records.

Consider the provenance data is on-chain, this means we can simply treat the provenance contract as a provenance store. The only thing we would need to address is how to access that information. As discussed in Section 3.1.2, we would need specialized clients that are able to interpret the Ethereum URI. These clients could then simply read out the provenance information from the provenance contract and return it as plain provenance information, see Figure 3.4.

As discussed in Section 2.1.2.4, the W3C model, for example, suggests to include this URI as part of either the resource or the transport protocol which is used to retrieve the resource. However, this alone would not yet allow anybody to access that information due to the need of a specialized client. This means someone who wants to query the provenance information would first need to find out which client to use. The resource could reference the client that is needed to access that information, instead of the Ethereum URI. This way by querying that client one would directly receive the provenance information itself. However, this would mask to a certain part where exactly the provenance information is saved. For example, in case of a proxy client, one would need to verify explicitly

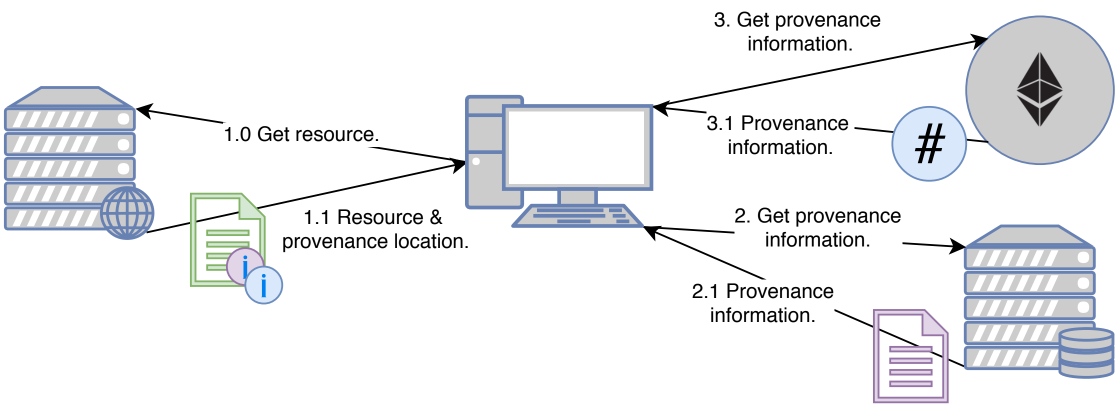

3. Get provenance information.
1.0 Get resource.
3.1 Provenance information.
#
1.1 Resource & provenance location.
2. Get provenance information.
2.1 Provenance information.

Figure 3.5: Referencing off-chain storage model on resource side.

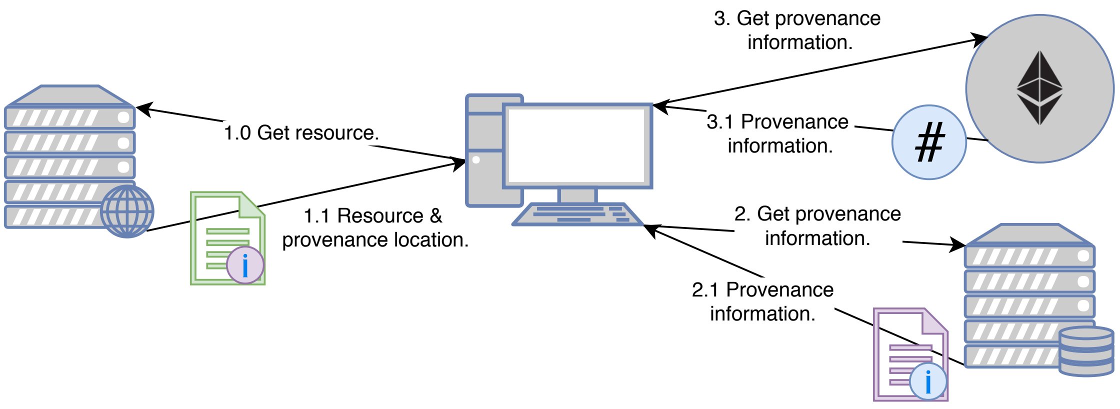

3. Get provenance information.
1.0 Get resource.
3.1 Provenance information.
#
1.1 Resource & provenance location.
2. Get provenance information.
2.1 Provenance information.

Figure 3.6: Referencing off-chain storage model on provenance information side.

that the client indeed queries the chain for the information. A third option would be to provide both, a link to the provenance contract and suggesting which client to use for querying this provenance contract. By simply referencing the Ethereum URI, on the other hand, we leave it open to the end user to decide if the used client is able to handle that specific provenance information. As you can see all three options have advantages and disadvantages regarding usability and extendability.

Furthermore, if we consider the off-chain model for example. In the provenance contract, on-chain is only the hash of the provenance information. The actual provenance is in a different store off-chain. This means that both the provenance contract and the actual store need to be referenced. How we solve this depends to a certain part on how we interpret the provenance hash since different interpretations offer different tools how we can link the hash to the provenance records it represents. All interpretations have the idea of referencing both stores in the resource in common, see Figure 3.5.

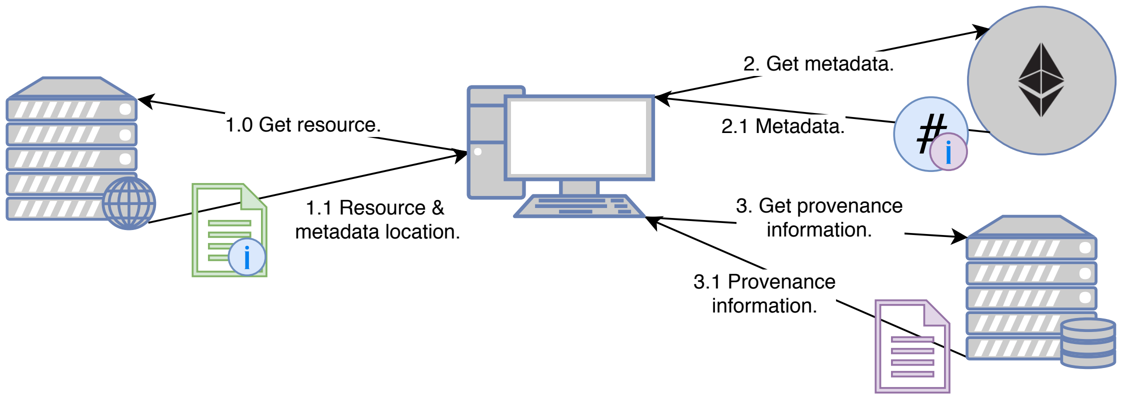

1.0 Get resource.
1.1 Resource & metadata location.
2. Get metadata.
2.1 Metadata.
3. Get provenance information.
3.1 Provenance information.

Figure 3.7: Referencing off-chain storage model on metadata side.

Provenance Information: Extension

In case that the provenance hash is interpreted as provenance information, in form of a model extension, we can simply reference the provenance contract as another provenance store in the original resource, see Figure 3.5. When a client retrieves the information from both stores, it will have the complete provenance information including the provenance hash which is guarding the provenance records.

Another approach is to define attributes for the extension that tell us which contract holds the hash. In this case, the reference is part of the provenance information itself and a querying client has to extract it from there, see Figure 3.6. However, this would only work in combination with smart contracts and not with any other way of storing provenance in the blockchain. This is mainly due to the need to be able to point to an address in the blockchain from within the provenance records before the hash is actually written to the chain. Resulting in a chicken-egg problem between the address of the transaction and the provenance hash. Smart contracts do not have this problem since they have already a fixed address in the blockchain which can be used in the provenance records and thus as part of the provenance hash.

Provenance Information: Resource

In this case, we see the provenance of the resource as a provenance resource. We have again two different ways of how we can solve the linking-issue.

The first option is to stay conform to the way the original resource is referencing its provenance records and to use the same method in the provenance resource, see Figure 3.6. Again, only having to solve the issue that is also present for the on-chain provenance, of how to reference the contract in general. However, this comes at the cost that the respective provenance stores have to be able to publish reference information as the resource provider does. Note also that although we have here the same figure as in case of the extension approach the actual link can be delivered in two ways. Either as part of the resource, which is, in this case, the provenance information or as part of the header information.

The second option is again to reference both the provenance store and the provenance contract in the resource, see Figure 3.5. This reference would need to specify that it references provenance information for the other provenance store and not for the original resource. This has the nice advantage that it stays backward compatible to already existing provenance clients since they can simply ignore the additional information provided by the custom tags and will still be able to query the provenance information from the store without the need to be able to interpret the new tag. Also, the provenance stores will not need to change how they handle provenance records since the linking information is managed by the original resource provider.

Auxiliary Metadata

In case the provenance hash is interpreted as metadata, we can simply add the necessary information about where to find and how to query the actual provenance records to that metadata, see Figure 3.7. The client querying the provenance contract would then be responsible for interpreting that metadata and querying the actual provenance from the provenance store. This is one of the easiest ways to establish the link between provenance hash and provenance records. It has also the side effect that it would behave the same way as the on-chain model. By only having to solve how the resource links to the provenance contract. On the downside, already existing clients will not be able to find any provenance information since the query would depend on the metadata. The obvious alternative is once again referencing both stores in the resource as already seen in Figure 3.5.

As we can see, the issue of linking the provenance contract to the resource stays the same as discussed for the on-chain model at the beginning of this section. This holds true independent of where and how we represent the information which is responsible for linking a provenance hash to the according provenance records.

3.1.5 Mix-Chain Interpretation and Referencing

Until now, we considered mainly on- and off-chain storage models in our discussion. This is mainly due to the fact that we can use the solutions to those also for the issues around the mix-chain storage model. Let us recall that the core idea behind mix-chain is that part of the provenance records are public as in an on-chain model and part of them are private as in an off-chain model. Now we could either treat the whole provenance records of a mix-chain model as one set of records which would need careful handling of the records and linking between the on- and off-chain records by the querying client application. Or, we could treat the on- and off-chain records as records from two different provenance stores. In this case, the used provenance model would be responsible for combining the provenance records to a complete picture since querying different stores for additional provenance information should be envisioned by the provenance model, as for example by the W3C PROV recommendation [10]. Furthermore, it would allow us to simply reuse the solutions for handling on- and off-chain models without the need of introducing further concepts.

3.1.6 Summary

In this section, we discussed four important things. First, we discussed how provenance information can be represented in the blockchain. Second, we discussed how provenance information can be queried from the blockchain. Third, we discussed how provenance information in the blockchain can be interpreted. And fourth, we discussed how provenance information in the blockchain can be linked to the original resource. As we saw, all these issues can be solved in different ways and these different solutions are often freely combinable. This means it is hard to find a common approach that fits many or even all imaginable use cases. Thus there will always be new approaches using some different component or trying to improve a certain aspect. For example, by using a different model or a different encryption algorithm. Since one of the goals of this thesis is to achieve searchability through generalization we decided to create a model agnostic solution. As we will see in Chapter 4 and Chapter 5, our solution can provide searchability while allowing different use cases to use different models, algorithms, or even storage strategies. This means we are allowing users to implement on top of our solution, extensions as required for their own use cases effectively enabling them to implement all the different strategies as discussed in this section.

The only aspect where we are not model-agnostic is identifying resources by URIs. This approach is directly related to the W3C PROV recommendation, as discussed in Section 2.1.2. However, as we will see in Chapter 4, this is mitigated by the architecture of our solution allowing users to implement a different way of identifying resources.

3.2 The Duplication Issue

In the previous section, we introduced the searchability issue and discussed some of the reasons responsible for its existence. In this section, we will introduce the duplication issue. We will introduce the issue by a simple example out of the viewpoint of the resource producer, then present some scenarios out of the viewpoint of attackers, and finally introduce some options for solving the issue.

3.2.1 Duplicating Blockchain-backed Provenance

Any information that gets written to the blockchain cannot be deleted or manipulated later on. Take a scientist for example. Suppose this scientist is doing an experiment and the software used is provenance-enabled and tracks the provenance of the experiment, see Figure 3.8. Once this provenance information is in the blockchain, it can not be manipulated anymore by said scientist. So if any colleague queries this provenance

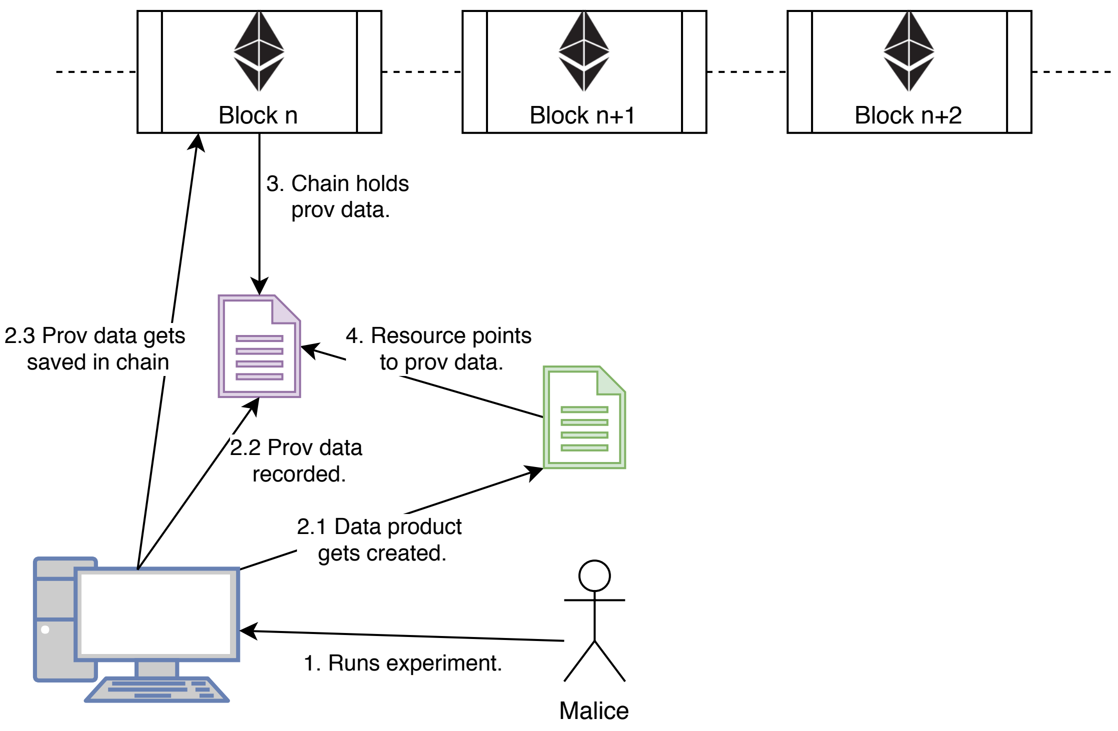

Block n
Block n+1
Block n+2
3. Chain holds prov data.
2.3 Prov data gets saved in chain
4. Resource points to prov data.
2.2 Prov data recorded.
2.1 Data product gets created.
1. Runs experiment.
Malice

Figure 3.8: Storing provenance data in the blockchain.

information, to build trust in the results produced by the scientist, this colleague can easily verify the integrity of the provenance information.

In retrospective, the scientist realizes that the results are not what was expected and wants to twist them just a little bit. However, the provenance data is already in the blockchain and can therefore not be changed. The easiest solution for the scientist would be to generate new fake provenance data, by running the experiment again with the necessary changes and save it also into the blockchain. By then referencing only the newly generated provenance data in the resource, it would be hard to suspect any manipulation, see Figure 3.9. Although the correct data is not lost, it would be very hard to find it, which would essentially mean that one has to scan the entire blockchain to be able to map this provenance information to the experiment in question.

Although this could be remotely possible with complete, un-encrypted provenance information in the blockchain, it would just take time. If one starts considering encrypted, off-chain, or mixed provenance information as discussed in the previous sections, it can easily get nearly impossible to prove that any given encrypted provenance information or hash belongs to this certain experiment in an earlier form. This problem is also independent of how the provenance information is stored in the blockchain. Since it is equally easy for the scientist to create a new transaction as it is to create a new smart contract or a new identity on any given public chain, including Ethereum.

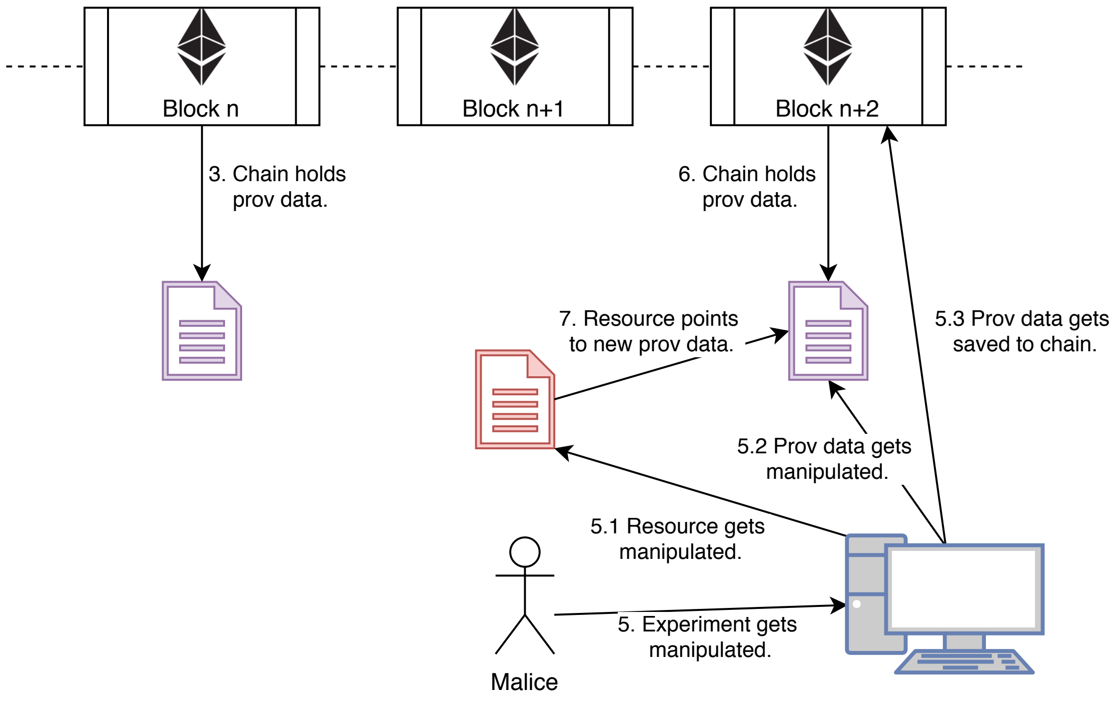

Block n
Block n+1
Block n+2
3. Chain holds prov data.
6. Chain holds prov data.
7. Resource points to new prov data.
5.3 Prov data gets saved to chain.
5.2 Prov data gets manipulated.
5.1 Resource gets manipulated.
5. Experiment gets manipulated.
Malice

Figure 3.9: Duplicating the provenance data.

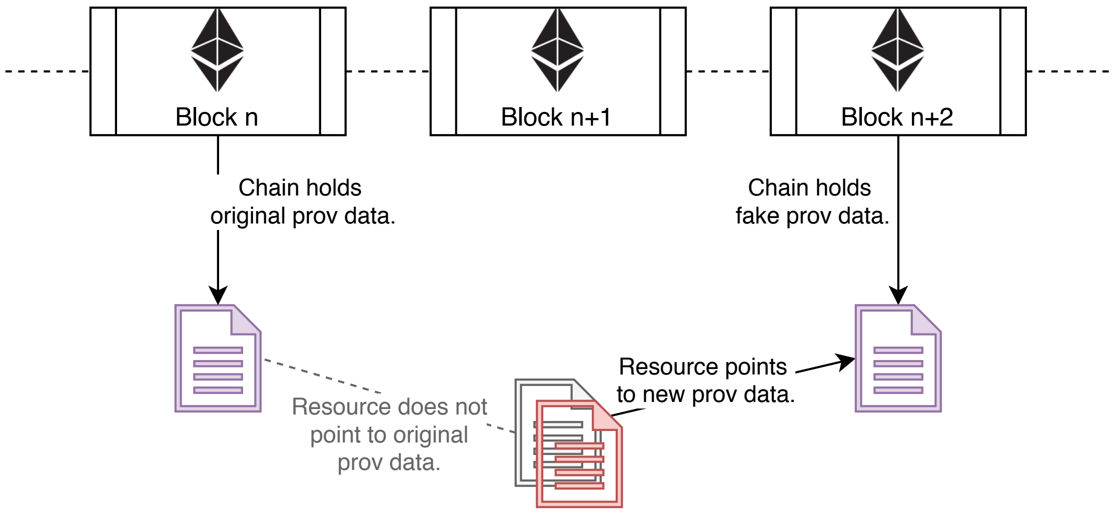

Block n
Block n+1
Block n+2
Chain holds original prov data.
Chain holds fake prov data.
Resource does not point to original prov data.
Resource points to new prov data.

Figure 3.10: Hiding original version of resource.

Figure 3.10 depicts the result of the above-mentioned problem of provenance data duplication. The original version of the provenance data is stored in some block. The maliciously modified version of the provenance data is stored in some later block without any reference to the old provenance information. The generated and manipulated resource is only pointing to the newly-generated provenance data and not to the old one. This way, the originally-generated resource gets hidden behind the modified one since there is no obvious reason to believe there ever was another. This problem can even occur in a non-malicious scenario. Due to error, it could happen that a new version does not get correctly linked to the old version, during the recording of the provenance data and thus part of the provenance information gets lost in the big amount of transactions and information saved on the chain.

If we take accidents and buggy software aside, the provided example lacks in severity since it is easily arguable that a malicious scientist could prevent the linking of the provenance information to the blockchain until the results are satisfying in the first place. It is obvious that the provenance-recording party is almost always able to manipulate the recorded provenance before it gets saved in a store of any kind. This is a commonly accepted assumption in the related work [4], [35] and can easily be supported by a simple example. Assume you have a provenance recording client with an open protocol, as we do, it is possible to write a client that uses the same protocol however shows the provenance information first to the user and waits for approval before saving the provenance data to the store. What we want to achieve with coupling to the chain is to make the provenance, once properly recorded, hard to nearly impossible to change. Still, the previously described scenario is useful as a simple introduction into the problem of provenance data duplication. In the next example, we will show how this issue can be used by an attacker who is not the producer of the data.

3.2.2 Attack By Duplication

Let us consider it from the viewpoint of a malicious long-term data storage provider. Suppose the original creator of the scientific results had to take them offline for some reason and there exists a secondary source for this data. A non-malicious provider would now claim that the resource provided is still the same as the original resource by pointing to the provenance of that resource to prove it. A malicious provider could do the same with some altered provenance information that backs a slightly manipulated resource. By carefully comparing both resources and the provenance data, a domain expert could probably derive which one is the wrong provenance information due to inconsistencies of timestamps or other metadata. However, this is only possible if a client who queries this provenance data is aware that there are two different providers for the resource. What if both providers are malicious or there is only one provider or the client is only able to locate one of the providers? Then there is no way of being certain that the provided resources are indeed the same as the original ones. The problem boils down to the same one as before that someone is hiding the original provenance data by simply not referencing them but referencing a duplication of them. Like in the last example, this can also be done with provenance contracts by simply uploading a new contract. There are a few possible solutions to this problem which we will discuss next. All of them are based on the idea of restricting the field of valid provenance information. In general, this means that like not all provenance data stores are given the same amount of trust, we can say that not all provenance references in the blockchain are given the same amount of trust.

3.2.3 Towards Solving The Duplication Issue

We will discuss now two simple and straightforward approaches to solving this issue.

Black-listing is one of the approaches possible to this issue. In the domain of blockchains, this would mean black-listing single identities and thus banning malicious users from creating provenance information for a certain resource. However, this proves difficult since especially in public chains it is very easy to create new identities or smart contracts and start all over with the distribution of false provenance data. Much more critical is that black-listing does not solve the issue of malicious parties trying to hide the original provenance information. So, to be able to black-list an identity on the blockchain, one first would have to identify that the provenance data provided by this identity is indeed a duplication of some other resource's provenance data which as discussed previously can prove to be difficult. Another issue with black-listing is how to propagate this information to all clients. Since in any blockchain-based approach the actual querying happens on the client side, this black-list would have to be distributed in some manner to the clients. One such possibility would be to use again the blockchain to save a list of black-listed identities in some smart contract however then we would open another attack vector for malicious users to start black-listing non-malicious users. We could try to solve this issue by employing some kind of distributed consensus mechanism, as used by the blockchain itself. Javaid et al [44] tried to use a consensus mechanism to accept or decline provenance in their solution however their mechanism has a tendency of favoring a negative outcome. In the end, the only reliable approach to black-listing would be to use a per-client approach where every client decides for each duplication which provenance to trust in the first place. However, this does not solve the main issue of finding the duplication in the first place.

White-listing seems to be the more promising solution to the problem. Instead of black-listing identities, we simply declare which identities are trusted to provide correct provenance information for a certain resource. A major difference comes from the idea that even if a malicious identity adds itself to the white-listed identities, it can not hide away the original provenance information since the original provenance information will be also in the resulting list of white-listed identities. Thus, a user that queries the provenance data will see that there are two different provenance information collections arguing for the same resource and will be able by analyzing the data carefully to find out which of the provenance data is supposed to be right. In case of two different provenance information collections together with two resources which are slightly different but each claiming to be the original again a person from the domain of that resource should be able to repeat the experiment, use other means to analyze which resource is malicious, or even determine if these are on each other dependent versions in case of an honest mistake.

Although white-listing identities would solve the basic issue of hiding provenance data, this can quickly get out of hand. For instance, let us take the simple provenance example as provided in Section 2.1.1. Consider the government environmental report. The government provides provenance for this report and white-lists the identity used to record the provenance information. However, this report could be the result of a long collaborative process between different internal and external actors thus resulting in far more than one identity that has to be white-listed. Similarly, the blogger who used the report in his blog entries will also have to whitelist his identities. Remember, the provenance data generated by those different actors is related since ultimately the blog entries are based on the government report. So, someone who wants to query the provenance of the blog will ultimately be interested in the provenance of all resources involved. To acquire this provenance data a user has to go through the transactions of all white-listed identities of all involved actors. As can be seen, the more actors are involved, the more identities will need to be white-listed to be able to express and find all of the provenance data.

Furthermore, there is the issue that white-listing, in this case, is openly announcing which identities are trusted by some specific actor. In our example, the government would simply announce its identities on some Web page or through some Web service. Even if the government is white-listing by using some smart contract, the address of this smart contract would have to be announced to the public. In case of the government, this should not be a big issue, since the government usually has some kind of infrastructure at its disposal. However, the white-listed identities of the blogger could more easily get lost after some time has passed. As can be seen, white-listing does not entirely solve the issue of hiding provenance since the list itself could in theory still be forgotten or not found.

We can solve both of these issues in a very convenient way by using only smart contracts as provenance identities and introducing the notion of contract linking both of which we will discuss in the next section.

3.3 Provenance Networks

In this section, we will first introduce concretely what a provenance contract is in our case. Before, we show how we can use linking to create networks of provenance contracts in the blockchain. We will then discuss how this provenance networks can help us to solve the searchability issue as well as the duplication issue.

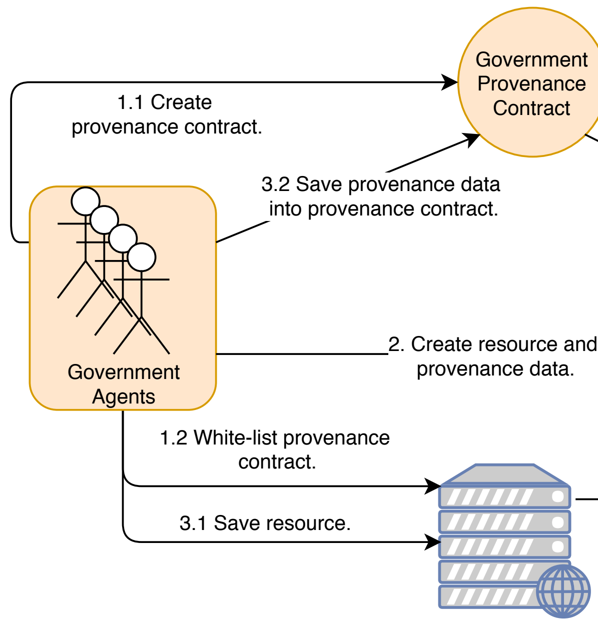

Government Provenance Contract
1.1 Create provenance contract.
3.2 Save provenance data into provenance contract.
4.2 Holds provenance data.
Government Agents
2. Create resource and provenance data.
Government Data
1.2 White-list provenance contract.
3.1 Save resource.
4.1 Holds resource.

Figure 3.11: Government saving all provenance data into one contract.

3.3.1 Provenance Contract

We define that a provenance contract is a smart contract which is written to store provenance information. Actors that have the necessary privileges can add provenance data concerning certain resources to a provenance contract. The provenance contract has no influence on the storage model or storage state, as presented in Section 3.1.1. Provenance contracts are simply the place where the data is stored. More details on the technical view of provenance contracts are given in Chapter 4. In this section, we will go into the details of how those contracts work on a conceptual level.

We ended the last section with the claim that white-listing provenance contracts is better than white-listing identities. Recall the example with the government report and the blog entry. Now, instead of white-listing the identities used to create the government provenance data, and the crawling through the respective transactions, the government could simply save all the relevant provenance data in a provenance contract and white-list the contract as a whole, as shown in Figure 3.11. It would not matter how many different identities were used to create this provenance data to the contract since the contract as a whole is white-listed in the end and contains all the necessary provenance information for the resource in question, in one single place. This way the contract itself becomes, to a certain degree, a representation of the government identities. In future figures, we will omit therefore the real identities and also a lot of the detail around the resource creation. We will just assume this default workflow as shown in Figure 3.11.

The blogger who is writing an article about the government data could now instead of white-listing the necessary identities simply add the provenance information about that

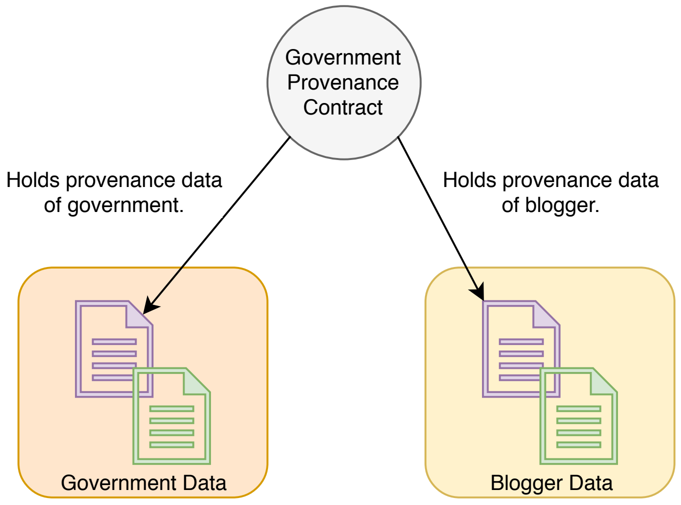

Government Provenance Contract
Holds provenance data of government.
Holds provenance data of blogger.
Government Data
Blogger Data

Figure 3.12: Government contract becoming a public provenance contract.

article to the contract of the government directly. Someone who now wants to query the complete provenance data has simply to read out the one contract that is white-listed by the different actors. This way the first issue, regarding the flood of identities, can be solved in a rather convenient way by using open, public provenance contracts as seen in Figure 3.12. It introduces, however, a few issues on its own. First, the blogger would need to split up the own provenance data over different contracts. For example, the blogger would need to write the provenance of different articles into different contracts, since the provenance of another article that has nothing to do with this government report would not thematically fit into this provenance contract. Second, the government provenance contract would become a public contract representing identities from multiple domains. Finally, everybody could decide that they want to write all their provenance information into the same contract likely leading to bad query times for clients and much worse no separation of governance. Separation of governance is simple to explain. Although, the government contract is technically still owned by the government, by allowing everybody to use it, it has become a public contract which leads to a few simple issue, for example, who has to decide which actors are allowed to use it and which are not, and would the government be allowed to disable the contract or not.

The second issue, regarding hiding of provenance information, has hereby been improved since now multiple different actors are providing a reference to the same smart contract. However it has not been entirely solved, yet. There is still a certain likelihood that all of these actors will stop referencing this smart contract. To solve this, we will now introduce the concept of contract linking.

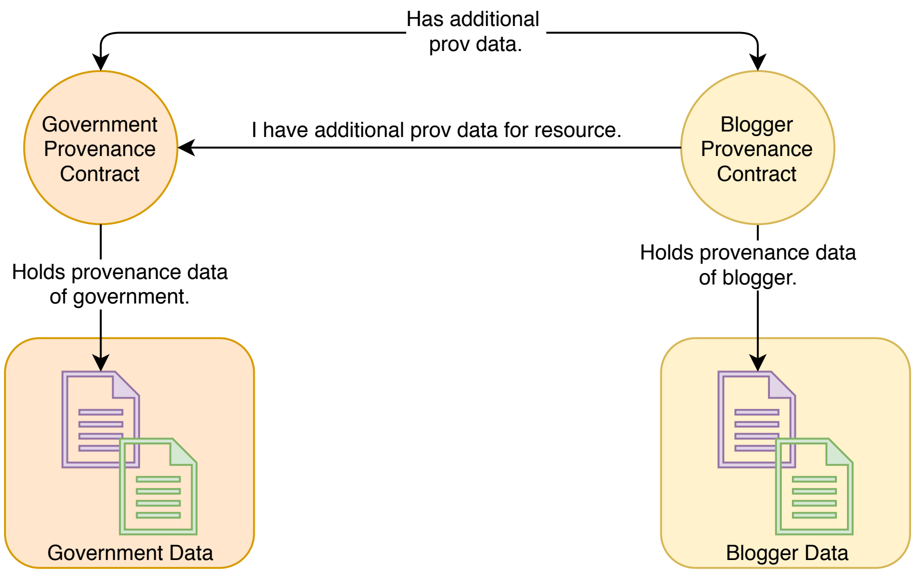

Has additional prov data.
Government Provenance Contract
I have additional prov data for resource.
Blogger Provenance Contract
Holds provenance data of government.
Holds provenance data of blogger.
Government Data
Blogger Data

Figure 3.13: Resource-based linking between the provenance contracts.

3.3.2 Contract linking

If the government and the blogger have their own provenance contract saving their part of the information regarding a specific resource, we would be back at the beginning simply on a higher abstraction level replacing identities with provenance contracts. However, if we allow the different contracts to know each other, we are basically letting them behave as one contract. In other words, the provenance contract of the blogger would announce to the provenance contract of the government that it has further information regarding a certain resource. The result would be that each of the contracts would know that the other contract has additional information about a resource they have provenance data about, see Figure 3.13. When some client now queries either of the contracts, the client would find the provenance data saved in that contract and the links to the other contracts containing provenance data about that resource. It then could query those contracts and so on. This reassembles pretty accurately the way that the W3C PROV model allows provenance storages to inform about other provenance storages containing more data, as shortly mentioned in Section 2.1.2.4. We will call this kind of contract linking *resource-based* contract linking, based on the fact that contracts only know of other contracts that are holding provenance information about the same resource.

Furthermore, we introduce the idea of resource-independent linking or *trust-based* contract linking. This way we could allow contracts, in general, to link to each other even if they do not share provenance information about some resources. Now, we can create a network of provenance contracts in the blockchain, similar to trust networks as shown in Section 2.3. For this, consider the following example. Take a provenance contract that was deployed by TU Wien. We can easily prove by cryptographic means that TU

Wien owns that contract and we also can see that researchers at TU Wien are using this contract to store research results on it. Now, TU Wien announces quite openly that only provenance information retrieved from this contract can be viewed as official and trusted provenance of TU Wien and to be created by researchers working for TU Wien. This contract would get quite crowded by default since all the different scientific departments of TU Wien would write their provenance into this contract. At some point, some of the departments, for example, the Faculty of Informatics, could decide to create their own provenance contract to improve query times, see Figure 3.14. Since TU Wien's policy, however, allows only provenance information from the main contract, the Faculty of Informatics could link both contracts with a *I trust this contract*-policy. This way, TU Wien itself does not have to do anything else than announcing the main contract and the different departments could announce their own sub-provenance contracts. Any query to the main contract would also reference the department contracts and any query to a department contract would also reference the main contract. As we discussed in Section 2.3, trust networks are directed graphs. Thus trust will only get propagated properly if both contracts in question link each other as we will see later in this chapter. From here on, it would be up to the clients how much of the provenance network to query. Furthermore, we know from Section 2.3 that trust networks are not only directed graphs but also weighted. Depending on the trust model that is used, this would also allow for expressing distrust towards a specific contract.

If we combine both approaches of resource- and trust-based linking, we get a network of provenance contracts, or simply a *provenance network*. By combining both examples to one, as we can see in Figure 3.15, we suddenly are able to see the complete provenance network of TU Wien. Knowing any of those contracts would reveal the whole TU Wien provenance network. This makes it rather hard to miss part of the provenance since it is interlinked in a complex manner spanning over many organizational borders. This way, even if someone tries to duplicate provenance information, this attacker would need to compromise the network of TU Wien by linking to it and announcing the malicious provenance information. This, however, would not hide the original provenance but just provide a un-hidden duplication which could be dismantled by domain experts as such. To successfully perform a duplication attack, the attacker would need to succeed in duplicating the whole network which gets more difficult with a higher amount of participating domains, contracts, and clients. The whole attack has thus been reduced to a spamming issue which is naturally restricted in a blockchain-based environment since malicious users would need to pay for every spam entry real money. The issue gets further mitigated through access controls as we will see in Section 3.3.4.

3.3.3 Trust Propagation in the Provenance Network

In the last section, we introduced the concept of resource- and trust-linking. Since we introduced trust-linking, our provenance network becomes comparable to a trust network. Thus, we are now able to work with trust propagation techniques as discussed in Section 2.3. Until now, we build provenance networks which could be represented

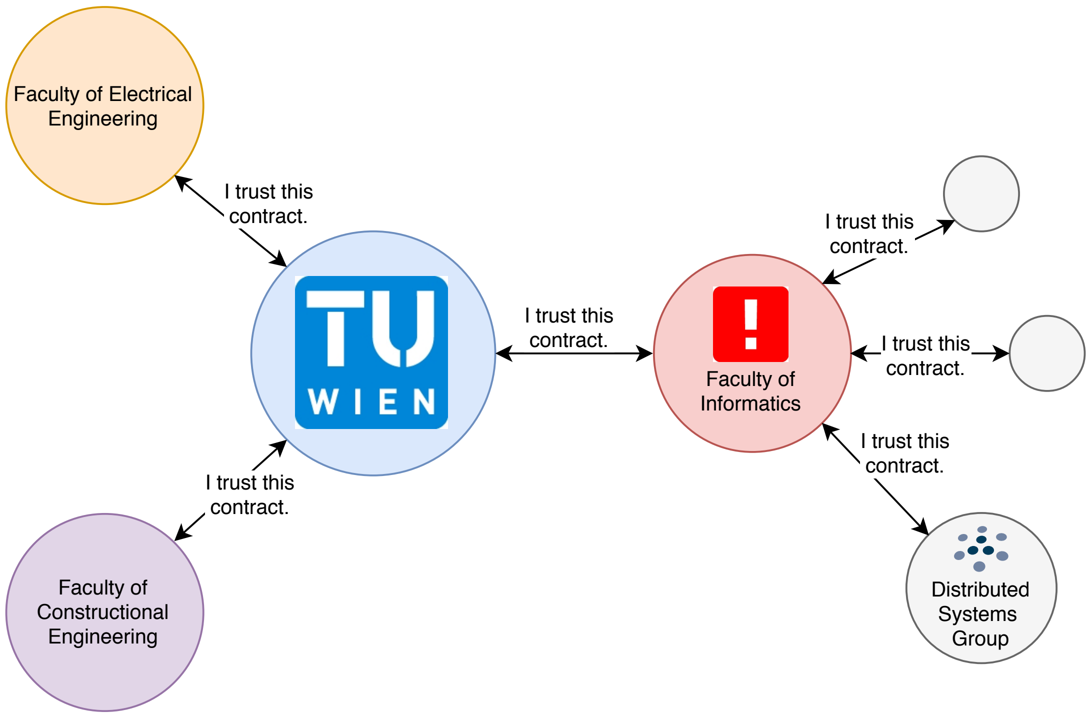


Figure 3.14: Trust-based linking between the provenance contracts of TU Wien.

as graphs where the provenance contracts are nodes and the links are edges. What we did not discuss, yet, is the direction and the weight of such a link. As we can recall from Section 2.3 trust networks are directed, weighted graphs. In our case, the different contracts correspond to the vertices of a trust graph and the links between contracts correspond to the edges in a trust graph. More precisely we define that a directed, weighted edge as known from trust networks [29] is a unidirectional link between two provenance contracts. As you can see in Figure 3.15, up to now we used bidirectional links in all of our examples. In the following, we will define in more detail the difference between unidirectional and bidirectional links.

Unidirectional Links

Since unidirectional links are directed edges, they have an outgoing side and an incoming side. In our case on the outgoing side of the link is the contract which holds the link. On the incoming side is the contract that gets linked. This means that the linked contract does per definition not know of the contract linking it unless informed by it. If we look at our example again, see Figure 3.16, a trust link from the blogger's contract to the Faculty of Informatics' contract is saved in the contract of the blogger. From the viewpoint of the Faculty of Informatics' contract, the blogger contract does not exist. Only after adding the bidirectional resource

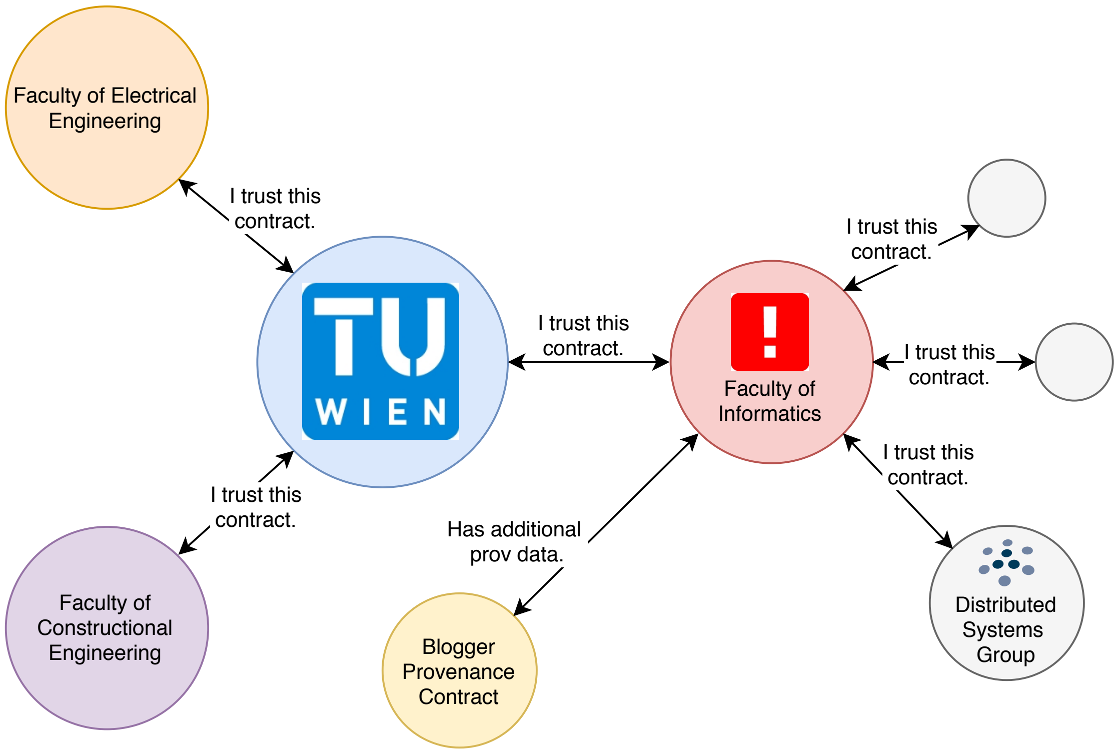


Figure 3.15: The provenance network of the TU Wien.

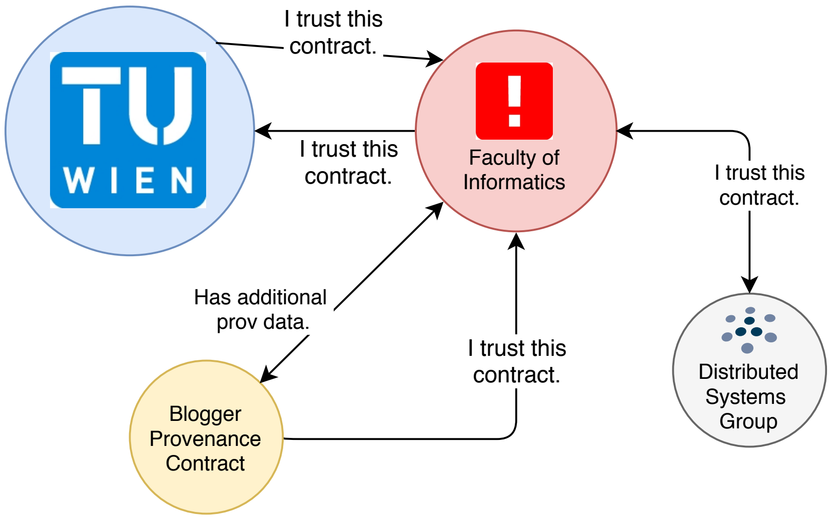


Figure 3.16: Link directions.

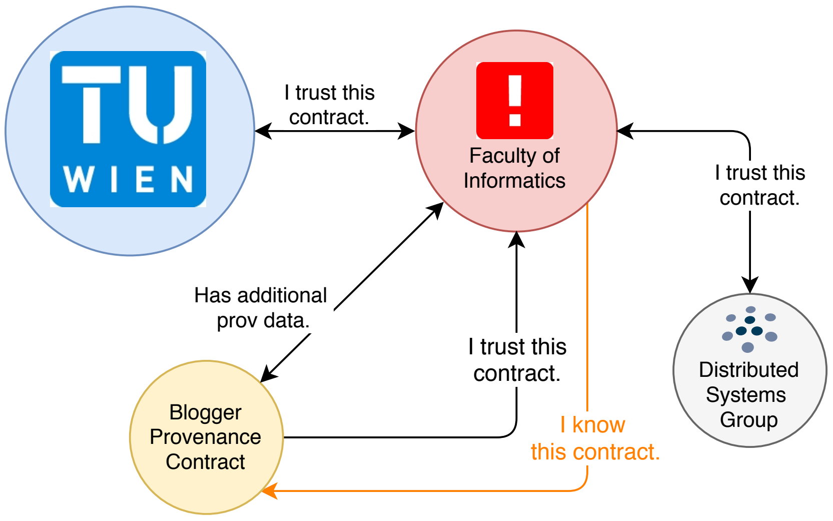


Figure 3.17: Weighted links.

link, the Faculty of Informatics becomes aware of the blogger's contract.

Bidirectional Links

We define a bidirectional link as two opposing, unidirectional links with the same weight. Meaning, there are two unidirectional links one held by each contract involved in the link with the same weight. For example, see Figure 3.16, the TU Wien main contract is hierarchically the more important contract than the contract from the Faculty of Informatics. A link from the contract of the faculty towards the contract of TU Wien is saved in the faculty contract and is a perfectly legitimate link although without a link from the TU Wien contract to the faculty contract there is no propagation of trust towards the faculty's contract. Bidirectional links when possible are to be preferred over unidirectional since they strengthen the overall provenance network in regards to duplication attacks.

We just defined that bidirectional links are to be preferred, however, not every contract in the provenance network can necessarily trust any other contract in the network. To solve this, we have the link weights. Until now we used only one weight, the *I trust this contract*-policy. However, in reality, we can have arbitrary link weights. For example, the blogger can indeed set up a link towards the Faculty of Informatics and trust the faculty to produce legitimate provenance data, however, the Faculty of Informatics can not necessarily trust the blogger's contract. Thus, as shown in Figure 3.17, the faculty might response with a *I know this contract*-policy towards the blogger's contract. This way the faculty can support a bidirectional link towards the blogger without having to trust the blogger's provenance data. Any client querying the network can decide on its own it the provenance of the blogger should be trusted. The same way, we can use links to also state distrust towards some other contract. In fact, we can implement any weights-based trust model, for example, the one discussed in Section 2.3. In this figure, we can see another interesting side effect of having resource- and trust-links. Although the Faculty of Informatics states that it does not trust the blogger's provenance, it allows nevertheless the bidirectional resource link. This way, it allows clients which decide to trust the blogger to still find all the relevant provenance information when querying.

This is important for understanding how trust gets propagated in such a provenance network. It depends on the kind of link, trust- or resource-based, and on the weight of the link to give meaning to the link. And finally, it depends on the direction of the link to show how this meaning is propagated towards other contracts. This, however, makes it also important to understand that it depends on which contract you are looking into the network. From the view of the blogger's contract, the whole network is trustworthy. As from the view of TU Wien, there is a part of the network for which we know that it has additional provenance data but we can not state if this data is reliable.

One last thing we have to talk about are organizational link policies. Organizational link policies are about how TU Wien, for example, allows its organizational child contracts to link external contracts. The DSG contract, for example, is part of the TU Wien organization and can be linked by the faculty contract without hesitation. However, the blogger's contract is not part of TU Wien and the rules that the faculty has to follow when considering how to link this contract are the link policies as defined by TU Wien. In the following we will describe two very simple policies that can be followed:

Open Policy
This means that the linking contract does not care about the security settings of the contract which is being linked. If this was the setting in the TU Wien main contract, this would mean that the contract of the faculty would have the freedom to set up its own policies and link further contracts any way they like. One would have first to analyze the policy of the faculty contract to establish what meaning a link has from the viewpoint of the faculty.

Restrictive Policy
This means that linked contracts have to follow the linking policies as defined by the parent contract. In the example of the TU Wien, this would mean that the faculty contract has to have the same linking rules and behavior as the main TU Wien contract which would automatically propagate the link meaning of the main contract to the links of the child contracts.

There is a variety of ways how linking policies could be enforced. Access restrictions are one very simple example. By allowing only a certain group of administrators, for example, to link new contracts, link policies could be enforced entirely in a non-technical way. Another option is to provide some mechanism which would automatically check policies which are set on the corresponding contracts.

Let us consider the contract of the DSG one more time and let us assume that TU Wien uses a restrictive security policy. Any user that wants to check if it is trusted by the TU Wien main contract would need to either follow outgoing links up from the DSG contract until the main contract is found, *bottom-up*, or outgoing links down from the TU Wien main contract until the DSG contract is found, *top down*. Of course, bottom-up would only work if TU Wien follows a strict bidirectional linking strategy including the same weights for contracts within the organizational boundaries of the TU Wien.

3.3.4 Provenance Contract Access Security

Without any access restrictions, the notion of contract linking we introduced would open the possibility for anyone to link their contract to the official TU Wien contract. This can be equally good as also bad. In case of another university, on the one hand, which is cooperating with TU Wien, this would mean that the two universities are creating among each other trust and automatically strengthen each other's contracts. Especially for smaller domains like the blogger, for example, this can be very important since if his site goes offline also the public announcement of his contract goes offline. But by having it resource- or trust-linked to other contracts, its contract would not disappear from the network of provenance contracts. A client which is searching the whole provenance contract network would ultimately also find the provenance information saved in the blogger's contract. In case of a malicious attacker, on the other hand, this openly available linking of provenance contracts would allow spamming the provenance network with contracts containing wrong information which would lead to the necessity for users to do filtering which provenance information is right and which is wrong when querying. Either way, the problem of duplicating provenance would be solved since the basic assumption would be that anyone linking his contract to a chain of contracts obviously does not try to duplicate any other provenance in this chain, since the duplicate would be easily discovered and domain experts should then be able to identify the original provenance information.

The issue of writing wrong provenance information into a store is a different thing. In theory, it is also given in classic implementations of provenance stores. Since anyone who gets access to a store can also freely add provenance information to this store. However classic implementations often have some kind of access control mechanism to solve this issue, as can be commonly seen in related work [12], [37], [44]. Authentication allows the storage providers to filter out all the provenance added by a malicious user once it is identified. Something similar could be implemented in a blockchain-based solution rather easily. Following this is a list of policies that can be used to construct more complex strategies for establishing access security to provenance contracts. However, keep in mind that those strategies are always towards write-access restriction. For read access restriction, the only possibilities are encryption or off-chaining, as we discussed in Section 3.1.1, since data on the blockchain is always publicly available.

Not at all.
In this approach, we are allowing the client-side to make expert decisions about the provenance and determining which provenance is the right one. A domain expert, given the resource and the provenance data, should be able to identify which provenance data is the fabricated one by analyzing it. We are mentioning this approach for the sake of completeness. In practice, it is useless since any resource could be basically spammed with wrong provenance data until it would take a huge amount of time to identify the correct provenance information. Furthermore, we are working on this approach under the assumption that domain experts will be querying the provenance data.

Restricted contract linking.
By restricting the resource-, trust-based, or both methods of contract linking we can avoid that a contract gets linked to contracts carrying malicious provenance information. The restriction could be done in two different ways. One way is that the contract owner restricts which identities are allowed to link contracts, which is basically identity white-listing as discussed in Section 3.2.3. The other way is that every linking request has first to be approved by the contract owner. The disadvantage of restricting contract linking is that the network around a particular contract will grow slower making it more vulnerable to the duplication attack.

Restricted write access.
Restricting write access is similar to restricting linking with the difference that in this case, we are talking about restricting who is allowed to write new provenance data to the contract instead of who is allowed to link contracts. The way this can be done is essentially the same as with restricting contract linking, either by request confirmation or by identity white-listing, as discussed in Section 3.2.3. This is useful when a domain holder wants to make sure that on his contract there is truly only legitimate provenance information saved.

3.3.5 Provenance Networks Summery

In this section, we introduced provenance contracts and provenance networks. Provenance contracts being simply smart contracts that are used by actors to store provenance information, and provenance networks being trust networks of linked provenance contracts. We then continued by discussing in more detail how this contract linking is done and what has to be considered, including link directions, weights, and policies. In the end, we discussed access restrictions for provenance contracts and how they help to secure the contracts.

To summarize all properly we will give a final example demonstrating different provenance contracts for different purposes. We will demonstrate two different strategies which we will call the *Organizational Border*- and the *Search Contracts*-Strategy.

Organisational Border-Strategy

Looking at the TU Wien example again, it is possible that TU Wien does not want anyone to be able to trust-link provenance contracts to their provenance network. To avoid this, they would simply restrict trust-linking. This way, whoever wants to link a new contract to the original TU Wien contract will have to wait until they authorize this linking. This would likely result in a contract network around the main TU Wien contract that represents the organizational structure of TU Wien since the different faculties would likely provide their own contracts and link them to the main contract to separate different domains, as shown in Figure 3.14. This way, the main contract would simply serve as a trust propagator, propagating its trust to all department contracts and thus authenticating them as trusted TU Wien provenance contracts. Furthermore, TU Wien could restrict the writing of provenance to TU Wien personnel by restricting write access. The result of these two restrictions would be that there are only trust-linked contracts in the TU Wien provenance network that belong to TU Wien and there is only provenance data saved in the network that is from TU Wien personnel.

To make scientific collaboration easy, TU Wien would allow for unrestricted resource-linking of contracts to their network. Furthermore, since the trust-linking is restricted, it is easy to establish for any querying client if a contract is coming from TU Wien or not and can, therefore, establish the trust in the queried provenance data.

Search Contract-Strategy

Let us have a look at search engine providers, for example. If one wants to implement a provenance search engine, this provider would need to allow public trust-linking so that anybody can add their provenance contracts to the search engine contract. For the search engine provider, it is not necessary to allow saving provenance on the contract itself so restricting the resource-linking and the write access to the contract could be a wise move however not strictly necessary. Of course, in this case, the trust-link is not weighted as a trusted link but as a known link and thus representing all the contracts known by the search engine.

In Figure 3.18, we can see a simplified example of the two strategies glued together in a world view of different domains. In this example, we see how the Faculty of Informatics and the search engine contract use the link weights to express different trust levels towards the contracts they link. In case of the search engine, all links are with a lower weight making the search engine contract simply a place to look up for the contracts holding the actual provenance information.

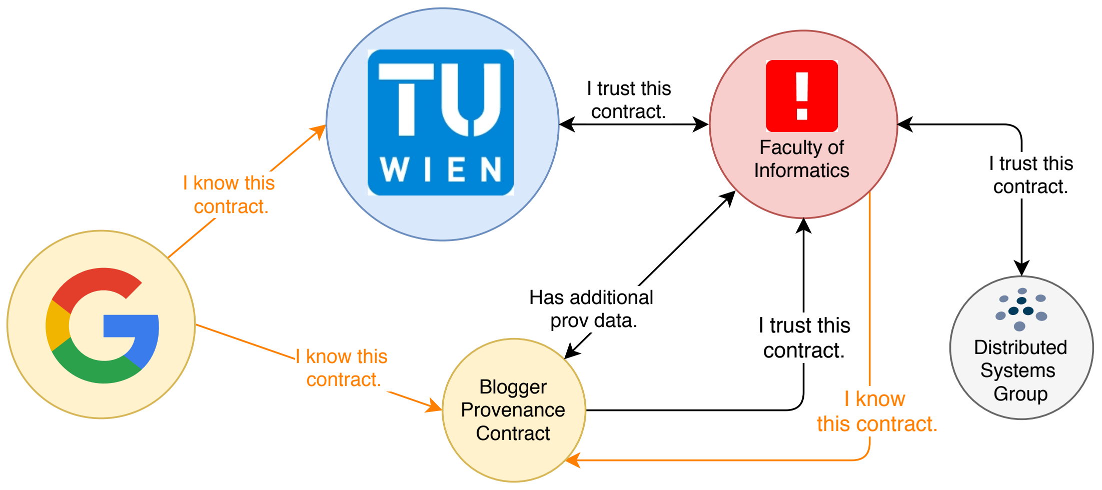


Figure 3.18: A simplified world view.

3.4 Summary

In this chapter, we introduced two major open problems, the searchability issue, and the duplication issue. We defined the searchability issue as a problem to find provenance across different models and solution domains. We continued by explaining why there is such a big variety of different solutions for data provenance especially in regards to blockchain-based solutions. Then, we introduced the duplication issue and explained how a malicious user could use this to hide valid provenance data behind forged provenance data in the blockchain.

Finally, we introduced provenance networks as a valid solution to both issues. Provenance networks, simplified, are trust networks between provenance contracts in the blockchain, with provenance contracts being simply smart contracts which store data provenance. Provenance networks allow us to easily solve the searchability issue by creating a directed, weighted graph, which does not limit its use to a certain provenance model, granularity, or kind of integration with the blockchain. Thus, allowing users to search for provenance across domains, use cases, solutions, and models. Once the data can be found, it becomes the duty of querying clients to make sure that they can reconstruct the provenance data from the way it was saved, allowing for broader interoperability between different models and solutions. Furthermore, provenance networks allow us to solve the duplication issue by preventing malicious actors from hiding the original provenance information. Although they are still able to duplicate provenance information, these duplications are now easily findable and can be discarded by domain experts.

In the next chapter, we will discuss in detail how our prototype is implemented and how it solves the design issues discussed in this chapter.

CHAPTER 4

Implementation

In this chapter, we will introduce the general structure of our repository and the most important architectural decisions we took. Our repository is structured into four different projects, namely *ETH*, *React-Client*, *Node-Client*, and *Eval*. Each of them contains one autonomous part of our solution that provides certain functionality to other projects, to end users, or our evaluation. The *ETH*-project contains the backend code, namely all the smart contracts that need to be deployed to the blockchain, some tests for them, and the required configuration. The *React-Client*-project contains a single-page, user interface client which we have created to allow for visually browsing and managing provenance contracts and provenance networks. The *Node-Client*-project is a simple remote client providing an API for the most important operations. And, the *Eval*-project contains a deployment and evaluation helper that is handy when deploying a large provenance network and also automates our evaluation. Figure 4.1 shows how the different projects depend on each other, where they are executed, and which of them expose interfaces.

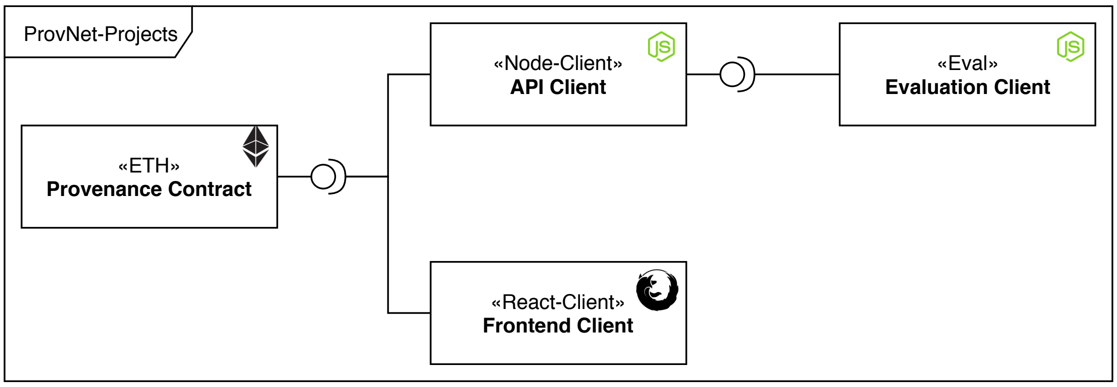


Figure 4.1: A overview of the different projects and where they run.

4.1 ETH-Project

4.1.1 Project Structure

We use Truffle[1] as a support framework for the Solidity development. It is a very active community framework that provides automation for necessary default tasks around the development with Solidity. Truffle provides automated migration, testing, and comes with local Ethereum chains optimized for development.

Adopting the Truffle default structure is a must to be able to develop with Truffle, resulting in five main folders:

build: The build directory holds the compiled contracts.

contracts: The contracts directory is where all the contracts are placed in.

migrations: The migrations directory holds the migration java files. These files are used to tell Truffle how to deploy and link the different contracts. The migration files are executed in the order given by the initial number of the file and they also provide information about the target network allowing to distinguish between test and productive networks.

test: This folder contains the test suites for the contracts. Truffle offers two kinds of tests. You can either write Javascript-based tests or Solidity-based tests. Solidity-based test are contracts that get deployed onto your development or test network and get executed. Every file beginning with T will be interpreted as a test suite and within that test suite every function starting with *test* will be interpreted as a test. Solidity tests are flexible and depending on the exact strategy used by the developer allow for accessing internal functions and data structures of the contracts under test. Truffle supports snapshotting and resetting the development networks if the network itself supports snapshotting. This feature is provided for both Solidity and Javascript tests. However, it is only applied between test suites leaving it to the developer to handle clean up between the single tests which can be rather cumbersome in Solidity. Javascript tests, on the other hand, use the Web3Js script to access contracts deployed on the development or test network. Those tests are run as if they where clients for the deployed contracts and can only access public functions of those contracts. Thus this Javascript tests behave more like integration tests.

4.1.1.1 Test Mocks

With this basic test structure in mind, to be able to properly test we need to write test mocks. These are mock contracts that are not intended for publishing on a productive chain but are only written to be used for testing the Solidity code. These test mocks can

[1]https://truffleframework.com/

be used for multiple things. They are an easy way of cleaning the memory after each test by simply deploying a new mock contract to the test chain. Furthermore, they can be used to expose internal and private functions to the public making it possible to reach them from outside the contract which can be useful for unit-testing. Although it is also possible to reach internal functions from Solidity tests directly, mock contracts are still necessary for the ability to provide clean and separated environments per test.

These mock contracts are naturally a test matter however historically they had to be placed within the contracts directory due to Truffle not finding them otherwise[2]. Due to this issue, many community projects still keep their mocks in the *contracts* directory. Since by now it is possible to keep the mocks in the test directory, we will diverge from the community's behavior at the cost of risking to need to refactor in case the Truffle team introduces breaking changes. The upside is that we clearly separate the mock contracts which are really only test helpers from the productive code.

Another decision that arises from the need to use mock contracts is that we will keep all our tests as Javascript tests. Although this is not strictly necessary since all our mock contracts can be used from either test environment, once deployed, and are written in a general manner. However, there are some advantages that arise from using the same environment for all tests:

- We can share test helpers between unit tests and integration tests, which rids us from the need to write common assertion helpers in two languages.
- It allows us to make use of a more mature test environment.
- All tests have the same demands towards the build environment and deployment framework.

4.1.1.2 Contract Deplyoment

Truffle offers its own migration process that can take care of contract deployment. This process can be configured with the help of so-called migration files, which are stored in the migrations folder as mentioned above. When it comes to deploying contracts, Truffle does two things. It compiles the contracts to EVM byte code with the help of the Solidity compiler and it executes the migration scripts. Building contracts, however, does not necessarily produce deployable byte code. In case of complex contracts which rely on external libraries or contracts, the compiler can not know the addresses of these dependencies. So, it leaves placeholders in the byte code which have to be filled by the actual addresses at a later point. The task of filling the placeholders falls to the linker which is where the Truffle migration files come into place. The migration files define the order in which the contracts have to be deployed and they also define which contracts have to be linked to which other contracts. This way Truffle abstracts and simplifies the process of linking contracts. The addresses of the deployed contracts also get saved

[2]https://github.com/trufflesuite/truffle/issues/141

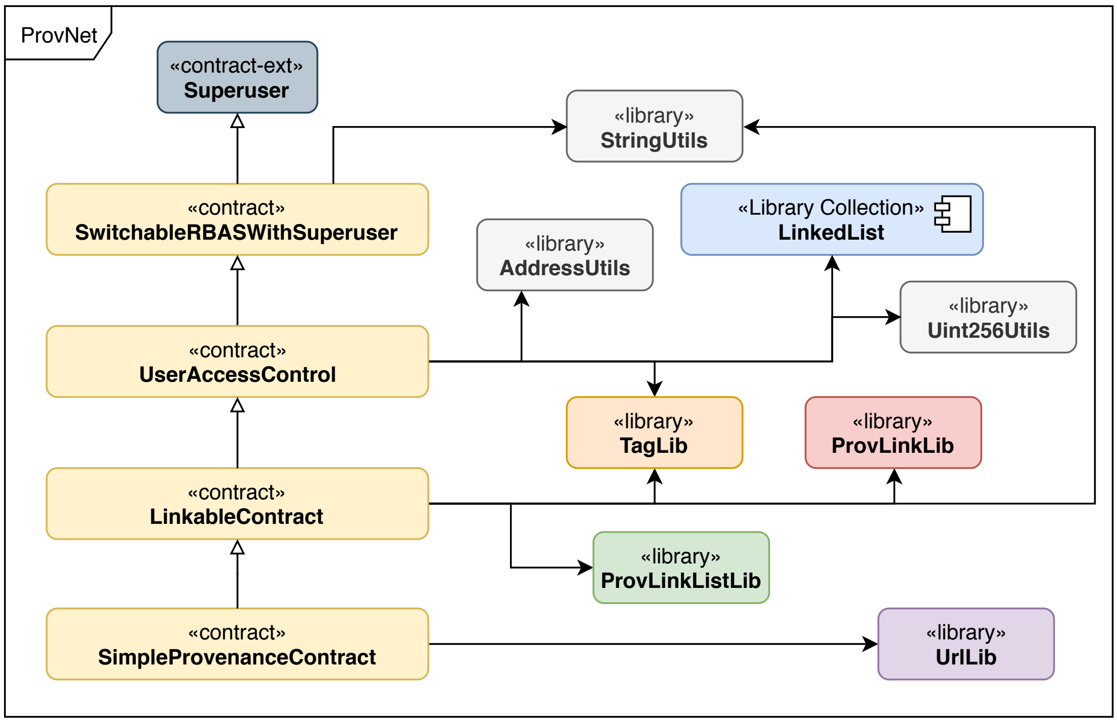

ProvNet
«contract-ext» Superuser
«library» StringUtils
«contract» SwitchableRBASWithSuperuser
«Library Collection» LinkedList
«library» AddressUtils
«library» Uint256Utils
«contract» UserAccessControl
«library» TagLib
«library» ProvLinkLib
«contract» LinkableContract
«library» ProvLinkListLib
«contract» SimpleProvenanceContract
«library» UrlLib

Figure 4.2: A general overview of the core architecture.

into the Truffle build files. Since we do not want to redeploy our auxiliary contracts every time we have to redeploy a contract, we keep them in the repository to persist the addresses of those auxiliary contracts. This general process of migration works fine as long as Truffle is used to perform this dynamic linking. However, in case of our front end client, we want to be able to deploy contracts from static byte code. This means that we need byte code which is already fully linked. To be able to do this, we extended the Truffle build cycle by a custom task. Our task generates for each known network static byte code and writes it to a custom build output file. This file contains the API and statically-linked byte code for each network Truffle knows about. This means that if users want to be able to deploy from within our clients to custom networks, like local test networks or private networks, they will have first to build the sources to such a statically-linked byte code. Otherwise, they will be only able to deploy by using Truffle.

4.1.2 Contract Design Structure

In Figure 4.2, we can see a simplified architectural view of the ProvNet backend. Depicted in yellow is a hierarchy of smart contracts with each contract adding another layer of functionality to the final provenance contract. This architecture has been chosen to allow use cases which need a more complex way of storing provenance data to inherit the rest of the functionality needed for the provenance network to function. Depicted in grey are simple auxiliary libraries and basic structs, and depicted with a different color each are our core libraries which abstract the main functionality of our solution. To keep the architecture overseeable, we have abstracted the dependencies of each core library into a separate diagram. They are only a graphical simplification. Finally, depicted in dark grey are contracts, libraries, and structs which we imported from existing libraries. Before we go into more details, a short note on contracts and libraries in Ethereum.

Contracts can hold complex logic, they can receive funds, they can transfer funds, and they can even call functions on other contracts. Although, in our case, none of the contracts can receive funds. Every user who is prepared to pay the execution costs can call any public function on contracts. In contrast, libraries are not directly callable by users. They must be called by contracts and get executed in the contracts' context. Libraries are a design structure to make code easily reusable and more efficient in the chain and to avoid frequent redeployment of common logic. Libraries work in a way that they extend a certain base type or struct with additional functionality. This way, they can behave as if they were written as part of the initial functionality of that type. This is also the reason why almost every library in our architecture uses at some point a struct type.

Since any user that connects to the Ethereum chain can call public functions on any contract, the contracts' security has to be designed with this in mind. Furthermore, given the previously explained differences between contracts and libraries, we decided to split off contract logic into libraries as good as possible but keep security-relevant details in the contracts. This lead to the main part of our architecture as shown in Figure 4.2.

Next, we will discuss the different contracts that we have:

Superuser: This contract is from a contract library[3] that we use and provides important user management functionality. More specifically, it allows us to manage the contract owner, to define superusers who have similar power to the contract owner, and it allows us to define user roles as required.

SwitchableRBACWithSuperuser: This contract is our extension of the superuser contract that defines some common access security checks that can be used in other functions. Furthermore, we implement here the functionality to allow administrators to deactivate access controls for certain user groups if desired. This can be necessary if an open linking policy is desired, as discussed in Section 3.3.3, or if certain access restrictions should be disabled, as discussed in Section 3.3.4. Although, we can not see any practical use case for the latter.

UserAccessControl: This contract is the heart of our access control, as discussed in Section 3.3.4. Here, administrators can define different user roles, add users with specific roles, and remove roles from users. This contract also defines role-based access control functions that can be used by other contracts.

[3]https://openzeppelin.org/

LinkableContract: This is the base contract that provides the linking functionality for the provenance network, as discussed in Section 3.3.2. No matter how exactly a contract is built to store provenance, as long as it derives from this contract, it will behave exactly as any other contract with respect to linking with the provenance network.

SimpleProvenanceContract: This contract is our simple implementation of a model-agnostic provenance contract, as discussed in Section 3.3.1. It allows users to store successive provenance data for a certain URI. Since the contract does not enforce any model validation users are free to use any desired combination of strategies as discussed in Section 3.1. Furthermore, the functionality for managing the provenance URIs is implemented in this contract. This is because the idea to identify resources by URIs was taken from the W3C PROV model and is therefore not necessarily model-agnostic, as discussed in Section 3.1.6. In future other implementations could decide to use another strategy to identify resources, which is why we put the functionality for identifying resources into the most concrete implementation level. This way, we are not disturbing other provenance solutions or models which want to use the general idea of provenance networks but identify resources differently.

A provenance network has no specific contract since it is the result of multiple provenance contracts, in our case *SimpleProvenanceContract* instances, being linked together, as presented in Section 3.3. Furthermore, interesting to note is that linking policies and access restrictions, as discussed in Section 3.3.3 and Section 3.3.4, are set to restrictive by default and have to be relaxed by administrators as required.

Next, we will give a general overview of the libraries we created:

LinkedList-Library Collection: The linked list library collection, as shown in Figure 4.3, is a collection of auxiliary libraries which extend the functionality of the *LinkedListLib*-Library[4] which we imported. Since we have to store a lot of data into lists, it is of utter importance that we can access these lists in an efficient manner. In our case, this means that all altering operations like, writes, deletes, and edits must be efficient without the need to iterate over the whole list. Read operations are not tragical and are allowed to require iterating over the list. This is due to the fact that we can use read operations for free on the Ethereum chain, however, have to pay for operations that manipulate the state [27]. This means in order to save cost it is important to have no loops of growing complexity in our list-manipulating functions. We found a library that provides the basic structure of such a list, the *LinkedListLib*-Library. All the other libraries we added in this collection are for code reuse or to make certain operations, mainly read functionality, easier to use.

[4]https://github.com/Modular-Network/ethereum-libraries

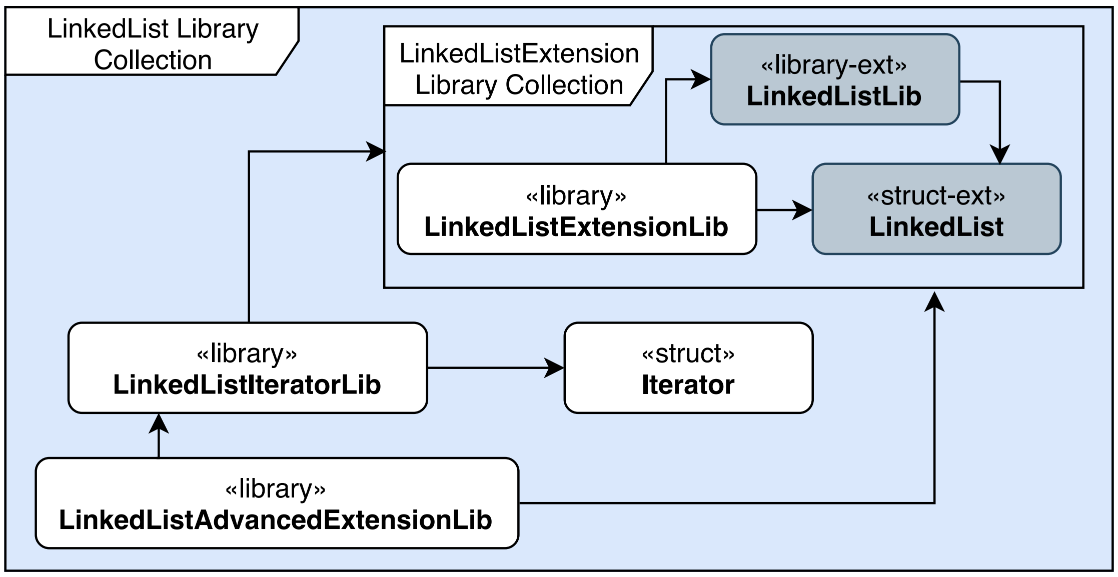

LinkedList Library Collection
LinkedListExtension Library Collection
«library-ext» LinkedListLib
«library» LinkedListExtensionLib
«struct-ext» LinkedList
«library» LinkedListIteratorLib
«struct» Iterator
«library» LinkedListAdvancedExtensionLib

Figure 4.3: An overview of the linked list library collection.

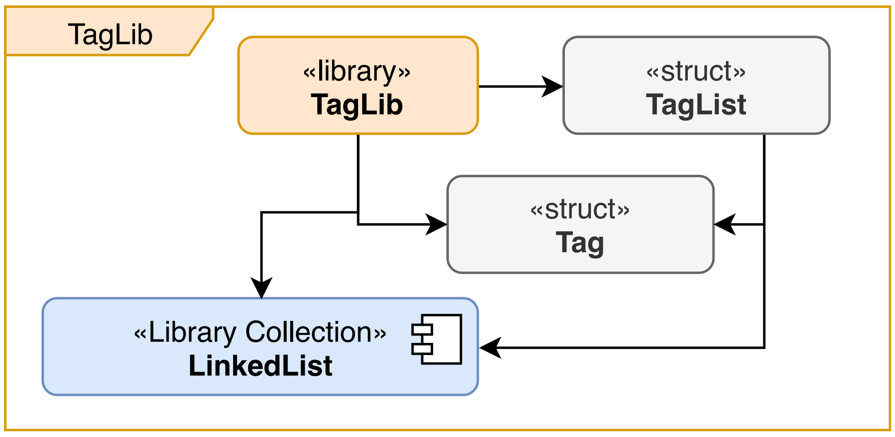


Figure 4.4: An overview of the tag library architecture.

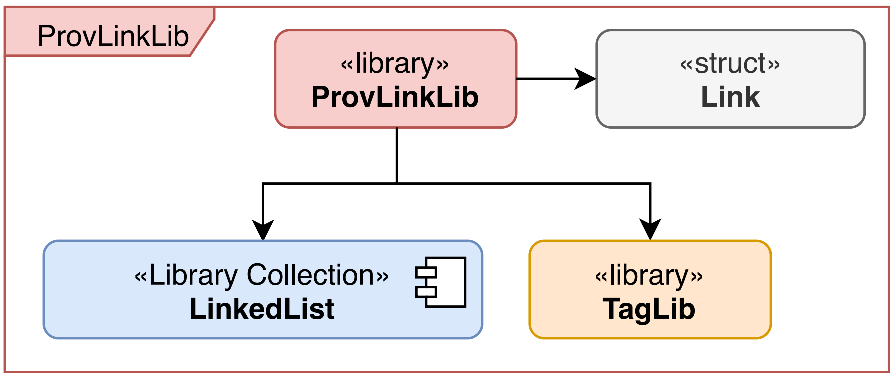


Figure 4.5: An overview of the provenance link library architecture.

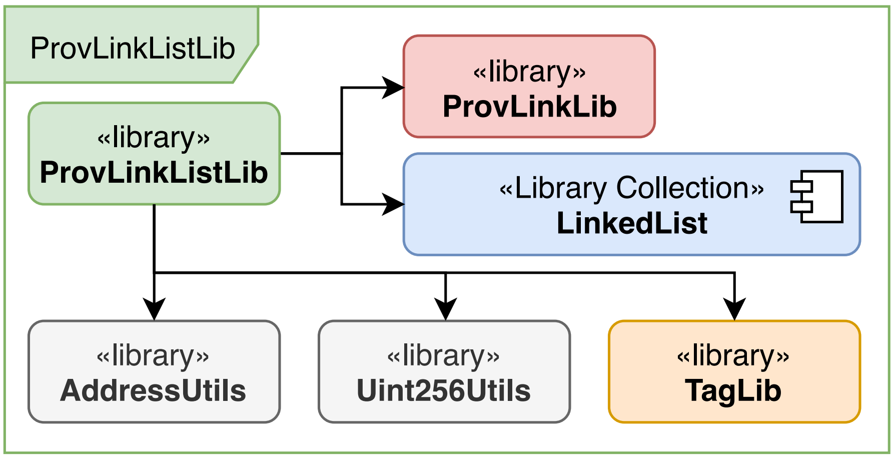


Figure 4.6: An overview of the provenance link list library.

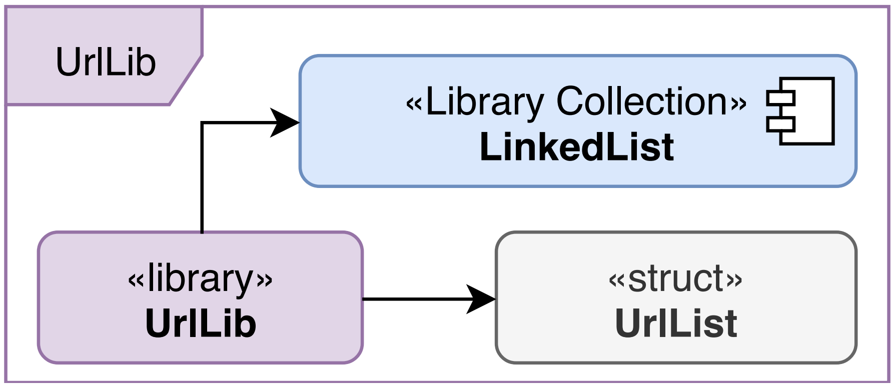


Figure 4.7: An overview of the url library architecture.

TagLib: The most important library after the linked list library collection is the tag library, as shown in Figure 4.4. This library allows us to create tags and tag lists. Those can be used by contracts to tag arbitrary other struct types. As we will see below, this is the core idea behind our linking implementation.

ProvLinkLib: This library introduces the struct required to represent links in the provenance network, as shown in Figure 4.5.

ProvLinkListLib: This library maps the general linked list library to a specific one capable of listing *ProvLink*-structures, as shown in Figure 4.6.

UrlLib: Similar to the *ProvLinkListLib*-library, this library maps the general linked list library to the more specific type of *URIs*, as shown in Figure 4.7. URIs are internally represented by simple string base types.

Auxiliary Libs: The rest of the libraries are auxiliary libraries which add helper functions to basic types.

Now that we have a basic overview of what the different components of our architecture do, we will shortly discuss the most important idea behind how linking and access control are implemented. Both of these functionalities are accomplished by using tagging. More specifically, we do not define what kind of links exist, but we define that every link can have a list of tags. These tags then represent the weights of this link, as discussed in Section 3.3.3. This way, one link can have multiple weights, representing different metrics of trust propagation. This helps us to stay easily extendable by allowing different solutions to employ custom trust metrics without having the need to design custom networks. More generically, with this functionality we allow our users to create use case-specific network overlays. For example, if the lowest level of a network is a tag suggesting *no trust* but interlinking the whole network, then users can use additional links to create overlay networks that contain only the parts interesting for their use cases. Besides the ability to use different trust metrics for different use cases, this functionality gives the users also a very powerful filtering tool without restricting or breaking network connectivity. As we will see in Chapter 5, this filtering tool is a huge help for solving the searchability issue in a model agnostic way, as discussed in Section 3.1. To make quick and simple interlinking of contracts possible, we introduce four different default tags. These four default tags are directly derived from the weights used in the examples in Section 3.3. All of them can be observed in the world view example in Section 3.3.5.

Trusted: A *trusted* link from contract A to contract B means that contract A trusts contact B. This link is for expressing organizational borders and other kinds of strong contract cooperation.

Known: A *known* link is for linking contracts that have weak dependencies on each other. They are in different organizational domains however store provenance on the same resource or build upon each other to express a completer provenance picture about some resource.

Pingback: The *pingback* is beside URIs the second thing that is directly inspired by the W3C PROV model. The idea is to give contracts the tools to inform other contracts that they have provenance about a certain resource that may interest the other contract. It is a way of informing other contracts and their users about the own contract and its provenance which could be of interest for them. We integrated this approach into our solution since it extends the abilities to interconnect the network which increases the overall network strength, as we will see in Chapter 5.

Linkback: Inspired by the *pingback* strategy for provenance data, we introduce the *linkback* link. This link is automatically applied on the other side of the link. This way, contracts that get linked, get informed about being linked and our network gets stronger by making the default link type a two-sided connection instead of a one-sided. This link type does not infer any trust at all. It merely expresses the existence of a contract.

A similar strategy was used for user access control, as discussed in Section 3.3.4. Instead of creating a complex rights management system that would never be able to cover all the use cases, we decided to create the simplest possible and make it extendable. The simplest operations that exist are adding links, adding provenance data, adding users, and managing users. To solve access control easily and extendable, we have to manage access to these operations. We use tagging to solve also this issue. More specifically, we split the different operations into linking operations and administrative operations. Thus, we have user roles regarding linking and user roles regarding administrative operations. Linking roles are basically every link type which gets created by users, meaning that if an administrator creates a tag *trusted* for linking contracts and gives a user the linking role *trusted*, then this user is able to link arbitrary contracts with a *trusted* tag to the current contract. The administrative branch of roles is more difficult to extend, however, it also works based on tags. The two most powerful tags are the *owner* and the *superuser* tags which are defined by the *UserAccessControl*-contract. Those roles have administrative rights by default. Another role that exists is the role of the *editor*, allowing users to push provenance data to the current contract. To extend administrative roles, inheritance has to be used and a new contract has to be created, for the time being.

4.2 React-Client

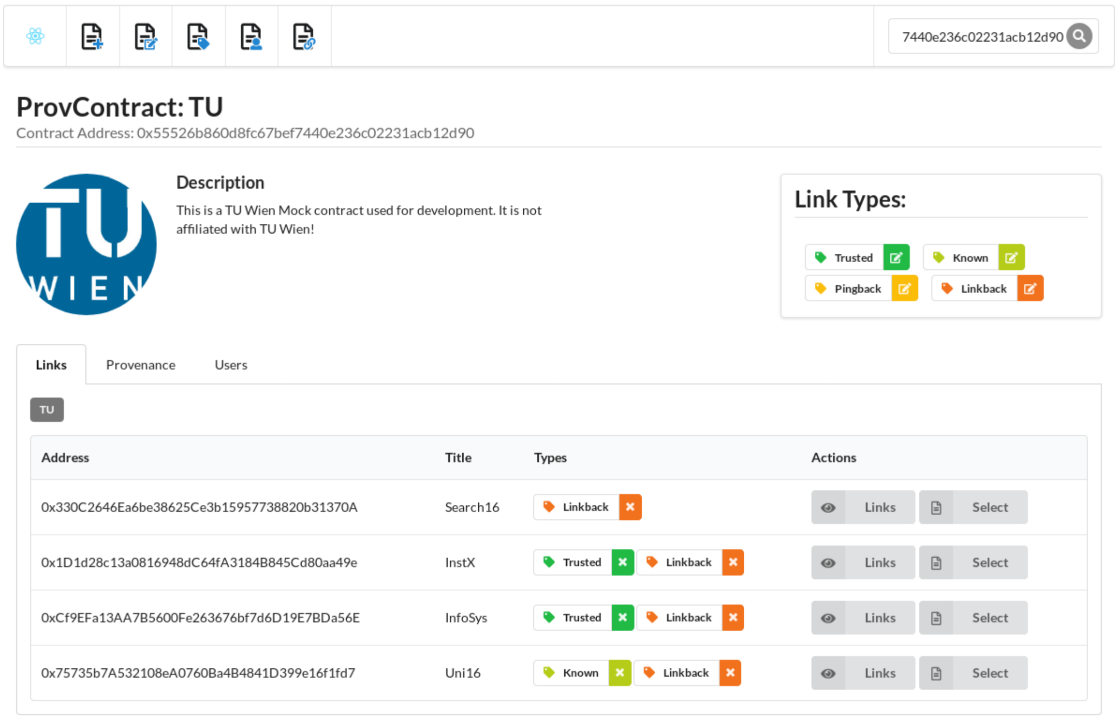
The *React-Client* project is our graphical user interface client. This client is built as a single-page application, as discussed in Section 3.1.2. Thus it is especially suited for our use case since we only need a static server for content delivery and can afterwards connect directly to the Ethereum chain. This connection is established from the user's browser without the need for further communication with any backend server. The client is built with a standard ReactJS[5] stack. ReactJS is hereby the frontend framework. Additionally, we are using the Redux[6] framework for state management and we are using the redux-observable[7] framework to manage side effects. To connect to the Ethereum chain, we are using the Metamask[8] browser plugin. Metamask is a browser plugin that allows for key management and transaction signing directly from the browser. Furthermore, it provides the necessary functionality to connect to the blockchain for apps running in the browser. This way, users do not necessarily need a local Ethereum node to connect to the Ethereum chain.

Our Ethereum client was built with browsing and administrating provenance networks in mind. Thus, it does not support writing provenance data to the network. At the time being, it also does not support searching the provenance network other than by exact provenance contract address. For these two operations, users will have to use the *Node-Client* which we will discuss later in this chapter.

The React-Client follows, to the best of our knowledge, common architectural guidelines and development standards. In general, we implement the principle of splitting view and connected components, meaning that a view component only knows of how to display certain content while a connected component is responsible for populating that view component with the required content. However, we do not want to go into too many details about the standard ReactJS development patterns in this thesis. Instead, we will focus on some design challenges that we had in context with provenance networks and when working with a blockchain.

In general, our client, as shown in Figure 4.8, consists of three main parts. The action bar in the top of the page, as shown in Figure 4.9, the selected contract details in the middle of the page, as shown in Figure 4.10, and the complex content section at the bottom, as shown in Figure 4.11. The action bar consists of a set of standard operations that can be performed on a loaded contract, an operation for deploying new contracts, and a search bar. Given a contract address of an existing provenance contract, the search bar will load this contract, and the newly loaded contract will be selected for display. Once a contract is loaded, you can use the four contract operations in the action bar to change the details of the loaded contract, to add new tags to the contract, to add new users to the contract, or to add new links to the contract. The leftmost operation is a special one, it allows to deploy new contracts to the currently selected chain. For this to work, however, the currently deployed version of the *React-Client* has to have the binary file for this specific network. Otherwise, an error message will be displayed that the selected network cannot be served. Currently, the Github-deployed version has only binary files for the Ropsten[9] test network. The middle section, the selected contract details section, is responsible for showing the details of the currently selected contract.

[5]https://reactjs.org/
[6]https://redux.js.org/
[7]https://redux-observable.js.org/
[8]https://metamask.io/
[9]https://github.com/ethereum/ropsten



7440e236c02231acb12d90

ProvContract: TU

Contract Address: 0x55526b860d8fc67bef7440e236c02231acb12d90

Description

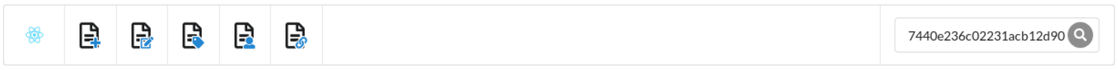
This is a TU Wien Mock contract used for development. It is not affiliated with TU Wien!

Link Types:

Trusted · Known · Pingback · Linkback

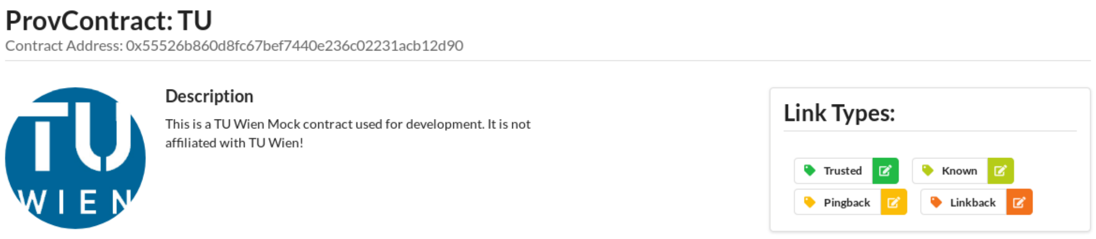
Address	Title	Types	Actions
0x330C2646Ea6be38625Ce3b15957738820b31370A	Search16	Linkback	Links · Select
0x1D1d28c13a0816948dC64fA3184B845Cd80aa49e	InstX	Trusted · Linkback	Links · Select
0xCf9EFa13AA7B5600Fe263676bf7d6D19E7BDa56E	InfoSys	Trusted · Linkback	Links · Select
0x75735b7A532108eA0760Ba4B4841D399e16f1fd7	Uni16	Known · Linkback	Links · Select

Figure 4.8: The React-Client view.



7440e236c02231acb12d90

Figure 4.9: The action bar of the React-Client.



ProvContract: TU

Contract Address: 0x55526b860d8fc67bef7440e236c02231acb12d90

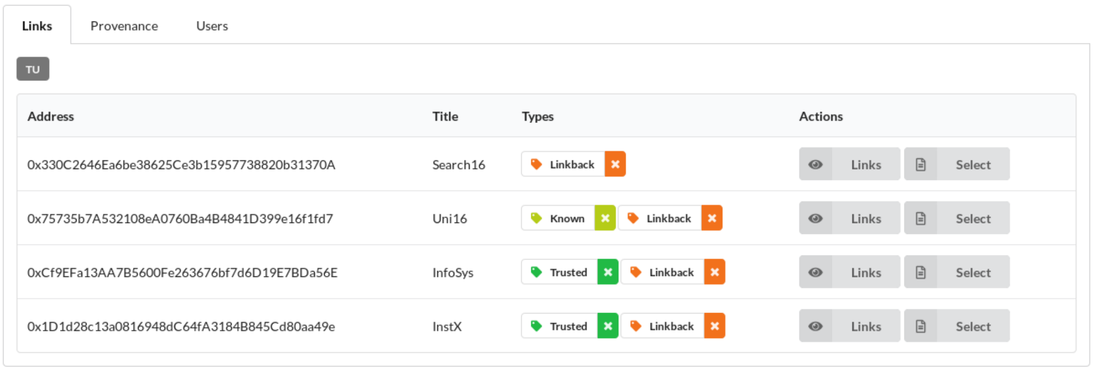
Description

This is a TU Wien Mock contract used for development. It is not affiliated with TU Wien!

Link Types:

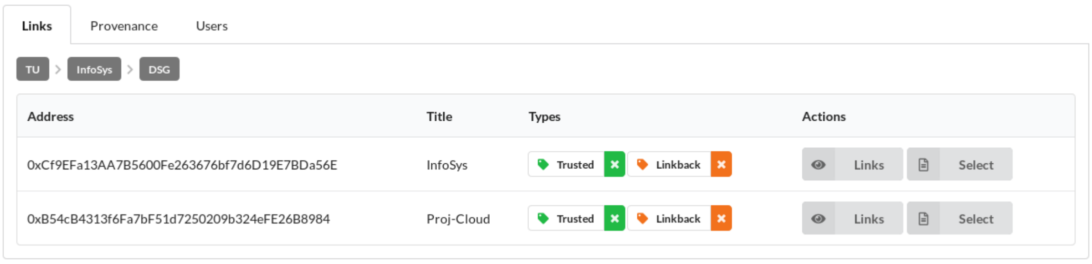
Trusted · Known · Pingback · Linkback

Figure 4.10: The details part of the React-Client.



Address	Title	Types	Actions
0x330C2646Ea6be38625Ce3b15957738820b31370A	Search16	Linkback	Links Select
0x75735b7A532108eA0760Ba4B4841D399e16f1fd7	Uni16	Known Linkback	Links Select
0xCf9EFa13AA7B5600Fe263676bf7d6D19E7BDa56E	InfoSys	Trusted Linkback	Links Select
0x1D1d28c13a0816948dC64fA3184B845Cd80aa49e	InstX	Trusted Linkback	Links Select

Figure 4.11: The complex content part of the React-Client.



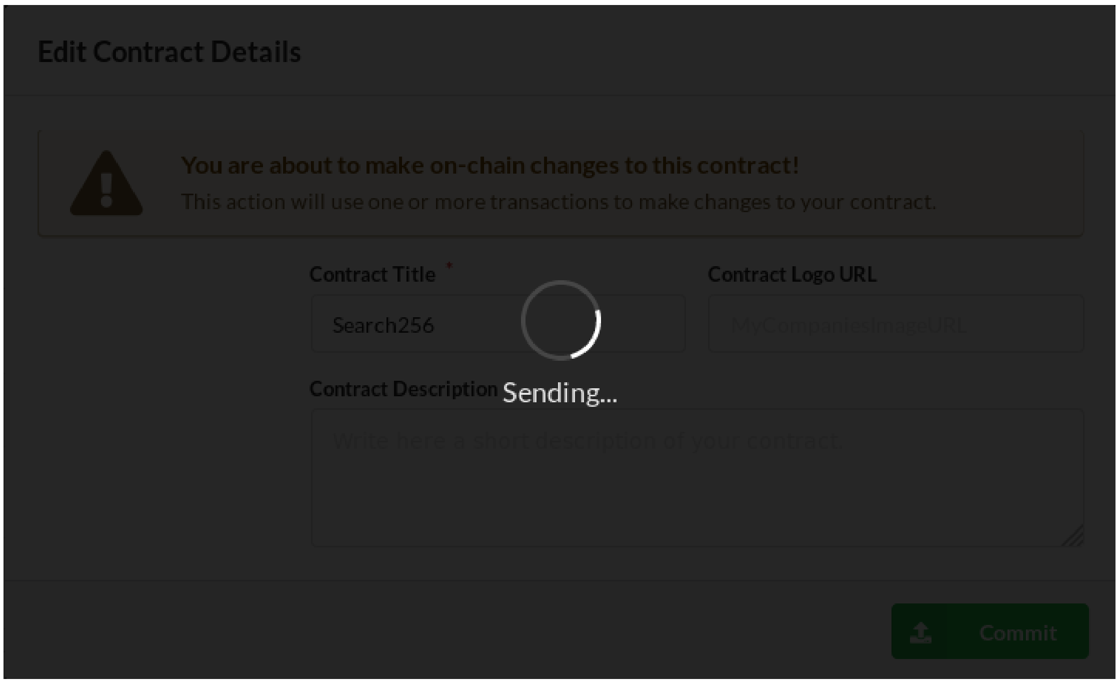
Address	Title	Types	Actions
0xCf9EFa13AA7B5600Fe263676bf7d6D19E7BDa56E	InfoSys	Trusted Linkback	Links Select
0xB54cB4313f6Fa7bF51d7250209b324eFE26B8984	Proj-Cloud	Trusted Linkback	Links Select

Figure 4.12: The link navigation view.

This includes the title of the contract, the address of the contract, the description of the contract, and if loadable also the picture of the contract. Furthermore, on the right side is a box showing the currently existing link tags in this contract. The bottom section, the complex content section, is for displaying the links, provenance, and users of the currently selected contract.

There where two particular challenges when we were designing the user interface for our solution. The first one was how to represent a directed cyclic graph easily and understandable in the user interface, and the second was how to properly handle the asynchronicity in a simple and understandable way.

The first issue was about how do we represent a complex graph without trying to draw the graph. The problem hereby is that we did not want to draw the graph to avoid complexity when working with big provenance networks. We however still wanted to allow users to easily browse the network by simply being able to follow links from contract to contract. The initial idea then was to use a tree view similar to file explorers. However, such a tree view has the disadvantage that it could get very deep and since we have a graph which allows for cycles we also would need a strategy to handle those. The main reason not to go with a tree view, however, was the depth issue. A very deep tree view would get difficult to display properly in the user interface since at some point it would



Edit Contract Details

You are about to make on-chain changes to this contract!
This action will use one or more transactions to make changes to your contract.

Contract Title *
Search256

Contract Logo URL
MyCompaniesImageURL

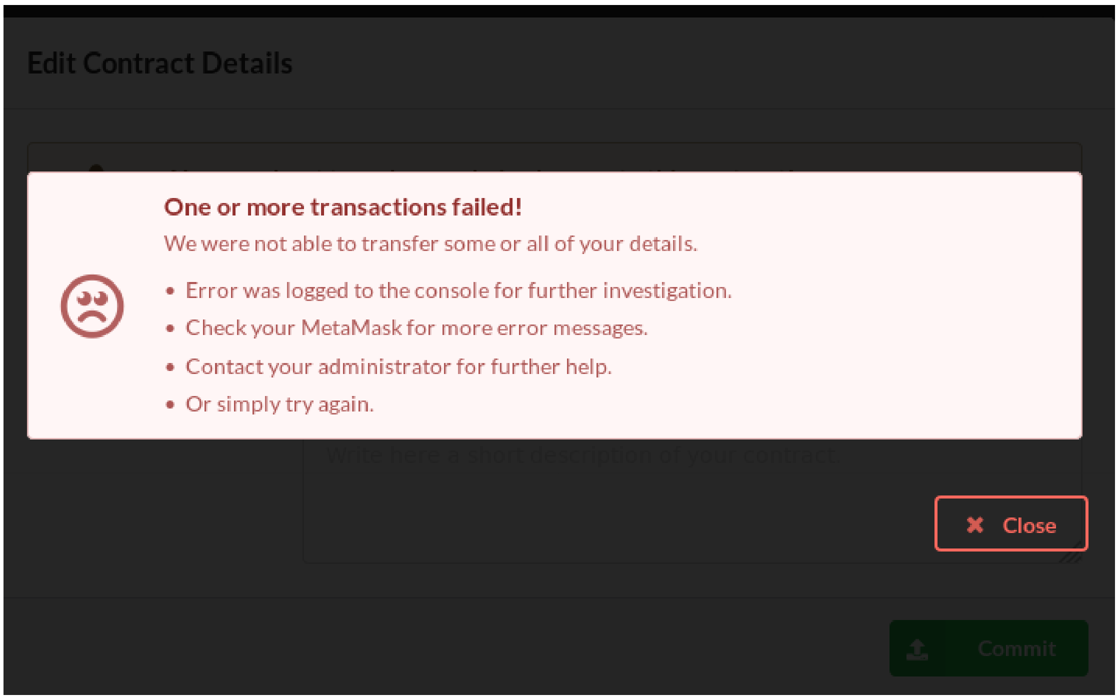
Contract Description
Write here a short description of your contract.

Sending...

Figure 4.13: View of some dialog while transaction is being processed.

simply reach the end of the monitor and could not display the link properties without introducing horizontal scrolling, which is naturally more difficult for users than vertical scrolling. As such, we decided to use simple tables to display links. And for the purpose of orientation, we introduced a navigational overview to display the depth of browsing from the initially loaded contract forward. As you can see in Figure 4.11, the links table initially displays the links of the currently selected contract. However, if you use the *links*-button on any one of the links, the table will display the links of that contract, as shown in Figure 4.12. Meanwhile, the navigational overview will show us which path we took from the currently selected contract to be able to see the links that are currently displayed. The navigational overview can also be used to jump back to a certain earlier contract. Once we find what we were looking for, we can use the *select*-button to load a specific contract and make it the newly selected contract, meaning that we will be able to see the details of that contract in the middle section of our view.

The second issue was about how to handle the asynchronous requests to the blockchain in an easily displayable way. Even more so that in case of changes users have to sign the transactions in the Metamask extension before it can be submitted to the blockchain and handled there. And then, the transaction still has to be added to a block and accepted into the blockchain before we can see and query the data. Since this can, in general, take a while, we wanted to have a way to visualize for users that a transaction is running, or in what state a transaction finished even if the users leave the contract and come later back to it. Furthermore, we wanted to be able to use the same pattern with all


Edit Contract Details

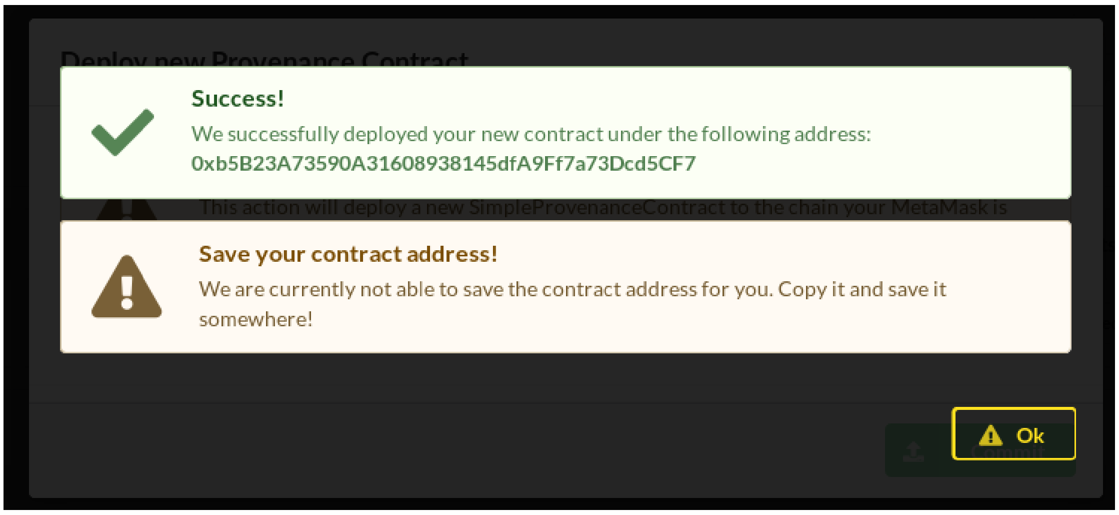
One or more transactions failed!

We were not able to transfer some or all of your details.

- Error was logged to the console for further investigation.
- Check your MetaMask for more error messages.
- Contact your administrator for further help.
- Or simply try again.

Write here a short description of your contract.

Figure 4.14: View of a failed transaction.


Deploy new Provenance Contract

Success!

We successfully deployed your new contract under the following address:
0xb5B23A73590A31608938145dfA9Ff7a73Dcd5CF7

This action will deploy a new SimpleProvenanceContract to the chain your MetaMask is

Save your contract address!

We are currently not able to save the contract address for you. Copy it and save it somewhere!

Figure 4.15: Message after successful contract deployment.

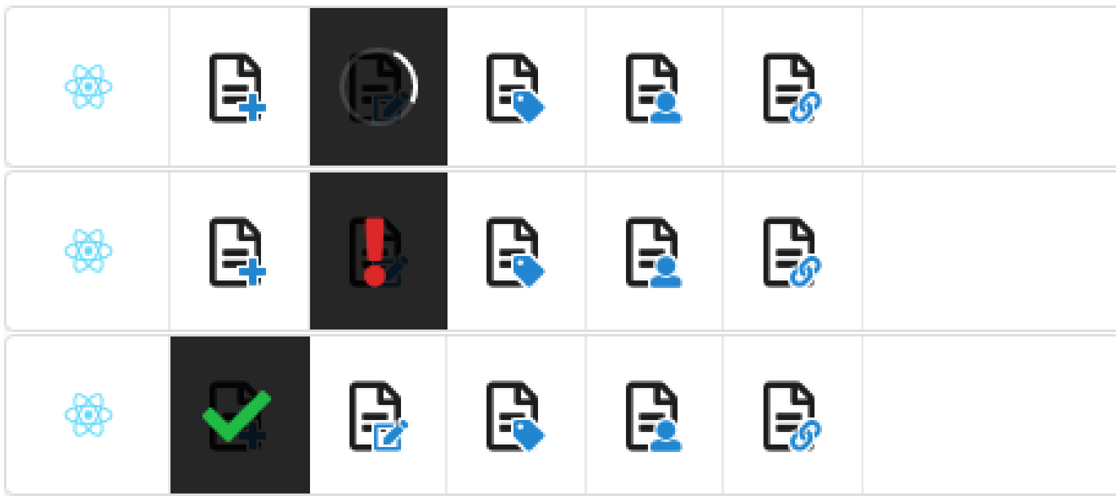


Figure 4.16: Main view transaction state information.

the different elements that we display in the user interface to avoid confusion. So, in order to not block a user after submitting a transaction to the blockchain, we came up with an element-based notification and visualization pattern. Once a user submits a transaction, for example, to change the description of a contract, the description dialog for this contract gets looked in a *in transaction* state and displays a load screen, as shown in Figure 4.13. Once the transaction gets executed successfully or fails with an error, the user interface will adapt to show this state until the users actively inspect what has happened, as shown in Figure 4.14. Only after that, the user will be able to use that same dialog again. We are using also the same behavior in the main view to keep users informed, as exemplarily shown in Figure 4.16. This way, we can force users to get important messages for certain transactions while allowing for it to happen when it is convenient for the users. A good example of this is the contract deployment. After successful deployment, the user has to write down the contract address since we have nowhere to save it, as shown in Figure 4.15. It also does not make sense to allow a user to edit the same field again while the last edit is still being processed. Since both of the user's transactions will eventually get processed, however, due to the distributed nature of the blockchain we are not able to influence the order in which they will be processed. Meaning, that we can not guarantee which value will be written into that field in the end. This is of particular importance for fields that contain a certain value, like titles and descriptions. Lists are not influenced by this behavior since multiple transactions creating new entries are treated as creating different entries. However, when editing or deleting single elements, we again have to ensure processing of the previous transaction before allowing the next.

4.3 Node-Client

The *Node-Client* is a simple NodeJS[10]-based application which exposes an API that can be used for programmatically accessing certain functionalities of the provenance network. It is a typical example of a proxy client, as discussed in Section 3.1.2. Besides some of the administrative functions which are also provided by the *React-Client*, the node client also offers functionality for writing provenance data to the blockchain and also for searching the provenance network. The general idea was that users will want to integrate the functionality of recording provenance data into existing, automated solutions instead of doing it manually. With a lightweight simple server component like this, it can be set up in various ways and simply used with HTTP calls. It does not matter if the node service is set up locally on a host or on a server as part of a more complex application landscape where multiple different applications can use it.

A Node-Client will look in the current directory for a *.keys*-folder. In this folder it will look for a configuration of the type *config.<network>.json* where the *network*-part is a placeholder for the Ethereum network to connect to, for example, the main net or Ropsten. However, this is simply a way of identifying the correct configuration file and users may enter any string they want at this point. The network, the URL where a blockchain node can be found, and the port on which the Node-Client will expose its API can be specified by flags when starting the application. If the Node-Client cannot find any configuration file for the chosen network it will generate a new key pair and create the respective configuration file. After the startup has been successful, the node client will give a brief overview of which node URL, which private key, and which address are used for the current instance. This way, administrators can give node clients custom rights if necessary and users do not have to share their keys with automated clients, although, they can if required or wanted. This entirely depends on the internal organizational structure and the use case in question.

Once the node client is set up and the used key has the required *editor* rights, the client can be used to simply push provenance information to the respective provenance contracts. As a response to a *push provenance*-request, the requestor will receive a UUID. This UUID can later be used to check the state of the transaction. This means that clients which use the node client to write provenance data to the chain have to poll the node client for the results. This is necessary since the actual transaction can take some time to be processed by the blockchain network and REST request should be fast and stateless. Actually, most of the API calls which require a transaction work in a similar fashion, although, not all of them provide a UUID for polling the state of the transaction. Some use natural UIDs to achieve the same behavior. One such example is the contract address since it is unique by nature.

The node client provides at the time being a simple search API which also relies on the clients to poll for results. Although it does not require any transactions, the search can take up a lot of time which might lead for some HTTP frameworks to automatically

[10]https://nodejs.org/en/

close the server connection. Again, to avoid this, we return a UUID identifying the respective search request and expect the clients to poll for the results. The search API allows searching for a resource URI with a set of weights which should be considered while traversing the network. The search algorithm will not make any inference about the set of weights but will match them exactly. For example, if a search request specifies the *known*-links to be considered, the search algorithm will consider only *known*-links. If a user wants to consider all weights with a trust of at least *known*, the user will have to specify this in the search request. In our case, the search request would then include *known*, and *trusted* weights. Once completed, the search will return a complex JSON object listing all contracts containing provenance data for the searched resource URI and also a list of all contracts that were searched during this request. Finally, our search algorithm uses a dead-end strategy, meaning it does not traverse the entire network. Instead, it continues to search for as long as it can find links that have the required weights. A contract, which has links, however, where none of the links has a required link weight will be regarded as a dead-end and the contracts linked will not be searched.

4.4 Eval-Project

For the evaluation of our solution, we had to deploy a quite large provenance network from scratch. This means deploying contracts, giving users the correct rights, and linking contracts before we can even start evaluating network behavior. Deploying the entire network takes a lot of time, and consumes a lot of resources. Furthermore, the different steps can fail due to a multitude of reasons, including network timeouts, disconnects, errors at node providers, and framework errors. Given these circumstances, we decided to create an evaluation helper project which allows us to deploy the entire network automated and supports failure recovery during arbitrary steps of the deployment process. Meaning if an error occurs during the deployment process, the evaluation project will be able the determine where the error has occurred and if the last executed transaction passed or not. Afterwards, it will continue at the point where it failed during the last execution. This allows us to recover quickly and cost-efficiently from failures without losing the already achieved progress. Furthermore, we automated most of the actual measurements in the evaluation projects allowing users to simply rerun those operations on the already deployed provenance network for easy result reproduction.

To make this possible, the evaluation project uses the Node-Client and acts as a client to the Node-Client. All the actions that the evaluation project has to perform are ultimately sent as requests to the Node-Client, processed there, and the results are then processed by the evaluation project, including logging and simple statistical evaluations. All in all, the evaluation project offers automated functionality for deployment, cost measurement, time measurement, and scenario-based evaluation.

The failure recovery process works for contract deployment by logging the addresses of successfully executed transactions. In case of a failure, all correctly logged contracts will be skipped. This means that it can happen that single contracts are deployed twice if the error happens before the evaluation client was able to log the successful deployment of some contract. However, single cases of redeployment do not hurt the time and resources required as severely as redeploying all contracts would. This is because only after the successful deployment of all required contracts the evaluation client will start adding user rights and linking contracts. This way, single, accidentally duplicated contracts due to errors while deploying can be simply ignored since they will not be added to the final provenance network. All the following operations, like adding user rights and contract linking, are also logged, however, the state for failure recovery is not inferred from the logs but from the actual state on-chain. This allows knowing exactly which operations have to be skipped and which have to be retried.

4.5 Summary

In this chapter, we introduced the general contract architecture of our deployed contracts. Furthermore, we presented two example clients of different natures with different focus groups, covering in total all the required tools to successfully build provenance networks and record provenance data. Finally, we introduced an evaluation helper project, which automates the deployment of our test network and of measurements required for the evaluation.

The *ETH*-project, as presented in Section 4.1, is mainly based on the design discussion, as discussed in Section 3.3. The two client projects, as presented in Section 4.2 and Section 4.3, are good examples how different use cases can impose a superset of functionality, as discussed in Section 3.1, on top of our general idea of provenance networks. In our case, the clients offer a different set of access functions with the search behavior being the most prominent difference. The same way the clients could use different storage methods, as discussed in Section 3.1.1, or different provenance models on top of our provenance network.

Finally, our architecture allows users to extend our contracts if they require model validation on-chain. They then can create model specific versions of the provenance contract without having to give up the general access control or linking capabilities of our solution.

CHAPTER 5

Evaluation

In this chapter, we will evaluate the main contributions and claims from the design chapter. We extensively discussed the different ways how provenance data can be saved in or linked to the blockchain. We also discussed the different advantages and disadvantages of saving and linking the data. Especially interesting was the issue of cost. Given that the granularity of the provenance data may vary a lot between different use cases, it can lead to very high cost. The broad spectrum of use cases and the implications to the cost lead us to keep the prototype, model- and use case-agnostic allowing for the different users to decide based on their specific needs in which form and granularity to write provenance data to the blockchain. In the first part of this chapter, we will, therefore, evaluate the cost of the most important operations of our prototype and especially show the cost of saving provenance data to the blockchain.

We contributed the provenance network to improve searchability and security of provenance data. Therefore, we will provide in the second part of this chapter a scenario-based evaluation of our suggested model, the provenance network. These scenarios will aim to show how our provenance network improves the searchability of provenance data and which limitations it has. Furthermore, the scenarios will discuss how the provenance network improves security with respect to the duplication issue. In the final part of this section, we will discuss the non-functional properties which we have identified in Section 2.4. We will discuss how our solution behaves in regard to those properties and compare our solution to the solutions presented in the related work.

5.1 Cost of the Blockchain

In the first part of this section, we will dive deeper into the cost of storing provenance data. Therefore, we will measure the cost of storing different amounts of provenance data within a single transaction and compare those to the cost of storing the same amount of data by utilizing multiple transactions with small chunks of data. In the second

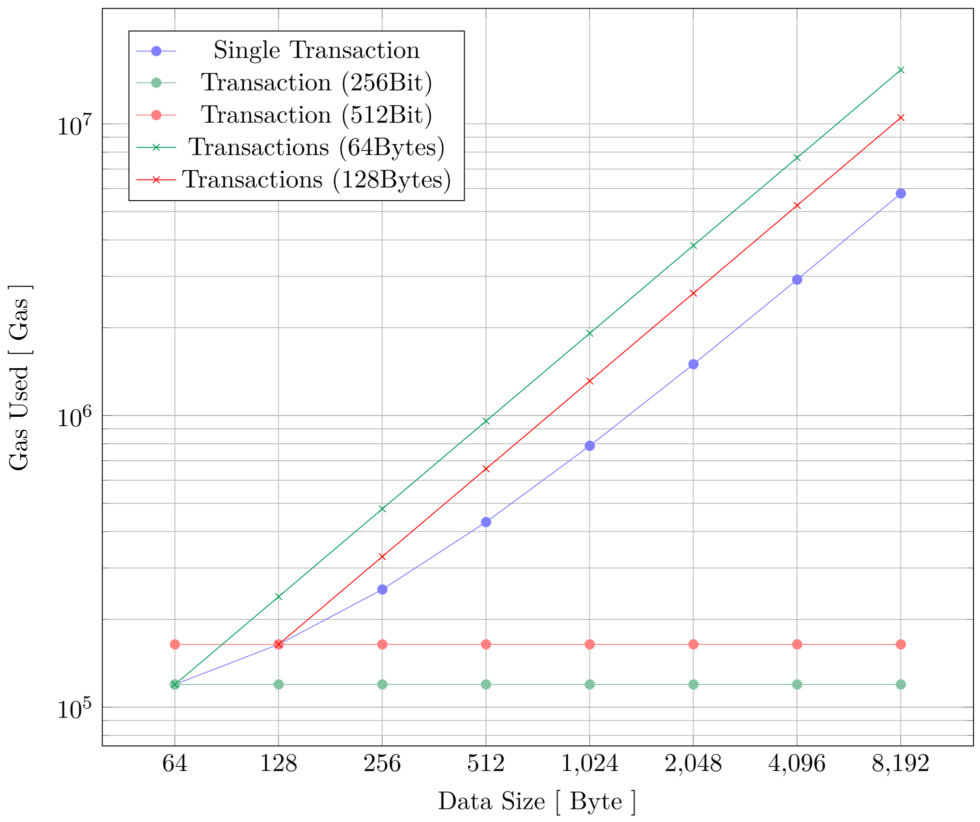


Figure 5.1: A comparison of the cost to store provenance data (logarithmic scale).

part of this section, we will talk about the cost of the most important operations our prototype provides. This will include a brief discussion about the deployment cost of new provenance contracts. We will measure the results in *gas cost* which is a unit stating how much gas was used by the EVM to run a certain operation. Gas cost is a more stable indicator of the cost of operations since the gas price can vary and is market dependent.

5.1.1 Provenance Cost

In our previous chapters, we claimed on multiple occasions that granularity and provenance data size play a major role in the cost of storing provenance. In Figure 5.1 we can see multiple interesting cost comparisons. The data size is calculated in bytes and the cost is depicted in *gas*. Later in this section, we will map some of the relative *gas*-cost to monetary cost. First of all, we can see, depicted with blue dots, the cost of big transactions. These are in our case transactions that hold the actual provenance data in them and are representative for the on-chain model, as presented in Section 3.1.1. The biggest transaction that we are showing holds $8KiB$ of provenance data in it and has a cost of $5.8MGas$. This has a specific reason, as of the time of writing the maximum allowed gas per block in the Ethereum chain is $8MGas$. The next step in our graph,

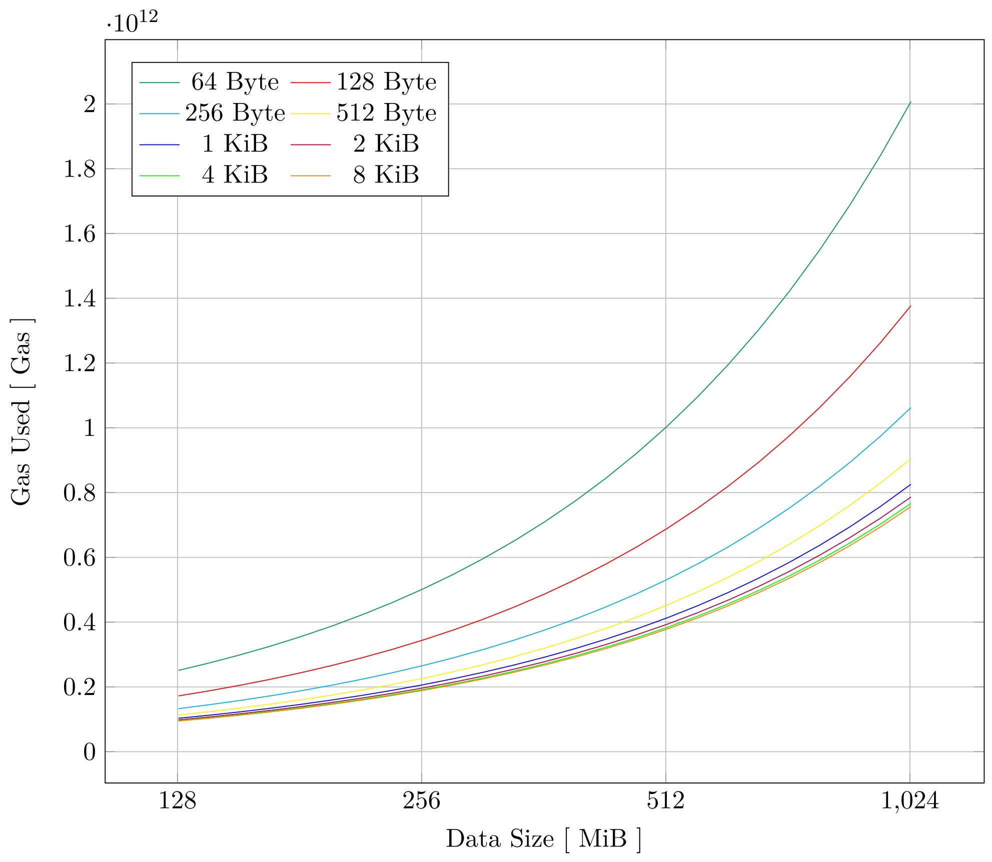


Figure 5.2: Interpolated cost to store provenance data.

$16KiB$, would already require a gas amount higher than the maximum allowed block gas amount and would thus not be accepted by the network. This means for storing more than $8KiB$ of data we would need to split this data into multiple transactions.

Furthermore, we can see depicted with red and green dots, two constant lines. Those are representative for off-chain data. Unlike on-chain data, in the case of off-chained data, we are using hashing algorithms to link the real data to the chain. As we learned in Section 2.2.1.1, hash functions have always an output of the same size no matter how big the input. Since the Ethereum network is designed to use the same gas amount for equal operations [27] and we are always calling the same function on the provenance contract with the same amount of data, the cost stays constant. The two algorithm categories we decided to present the cost for are 256- and 512-bit algorithms. The number of bits stands for the output size. A 256-bit hash has 32 bytes. To represent such a hash as a hex string on-chain we need 64 bytes. The 512-bit hashes behave identically with a required space of 128 bytes, on-chain.

For comparison, we can see, depicted with red and green crosses, two measurements of how the cost evolves if we use a lot of tiny transactions to store the same amount of data.

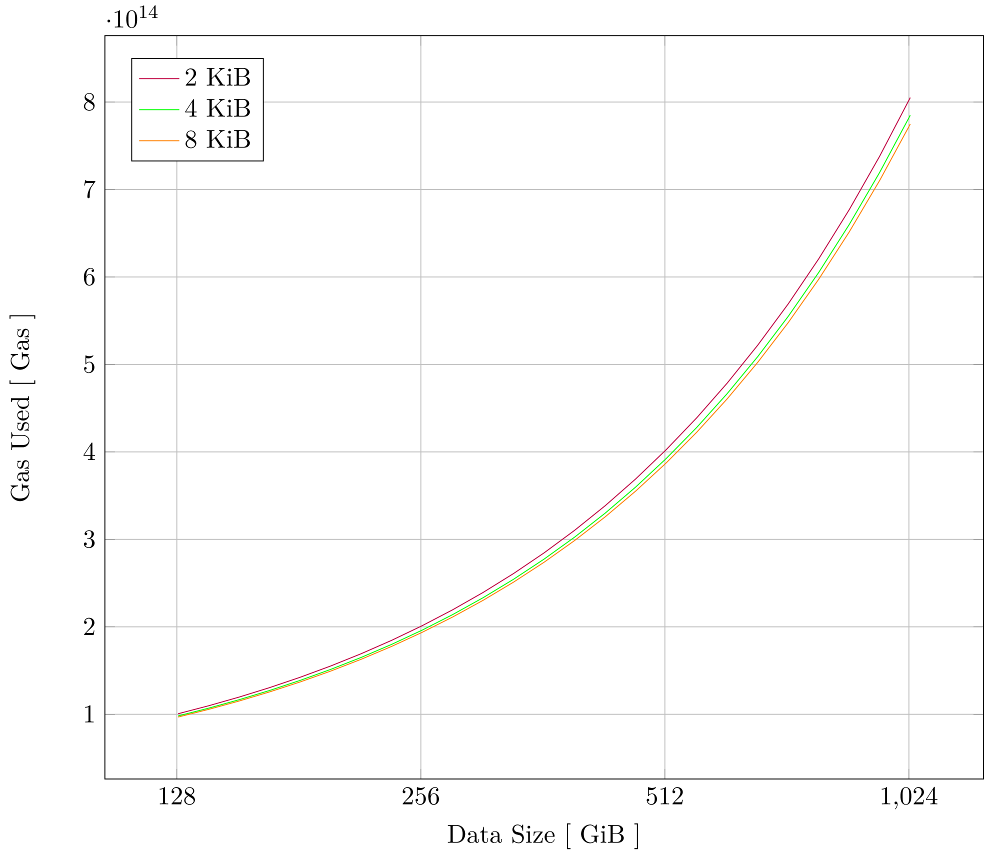


Figure 5.3: Zoomed in, interpolated cost to store provenance data.

As can be seen, the cost grows linearly with the amount of data, however, a lot faster compared to a single transaction with the same amount of data. For instance, if we take again the $8KiB$ point, in the case of the $128B$ curve we have at this point already a total cost of $10.5MGas$. The linear growth has the same explanation as in case of the constant functions. Since we are calling the same function with the same amount of data, the single calls will always produce the same cost. However, since we are actually storing the data on-chain, in comparison to only hashing it, we get a linear cost growth with the amount of stored data. This also allows us to interpolate any measured transaction cost with regard to the cost of multiple transactions of the same kind and size, as shown in Figure 5.2. These interpolations show us how the storage cost would grow for repeated transactions of the different measured sizes.

Another interesting detail that can be seen in Figure 5.2, is that transactions with sizes of $2KiB$ to $8KiB$ have no significant difference in their cost development. Even if we scale them up to $1TiB$ as can be seen in Figure 5.3. This is interesting for system designs that tend to save more data into the blockchain since smaller transactions have better chances to be accepted quicker than bigger transactions. This is a direct consequence of the total

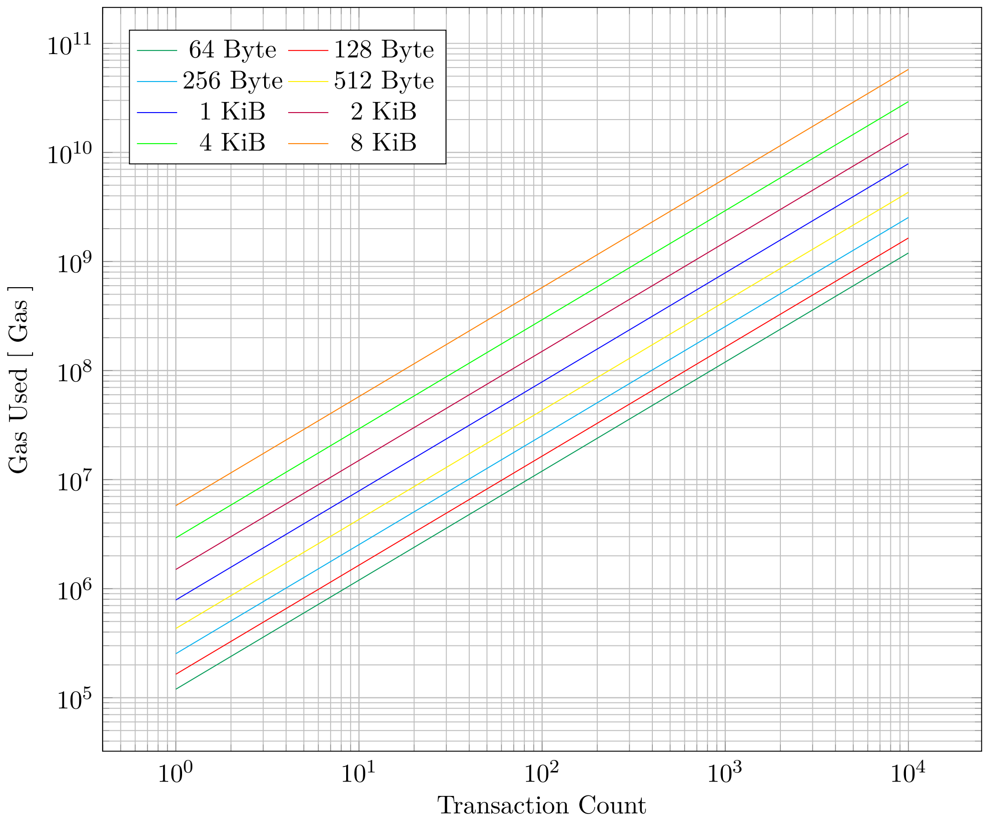


Figure 5.4: A comparison of the interpolated cost per transaction count (logarithmic scale).

block gas limit and can be easily explained. A miner who is currently building a block adds new transactions to the block as they keep arriving, as discussed in Section 2.2.3.3. With each added transaction, the left totally allowed gas size for the current block gets smaller, thus it is easier to fit smaller transactions than bigger transactions. If we cross reference this with Figure 5.1, we can say that the probably best-suited transaction size for storing bigger amounts of data is between $2KiB$ and $4KiB$.

Finally, we also take a look at the cost development depending on the amount of transactions. As can be seen in Figure 5.4, when it comes to the total number of used transactions, smaller transactions are significantly cheaper than bigger transactions, as expected. Use cases that have a high throughput and need to write very often to the blockchain should, therefore, aim for solutions that keep the transaction size as small as possible. If we compare the most-suited three transaction size which we measured, as shown in Figure 5.5, we can see that there exists a difference of around factor two between the $64Byte$ and the $256Byte$ sizes.

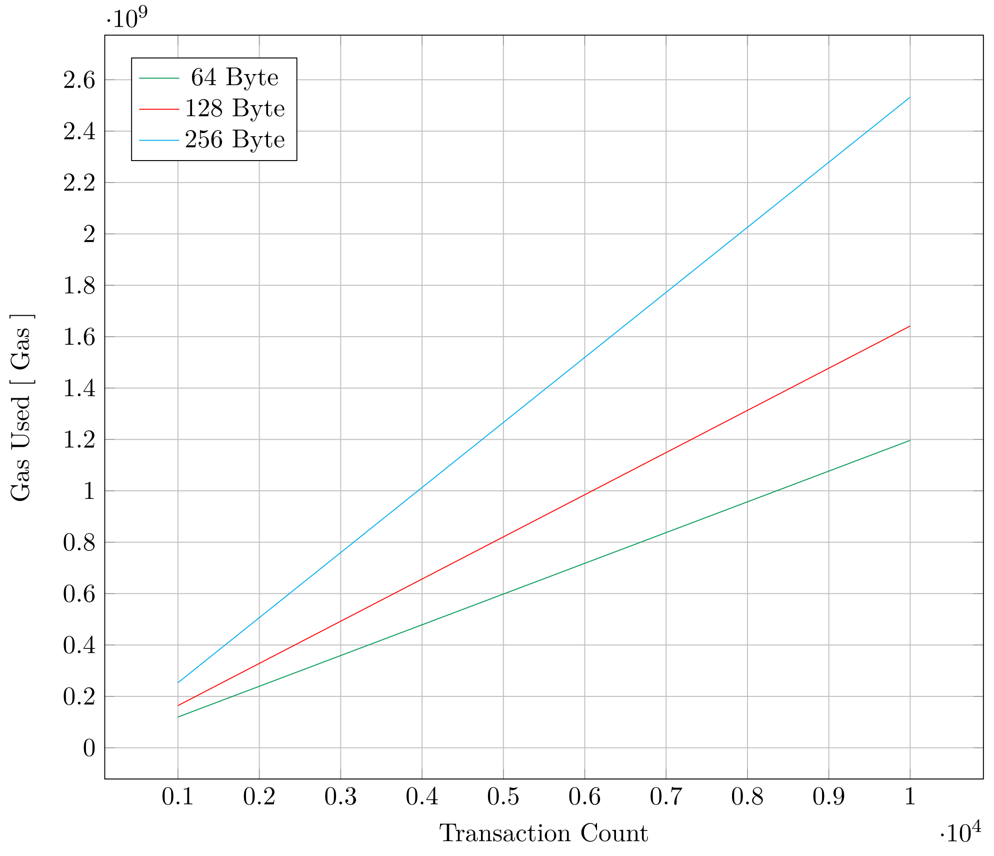


Figure 5.5: Zoomed in, interpolated cost per transaction count.

5.1.2 Monetary Cost

In our cost evaluation up to now, we measured the cost in used *gas*. This is due to the fluctuating gas price. As with the gas price for cars, the gas price for Ethereum fluctuates. At the time of writing, February 2019, the suggested gas price was between $2GWei$ and $3GWei$. The Ethereum price was around € 92 for $1ETH$ as listed by the Kraken exchange and $1ETH$ is $10^{18}Wei$ [27], which are $10^9 GWei$. We can use these numbers to have a look at some interesting values. For example, the cost for one 64 Byte transaction would be € 0.033. On the other hand, to store a total of $1GiB$ of provenance data on-chain split in transactions of $4KiB$ would cost € 211,519.962. And, storing $10k$ of transactions each $64KiB$ in size would cost € 330.206. As can be seen, there is a huge cost gap between storing provenance data on-chain versus storing the data off-chain and just storing the hash on-chain. This is one of the main reasons why it is of utter importance to use a common way of handling data provenance on the blockchain side. This allows us to work towards interoperability without losing the flexibility of customizing solutions towards specific use cases.

5.1.3 Operational Cost

Since besides storing provenance data, creating a well-connected provenance network is the main goal of our solution, we will now have a short look on the cost of the most important operations beside the operation for storing provenance data. To create a provenance network, two operations are of particular importance, deploying a provenance contract, and linking provenance contracts. Furthermore, to be able to achieve organizational requirements configuring proper access control is a very important operation. For those three operations, we measured the required cost and came up with the following results.

Deploying a Contract: Deploying a contract cost us $5.3MGas$. This cost are rather high considering the block size limit and mean that we should consider for our future work a redesign of our internal contract structure to allow for cheaper deployment cost of provenance contracts.

Linking Contracts: We measured over 500 linking operations with an average cost of $166kGas$, with a low of $58kGas$ and a high of $231kGas$. The exact amount of required gas depends on how many operations have to be performed in the blockchain and also if for the specific link a *linkback* already exists or not. This means that linking a contract cost in fiat on average € 0.046 at the time of writing.

Adding Users: We added one user with three roles to 256 contracts and measured the following cost. For the initial creation of the user at a contract, we required $96kGas$. For adding additional roles to that user once added to the contract we required around $52kGas$. It slightly changes depending on the length of the role name. This is in fiat € 0.026 for creation and respectively € 0.014 for adding roles at the time of writing.

5.2 Searching Provenance Data

In this section of the evaluation, we will use a scenario-based approach showing different scenarios that were taken into account during the design of our solution. Based on these scenarios, we will present that we are able to find all the relevant provenance for a given resource in our network and that it is not possible to hide provenance in the network. More concretely, we will show that a malicious user is not able to infer trust from a trusted contract while hiding some provenance at the same time. After the scenario-based part of this section, we will also show and compare different search times based on different network sizes.

5.2.1 The Evaluation Network

For this evaluation, we built a provenance network with a total of 256 contracts, as shown in Figure 5.6. Each circle represents one contract and each background color represents one domain, for example, all contracts which are part of the *TU Wien* domain have a blue

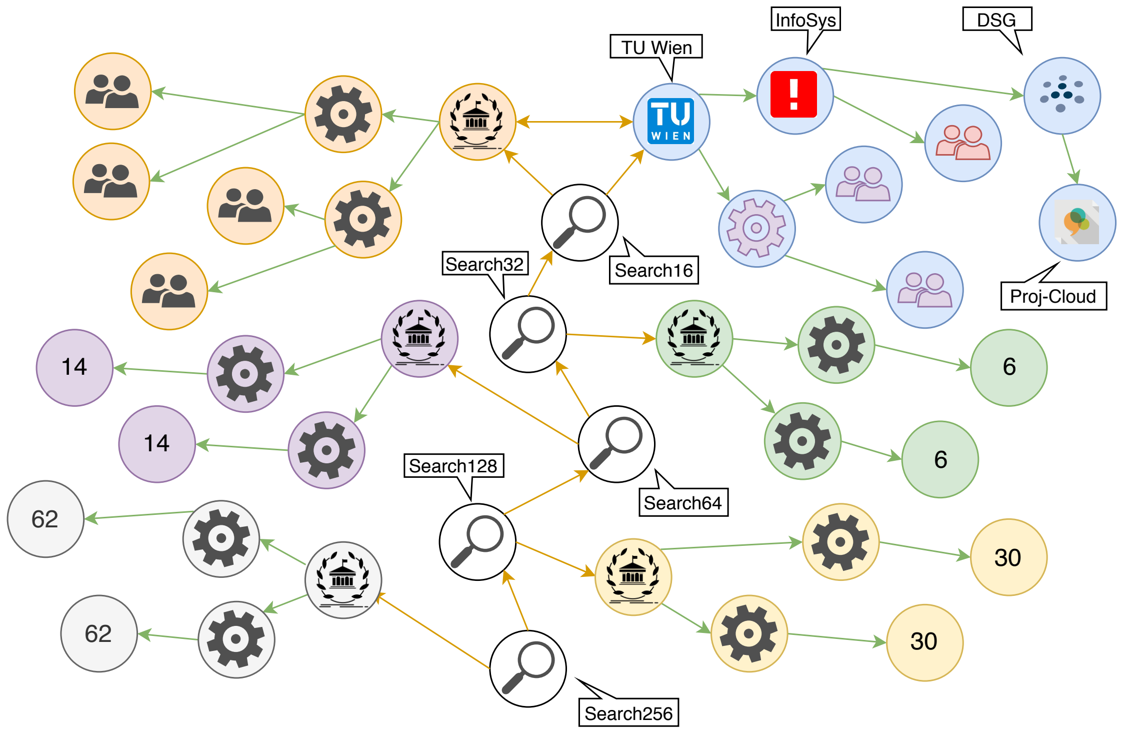


Figure 5.6: Evaluation network as deployed on Ropsten.

background color. Each contract with a number is a place holder for the exact amount of contracts as expressed by the number, where each of these contracts has the exact same characteristics. For example, a green contract with a six inside means that there are six green contracts each with the same connection from the same parent contract. The green arrows represent links with a *trusted*-weight and orange arrows represent links with a *known*-weight. Each unidirectional link in our figure has a counterpart with a *linkback*-weight making the network on the lowest trust level completely bidirectionally connected.

The network was designed with a few interesting characteristics in mind. If we start at the contract labeled as *Proj-Cloud* and follow the path up to the contract labeled as *Search256* we will realize that each step represents a power of two, with *Proj-Cloud* being 2^0 and *Search256* being 2^8. This will be of particular interest when measuring search times in this network. Furthermore, we have defined five so-called search contracts, as demonstrated in Section 3.3.5. Each search contract plus the contracts it links to, results in the number of contracts state in the search contracts name, for example, *Search16* denotes that if searched from this contract downwards we will search a total of 16 contracts.

One last note on the network size. Since we have to deploy each of these contracts and the corresponding links to the blockchain, the deployment process is very time- and

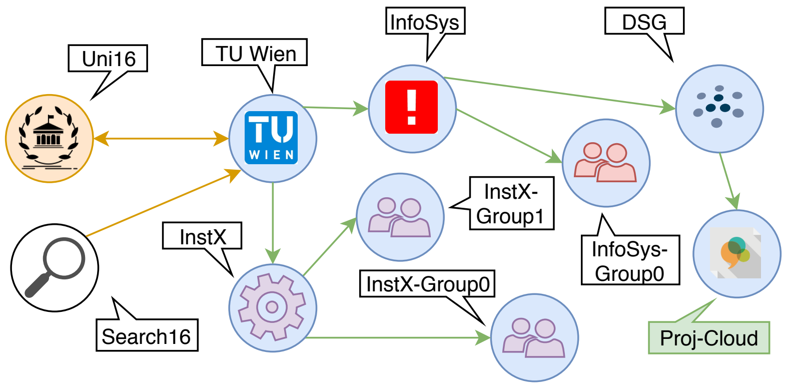


Figure 5.7: Sub-network interesting for Scenario 1.

ether-consuming. We can get free ETH for the Ropsten test network from a faucet, however, this faucet only grants about 15ETH per address, since it assumes this should be enough for any kind of evaluation. This is one of our main restrictions why we did not use a bigger network for our evaluations. The on-chain address of every contract in our test network can be found in Appendix A.6.

Finally, our solution builds upon the idea that contracts propagate certain trust properties towards other contracts, as discussed in Section 3.3.3. Although our solution provides only a very simplified model with four tags, it is completely extensible. The four tags that we provide merely serve as proof of concept and users can build custom trust models by adding their own tags to the provenance network. It is even possible to use different trust models on the same links to satisfy different scenarios.

5.2.2 Scenario 1: Searching for Provenance Data

Our first scenario is the most basic one that there is. The Distributed Systems Group, *DSG*, works on some new project, *Proj-Cloud*. Provenance data for that project gets saved into the *Proj-Cloud* contract. After publishing some papers about that project some reviewers decide to query for the provenance data. To make sure that the provenance stored in the *Proj-Cloud* contract is indeed the official provenance data that is trusted by *TU Wien* they query from the *TU Wien* contract and they include only *trusted* links in their query. An overview of the scenario can be seen in Figure 5.7.

Query: We query the *TU Wien* contract for the resource *http://thesis.eval/scenario1* considering all *trusted* links.

Expected Outcome: As a result, we expect to search a total of eight contracts and find one contract holding provenance about this resource, namely the *Proj-Cloud* contract.

Actual Outcome: Since in our provenance network *TU Wien* uses *trusted* links only for contracts within the organizational border, a search as described

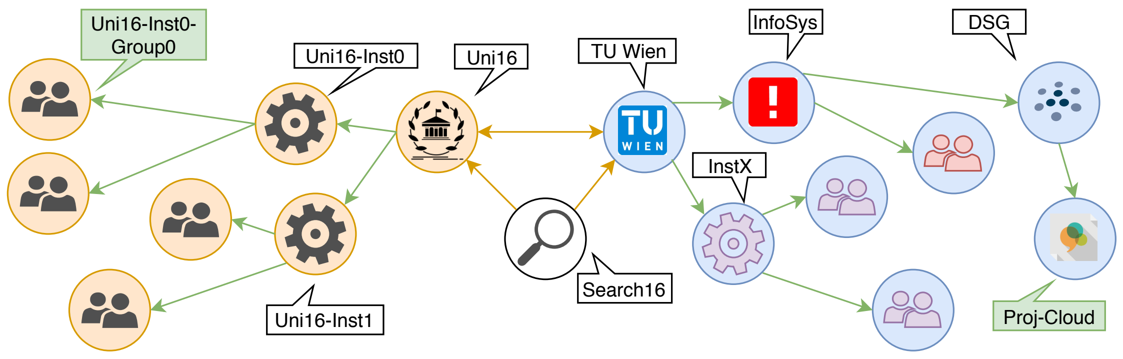


Figure 5.8: Sub-network interesting for Scenario 2.

above returns all contracts which get trust propagated by *TU Wien*. This are exactly eight contracts including the *Proj-Cloud* contract. The search output can be seen in the Appendix A.1.

5.2.3 Scenario 2: Searching between Domains

In our second scenario, we will consider two universities working together on a project. In this case after some contributions on *Proj-Cloud* by *TU Wien* and some conferences some scientists from the University of 16, in short, *Uni16*, decide to collaborate with *TU Wien* on some future work. Together, the two research groups advance their work and produce the resource *http://thesis.eval/scenario2*, which is a derivation of the resource *http://thesis.eval/scenario1*. Since each university has their own parts on which they work, each university writes their respective provenance into their own contracts. Reviewers who want to query for the complete provenance have to query either both of the domains or since *TU Wien* and *Uni16* have a bidirectional *known* link they can simply query any of the domains for the resource while including both *trusted* and *known* links. An overview of the scenario can be seen in Figure 5.8.

Query: We query the *TU Wien* contract for the resource *http://thesis.eval/scenario2* considering all *trusted* and *known* links.

Expected Outcome: We expect to find two contracts holding provenance information, one from *TU Wien* and one from *Uni16*. Furthermore, we expect to search a total of fifteen contracts, those of the blue and orange domains as shown in Figure 5.8.

Actual Outcome: The search results match our expectations. As can bee seen in Appendix A.2, we searched a total of fifteen contracts and found two contracts with provenance information about the searched resource. The *Proj-Cloud* contract which is a contract from the *TU Wien* domain, and the *Uni16-Inst0-Group0* contract which is a contract from the *Uni16* domain.

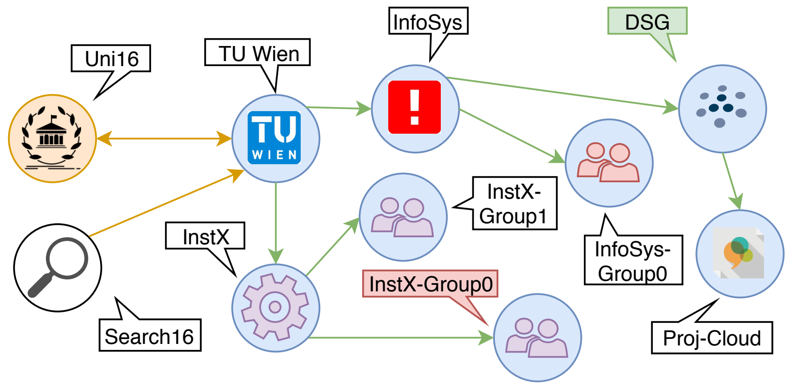


Figure 5.9: Sub-network interesting for Scenario 3.

5.2.4 Scenario 3: Malicious User within the Domain

This scenario will be the first of three scenarios that try to duplicate and hide provenance data. The *DSG* produced the resource *http://thesis.eval/scenario3* during some previous contributions. The provenance data for this resource is saved in the *DSG* contract and thus trusted by *TU Wien*. A malicious user manages to gain write access to one of the provenance contracts, trusted by *TU Wien*. Maybe by an honest mistake of an administrator at *TU Wien*. This malicious user knows that in order to successfully tricking other users to trust into the forged provenance data, this forged provenance data has to be linked to *TU Wien*'s main contract. Thus, after gaining access to one of the *TU Wien* contracts, the malicious user posts forged provenance data regarding the resource, *http://thesis.eval/scenario3*. When users want to check the correctness of the provenance data, they will query the *TU Wien* domain for provenance about this resource. They will expect to find one source of provenance information but will suddenly be presented with two different contracts containing provenance. The real and the fake provenance. Domain experts are now able to determine that one set of provenance information is indeed malicious, inform the administrator, and then the administrator can revoke access of the malicious user to the provenance network. An overview of the scenario can be seen in Figure 5.9, marked in red the contract with fake provenance data and respectively in green the one with the correct provenance data.

Query: We query the *TU Wien* contract for the resource *http://thesis.eval/scenario3* considering all *trusted* links.

Expected Outcome: We expect to find two contracts containing provenance information about this resource and search a total of eight contracts. Furthermore, we expect both contracts to be from the *TU Wien* domain.

Actual Outcome: As can bee seen in Appendix A.3, we found both contracts which are holding provenance information about the resource. Both contracts, *DSG* and *InstX-Group0*, are within the domain of *TU Wien*. Furthermore, a

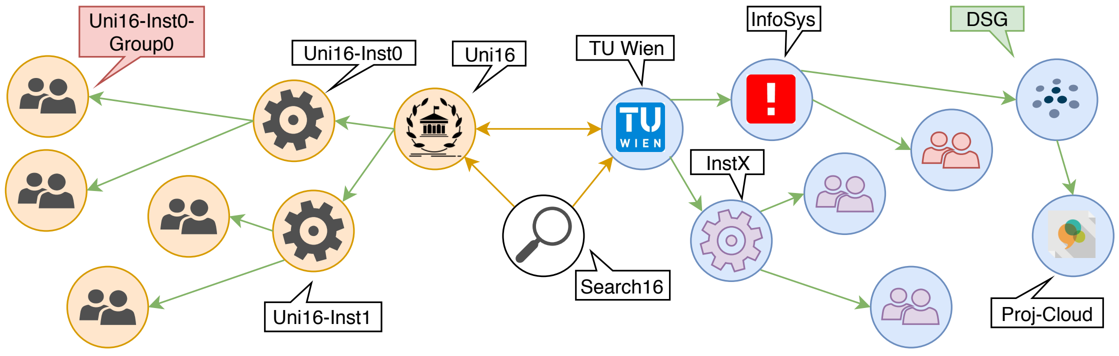


Figure 5.10: Sub-network interesting for Scenario 4.

total of eight contracts were searched. This results show us that a malicious actor can not hide provenance information within a trusted part of the network.

In this scenario, the malicious user managed indeed to publish provenance information to the *InstX-Group0* contract which is formally trusted by *TU Wien*. However, the malicious user was not able to hide the original provenance information which is a necessary step in order to convince other users. Even if the original resource does not exist anymore, users who query for the provenance will always see that there is a duplication and will know that the resource they were given does not fit the original provenance.

5.2.5 Scenario 4: Hiding Provenance in the Network

Similar to our last scenario, a malicious user could try to duplicate provenance by hiding it in another domain. In this case, the target of the attack is the befriended university, *Uni16*. The malicious user again tries to duplicate some old provenance of the *DSG*, *http://thesis.eval/scenario4*. As we know from earlier, these two universities are cooperating on certain projects and have therefore some kind of trust relationship. This means that the duplicated provenance information that was pushed to one of the contracts trusted by *Uni16* has the same trust relationship from *TU Wien*. In this scenario, users can search in three different ways for provenance information. First, users that have only the original resource and therefore a link to *TU Wien*'s provenance domain might search only the *TU Wien* domain leading to them only finding the original and correct provenance information. Second, users who have a fake resource pointing to the modified provenance information may decide to query only the domain of *Uni16* which would allow them to find only the fake provenance information. However, by doing so, this provenance information, strictly speaking, has no trust assurance by *TU Wien*. And third, by extending the search of *Uni16*'s domain to include *TU Wien*'s domain and thus have the trust assurance from *TU Wien*. This way, users would automatically also find the original provenance information and therefore be able to determine that there might be an attempt of duplication going on. An overview of the scenario can be seen in

Figure 5.10, marked in red the contract with fake provenance data and respectively in green the one with the correct provenance data.

In this scenario, we will perform all three of the described queries:

Query 1: We query the *TU Wien* contract for the resource *http://thesis.eval/scenario4* considering all *trusted* links.

> **Expected Outcome:** We expect to find only the original provenance data and to have queried a total of eight contracts.
>
> **Actual Outcome:** As can be seen in Appendix A.4.1, we searched exactly eight contracts and found only the *DSG* contract with provenance information, as expected.

Query 2: We query the *Uni16* contract for the resource *http://thesis.eval/scenario4* considering all *trusted* links.

> **Expected Outcome:** We expect to find only the fake provenance and to have queried seven contracts. Furthermore, we expect that the *TU Wien* main contract is not part of the searched contracts.
>
> **Actual Outcome:** As can be seen in Appendix A.4.2, we found only the fake provenance information stored in the *Uni16-Inst0-Group0* contract. Furthermore, also as expected, we searched seven contracts in total which did not include the *TU Wien* contract.

Query 3: We query the *Uni16* contract for the resource *http://thesis.eval/scenario4* considering all *trusted* and *known* links.

> **Expected Outcome:** We expect to find the original and the fake provenance information and to have queried fifteen contracts in total.
>
> **Actual Outcome:** As can be seen in Appendix A.4.3, we found, as expected, both contracts containing the real and fake provenance information. Furthermore, we searched a total of fifteen contracts.

This scenario has a special case when the malicious user is a scientist in *Uni16*, who tries to claim a certain resource originally produced by *TU Wien* was produced by *Uni16*. Since in this case the fake resource would claim to be originally by *Uni16*, users might be inclined not to search more than the *Uni16* domain for provenance which would produce the same results as by *Query 2*. In this special case, we have to distinguish two different possibilities. First, the malicious user changed the URI of the resource, which makes the resource from a provenance viewpoint a different resource than the one at *TU Wien*. In this case, it is up to other mechanisms to identify the resource as a duplication attempt. Second, the resource URI stays the same. In this case, the fake resource would need to be faked in such rigorous detail that it is not possible to connect it to *TU Wien* so that

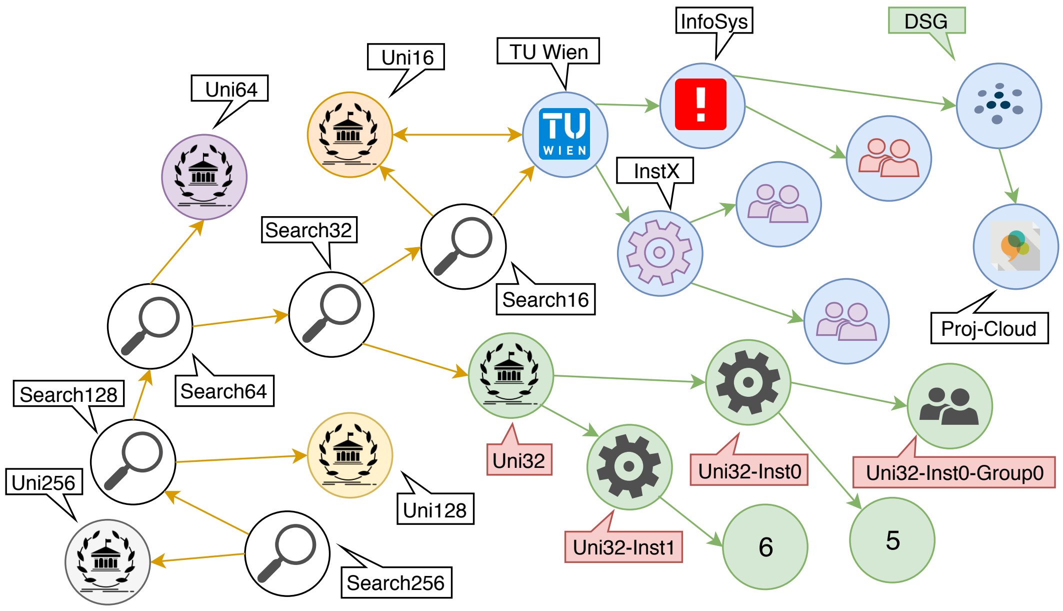


Figure 5.11: Sub-network interesting for Scenario 5.

users are not inclined to search the *TU Wien* domain. Even if this is successfully done, any user with domain knowledge of the resource who has seen both resources will likely figure out that one of them must be a duplication and will be able to find all the relevant provenance information.

As can be seen, our mechanism is limited to preventing the duplication of provenance data as discussed in Section 3.2, however, not necessarily the duplication of the resource itself. In this case, our solution still provides a reliable way of searching for related provenance and proving that some resource is a duplication attempt of another.

5.2.6 Scenario 5: Duplicating an entire Domain

In this scenario, a malicious user could try, by utilization of enormous amounts of capital, to duplicate the entire *TU Wien* domain. For this scenario, we assume that the attacker is not capable of changing the officially announced address of *TU Wien*. Users who query this fake provenance network will then realize that the real address of the *TU Wien* main contract is not part of the searched addresses. Furthermore, the attack can not stop any search contract provider from linking to this fake network which decreases the chances of deceiving anyone further. This is due to the fact that once added to a network which also references the real *TU Wien* network, any user who performs a general search on a very low trust level, as for example the *linkback* level will find the original as well as the fake provenance information. An overview of the scenario can be seen in Figure 5.11, marked in red the fake domain and respectively in green the contract with the correct provenance data.

In this scenario, we will, therefore, perform three different queries:

Query 1: We query the *TU Wien* contract for the resource *http://thesis.eval/scenario5* considering all *trusted* links.

- **Expected Outcome:** We expect to find only the original provenance data and to have searched a total of eight contracts.
- **Actual Outcome:** As expected, we searched a total of eight contracts and found only provenance information in the *DSG* contract. The results ca be seen in Appendix A.5.1. This results show us that a network replication will not disturb users which already know the real address of the *TU Wien* domain.

Query 2: We query the *Uni32* contract for the resource *http://thesis.eval/scenario5* considering all *trusted* links. The *Uni32* contract herby represents the main contract of the fake domain.

- **Expected Outcome:** We expect to find only the fake provenance data, to have queried a total of fifteen contracts, and not to find the TU Wien contract among the searched contracts.
- **Actual Outcome:** As expected, we searched a total of fifteen contracts. We found one contract holding provenance information, the *Uni32-Inst0-Group0* contract which is affiliated with the fake domain but not with the domain of *TU Wien*. Most importantly our searched-contracts result list did not include the *TU Wien* contract, as can be seen in Appendix A.5.2. This results show us that a network replication can attempt to fool only users who do not take the time to verify affiliation to announced contracts.

Query 3: We query the *Uni32* contract for the resource *http://thesis.eval/scenario5* considering all *trusted* and *linkback* links.

- **Expected Outcome:** We expect to find the real and the fake provenance contract and to query a total of 256 contracts.
- **Actual Outcome:** As expected, we found exactly two contracts holding provenance for this resource, the *DSG* contract and the *Uni32-Inst0-Group0* contract. Furthermore, as can be seen in Appendix A.5.3, we searched a total of 256 contracts which is in our case the whole provenance network. This results show us that hiding provenance in a connected network is not possible once the whole network gets searched.

To be really successful with this attack, the malicious user would need to hack into *TU Wien* and change the address of the announced main contract. This is not an easy task in the first place and would be reversed once discovered by the *TU Wien* administration. Furthermore, the real *TU Wien* provenance contract has at this point already some prestige, meaning, that the attacker would need not only to hack into *TU Wien* and

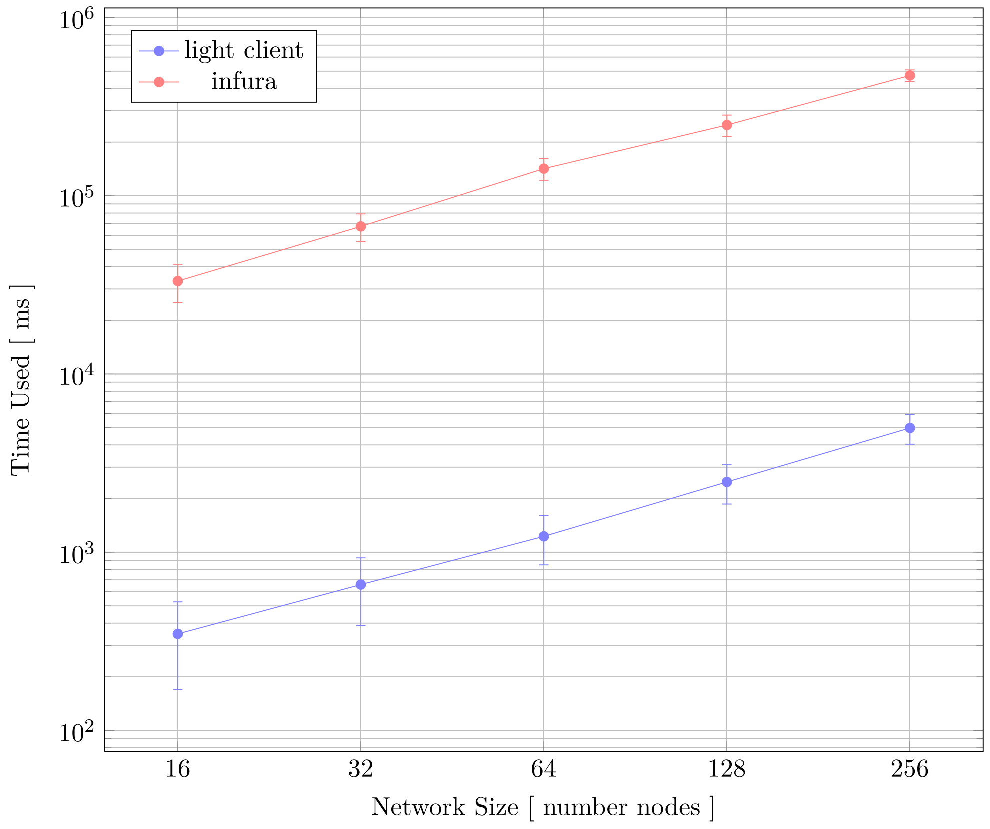


Figure 5.12: Measured average time to search graph based on graph size (logarithmic scale).

change the announcement without anyone realizing but also manage to change it in all the other places where it is referenced, for example, *Uni16*, and any published resources by *TU Wien*. Finally, additional security is provided by the contracts in the *TU Wien* domain themselves, since anyone who uses already a correct contract, for example, the *DSG* contract to query or store data will not simply decide to switch to a new unknown contract in a fake network.

5.2.7 Search Times

For this evaluation, we use two different connections to the blockchain and we run every query multiple times, taking the average value of all measurements. We decide to take the average and not the median to purposely include extreme times as needed when caches are populated for the first time to make sure that one time and first time searches are not surprisingly slow. Furthermore, we perform the measurement with two different ways of being connected to the Ethereum chain. In the first case, we connect with a locally running node in *light* mode, meaning that the node is only synchronizing block

Table 5.1: The measured average times and the corresponding variances.

	Average Time	Unbiased Variance
Local-16	348.25 ms	178.23 ms
Local-32	658.22 ms	271.31 ms
Local-64	1.23 s	378.22 ms
Local-128	2.48 s	615.19 ms
Local-256	4.98 s	941.86 ms
Infura-16	33.23 s	8.07 s
Infura-32	67.33 s	11.82 s
Infura-64	141.98 s	19.74 s
Infura-128	249.45 s	34.14 s
Infura-256	473.22 s	35.24 s

headers and contacts to an archive node when validation or extra data is required. In the second case, we connect to the Infura public nodes. Both connections are done by HTTP and we perform each local query 100 times and each Infura-based query 10 times. The second was performed fewer times since it takes significantly longer and we did not want to put any unnecessary load onto the Infura nodes.

As can be seen in Figure 5.12, the network can be searched quite fast when connecting to a local node. In case of the locally running *light* client we are achieving times of around $5s$ to search the entire network of 256 nodes. The absolute worst case for a 256-nodes network was $10s$ and with an average around $5s$ we are able to perform search queries which will not disturb auditors in their work. This is probably mainly due to the light node running locally and allowing us to query the network only for specific information when needed. We expect search times to become even better if we deploy a *full* node on premise since then we would not have to leave our local network at all, since the *full* node would be in sync with the chain and able to answer our queries immediately. A *full* node on the same machine as used to query the provenance network would, even further, improve the search time since all the data needed would be literally on the same machine. As we can see by using the Infura public nodes to connect to the blockchain, our query time receives a penalty of around a factor of 100. This is probably mainly due to the network communication and to higher load on the Infura nodes which are used by a lot of applications. If we look at the sample variances, as shown in Table 5.1, we can see that especially in the case of the local measurements they are quite high. Meaning that our results have a high dispersion which is in general not good. However, if we consider the time frames about which we are talking, in case of the local node, we have to assume that this behavior is mainly due to caching strategies and implementation details of the local node. After all, the smallest network we measure is searched in under $350ms$. If we look at the Infura-based results the variance is in general smaller. We assume this is mainly due to the network access times which are significantly bigger in this case and have a smoothing effect on the measured times. In both scenarios, local-

and Infura-based, we experience a relative improvement of the variance with the size of the searched network. This is probably due to the fact that once a certain network size is reached the necessary work to search the network begins to outweigh the side effects like implementation details, caching strategies, and network access times.

Besides the obvious cost advantages of having the search algorithm not run on the blockchain, we get a few other advantages. First of all, maintainability. Since the search algorithm is running in the local code, errors in complex search algorithms can be easily fixed without the need to redeploy contracts to the chain. Second, extensibility, since the search algorithms are running locally it is easy to experiment with new search algorithms on the network without incurring new cost. And finally, having the search locally means that we have a natural distribution of the computational needs. Although the blockchain is distributed in nature, the single nodes have still to perform the required computations on reading requests, also in case of connections over Infura, we would be offloading all the computational requirements to one node. By having the search client side, every client itself has to provide the necessary computational power to be able to deal with the search requests and organizations are able to provide more powerful machines to improve the search times of their clients where necessary. This provides a natural load distribution not only across nodes in the blockchain but also across the blockchain-utilizing clients. One disadvantage of this approach is of course that the search time is dependent on the local client and thus can vary from client to client.

5.3 Non-Functional Evaluation

In this section, we will discuss how our solution satisfies the non-functional requirements which we have identified in Section 2.4.

5.3.1 Confidentiality

Our solution does not provide a specific approach to solve confidentiality. The commonly used approach of encrypting the data as discussed in Section 2.4.2 can be used together with our solution since we do not limit the users to a specific model or strategy. This also has the advantage that in case of solutions where the data is off-chained we do not add complexity to both the code as well as to the data. This is particularly interesting when it comes to keeping the cost low as we saw in Section 5.1. However, this has, of course, the disadvantage that if the users do not take care of securing the confidentiality we will not do it for them which can lead to unintentional plain text provenance in the blockchain.

5.3.2 Integrity

On this property, we have to consider two different aspects, since integrity can be enforced by the utilized transport protocols and furthermore it needs to consider storage properties disallowing manipulation of stored data. Regarding the first property, transport integrity, we are not providing any specific solution, however, we are supporting any integrity strategy that is ultimately encoded into the stored data. For example, if we consider a solution that uses HMACs, then the integrity verifying part will be attached to the end of the provenance data. Once stored in the blockchain, we can easily verify that the data was correctly stored by simply reading it and verifying the HMAC. This allows users of our solution to utilize different integrity strategies without the need to specifically support them in the backend, being in our case the blockchain. This is a direct result from the second property, the integrity of stored data. Since once stored into our provenance contract, the provenance data cannot be manipulated anymore, we are providing long-time data integrity by utilizing the natural properties of the blockchain itself. This means, for example, in the case of off-chained data that the hash verifying the data is saved for as long as the blockchain exists and can extend the property of integrity onto the original data. The same security is provided for on-chain data. And finally, this is the reason why we can perform integrity validation also on the client side. Since once the data is written it cannot be changed anymore, this also means that to make sure that the data was not damaged by the transport we can simply write it to the blockchain and then validate it by reading it out again. In case our integrity validation strategy discovers a mistake, we can simply rewrite the data again since the first attempt will also be discarded as invalid data by any other client utilizing our integrity strategy.

5.3.3 Availability

Availability is in our case somewhat design-dependent. In case of on-chain data, we provide very high availability since one would need to take down the entire blockchain network to take down our provenance data. However, as we saw in Section 5.1, saving all the provenance data in the blockchain is rather expensive, which brings us to off-chain solutions. For those, we provide on our end still a very high availability. Meaning, that the hashes will have the availability assurance by the blockchain, however, we can not say anything about the actual provenance data since this would highly depend on the actual solution and where this data is saved. It opens, however, interesting possibilities for security uncritical data. This provenance data could then be freely replicated by interested users and whenever a party needs to verify that the provenance data presented are an unmodified copy of the original, they could crosscheck with the blockchain. For example, public provenance data could simply be stored into git repositories, which allow for easy duplication and distribution. Whenever users clone a repository the provenance data could be verified by checking the hash on-chain.

5.3.4 Non-Repudiation

As has already been discussed in Section 3.1.1, non-repudiation depends on the ability of the provenance providers to map private keys to certain users. Otherwise, especially in the blockchain environment, anybody could create new keys and claim that a previously used key was not theirs. We improve this state by implementing out of the box a rather strict access control. Users who want to write provenance data to a certain contract must first contact an administrator of this contract and request write permissions for the contract. The administrators are then able to fulfill any kind of internal requirements of the respective organization to map that key to the real persona without exposing sensitive data of that person to the whole world. This way, when a certain user requests that a key gets renewed, two things happen. First, an administrator can inquire why the key has to be renewed. Second, the administrator can keep, when necessary, a complete history of which keys belonged to which persons. An interesting extension would be to solve this problem of matching users to keys in an on-chain way without necessarily exposing user data.

5.3.5 Unforgeability

The main property we aimed to improve with the work in this thesis is unforgeability, more specifically provenance duplication. As we have already discussed, provenance duplication is part of the forgeability attacks since a malicious actor uses a second set of forged provenance data to hide the original provenance data behind, as discussed in Section 3.2. As we have shown in Section 5.2, with help of a scenario-based evaluation, hiding provenance data in a fully-connected provenance network is not possible. Furthermore, we have shown that hiding provenance in a provenance network is only possible in exchange for the loss of trust propagation, meaning an attacker can only hide provenance from another contract if the trust level is set in a way that a search will not regard the malicious contract as relevant. However, this also means that this malicious contract does not get any trust propagated by other important contracts and should be disregarded. Equally, if a contract claims to have provenance about some resource which is known to be part of a bigger organization but the contract is not part of any provenance network, it again cannot be trusted and should, therefore, be disregarded. This makes it effectively impossible to hide provenance information and retain trust propagations in a provenance network. Combined with the natural integrity properties provided by the blockchain, it becomes to the best of our knowledge impossible to forge the provenance data given they were properly recorded. The unforgeability of the resource itself is dependent on the actual solution and how the provenance data interacts with the resource data.

5.3.6 Granularity, Model and Storage

We will discuss these three properties together since they have a very simple common answer: Our solution is granularity, model, and storage agnostic. As discussed in Section 3.1, one of our major goals was to provide a common place where provenance can be searched and found on a design-independent level. To achieve this, we decided purposely to stay model-agnostic. Storage-agnostic was another decision to allow us to cover a broader set of provenance solutions and to enable off-chaining which is certainly necessary as seen in Section 5.1 and discussed by Eberhardt et al [14]. These two decisions brought granularity automatically with them since granularity is often model- and use case-dependent. Furthermore, in the case of off-chained data, it does not affect us at all.

5.3.7 Chain

We implemented our solution with the Ethereum chain. This is mainly due to the fact that it is one of the most-commonly adopted chains that provides a higher programming language. Furthermore, it is a public chain and thus does not require any special pre-requirements from users to use it. This said, in theory, our solution should be mappable to other chains that support a complex enough script language and smart contracts. Our solution does not make use of any special Ethereum features aside from smart contracts. In case of mapping to another chain, it would also be of interest how cross-chain linking could be established to allow for provenance networks across chains to be linked to each other and improve the overall security of both networks.

5.3.8 Comparison

After discussing the different non-functional properties of our solution, we will compare our solution to other solutions as presented in the related work Section 2.4. For this we extended the tables to contain our solution, as can be seen in Table 5.2 and Table 5.3. The non-security related properties can be seen in Table 5.2. As can be see, we offer full flexibility by being granularity-, model-, and storage-agnostic. This allows implementing all the solutions presented in the related work on top of our provenance network architecture. Doing so allows us to search for provenance information in a domain-independent way. As can be seen in Table 5.3, the decision to be model-agnostic has also consequences for the security properties. We are able to support all the relevant security properties given that the chosen use case implements them on top of our solution. This can be easily done by extending our contract architecture and implementing specialized provenance contracts which have stricter security enforcement at the cost of higher gas usage. Another way would be by building clients which fulfill the special needs of the different use cases. At the same time, we do not enforce certain security patterns onto use cases using our solution. This allows for public use cases to stay that way and furthermore, does not impose any unnecessary cost for functionality which might not be needed by every single use case. This flexibility allows us to propose provenance networks as a common ground solution for solving the duplication issue and for solving the searchability issue, as presented in Chapter 3, without restricting use cases in their model- or security-related choices.

5.4 Code Quality

During the evaluation of our solution, we had to realize that there are still some bugs in our code which we track on Github[1] for future development. None of the bugs discovered did affect our evaluation or the concepts which we are demonstrating in this work. All but one of the bugs are in the *React-Client* and therefore not critical at all. This is because client-side bugs can be easily fixed without consequences for the deployed contracts. Most

[1]https://github.com/vauvenal5/ProvNet

Table 5.2: Properties of the related work compared to our solution.

	Granularity	Chain	Model	Storage
DataProv [39]	coarse	Ethereum	OPM	on-chain
ProvChain [12]	coarse	Tierion	custom	on-chain
Stoffers2017 [40]	custom	BigchainDB	W3C PROV	on-chain
TOVE [43]	coarse	Ethereum	custom	on-chain
BlockPro [44]	fine	Ethereum	model agn.	on-chain
Neisse2017 [13]	custom	Ethereum	model agn.	on-chain
Massi2018 [45]	custom	Hyp. Fabric	W3C PROV	on-chain
ProvNet	**custom**	**Ethereum**	**model agn.**	**custom**

Table 5.3: Security properties of the related work compared to our solution.

	Integrity	Confid.	Avail.	Non-repud.	Unforg.
DataProv [39]	high	mediocre	high	supported	supported
ProvChain [12]	mediocre	mediocre	high	partial	partial
Stoffers2017 [40]	mediocre	low	high	custom	not avail.
TOVE [43]	mediocre	low	high	custom	custom
BlockPro [44]	high	low	high	supported	supported
Neisse2017 [13]	mediocre	high	high	supported	supported
Massi2018 [45]	mediocre	mediocre	mediocre	partial	partial
ProvNet	**high**	**custom**	**custom**	**supported**	**supported**

notable is the one bug which we found in the backend. This bug prevents us from deleting links between provenance contracts. This bug does not affect the results presented in this work in any way. However, it is of high importance to fix it as soon as possible since it affects the long-term maintainability of the contracts. Furthermore, we discovered that deploying our provenance contract has become quite expensive. This combined with new architectural guidelines for the development of Ethereum-based contracts will make a restructuring of our backend architecture necessary. However, this restructuring will be only from a technical viewpoint to align the backend with new architectural recommendations. It will not affect the overall functionality or the discussed concepts of our solution in any way.

5.5 Summary

As can be seen, our solution has a few clear limitations. First of all, the strength of our solution grows with the size of the network allowing more and more provenance to be properly audited. For example, if we have only one contract which is not interlinked with any other contract, our solution cannot help to improve the trust into this contract. Only after linking this contract to others and making it visible to the world, we can infer that this contract does not try to hide anything. Second, our solution can not help when a malicious user tries to completely forge a resource. For example, if one scientist tries to steal the results of another, we are not able to identify this since from a provenance viewpoint these are two different resources and it is up to domain experts to identify that there are two resources claiming the same. However, once the resources in question are identified, we provide an easily searchable solution to finding the respective provenance data and thus support the process of dismantling the forgery, even if the resources originate in different domains or use cases. Third and final, in case that a user has to search regularly the entire provenance network, this user should deploy a local Ethereum node to improve query times.

These limitations aside, our solution provides a real model-, domain-, and use case-independent solution for finding and mapping provenance. And with no model-dependent parts in the backend, namely the smart contracts, a user can build front end clients as required to establish collaboration between models, domains, and use cases. Thus allowing every solution to use the methods best fitting its domain without restricting its ability to collaborate with other solutions. For provenance auditors, a complex image about provenance data can be created and investigations into resource forgery can be supported by an easily searchable network. Ideally, this can pave the way to extend this simple search to a cross-domain provenance analysis tool.

On the security side, we are providing a solution that does not limit individual users and domains to a certain model but as security algorithms and practices evolve so can the used methods evolve without the need to change the smart contracts in the backend. Every use case can choose the required level of confidentiality and integrity checks as required. Furthermore, every use case has the availability assurance as provided by the blockchain. And most importantly, we are making it a lot harder to fake provenance data once recorded, or to fake a resource by duplicating and hiding its original provenance data. Simple, initial access controls allow us to confirm the authenticity of private keys and thus to harden the non-repudiation property of any provenance data in the chain. And last but not least, by introducing an easily customizable solution through link weights, we allow for the use of different trust propagation models on top of the same network, so that use cases can propagate trust as they see fit.

CHAPTER 6

Conclusion

In the introduction, we motivated our work with a simple Web service-based workflow example, where some input data passes a complex flow of Web services and human actors resulting in some output data. To be able to build trust in this output data and to be able to reproduce the process involved in creating this output data, we employed data provenance. Data provenance is a type of metadata which helps us to make deductions about a data product and how it came to be. To be able to provide trust for the original data product, this provenance data has, in turn, to be protected. As we saw, this combines the already complex domain of data provenance with the domain of security. Many different solutions exist which provide domain- and use case-based approaches towards solving the issue of securing provenance data in their specific context. Furthermore, different approaches of combining the blockchain as a trust-provider with data provenance started to appear in recent years. The blockchain provides by design some strong security assurances towards some of the security properties as required by secure data provenance.

The huge amount of different approaches towards data provenance without any common basis to work with led us to one of the problems which we identified as the searchability issue. The other issue which we identified is the duplication issue. Although the blockchain provides strong security for the integrity of data, it does not natively solve the issue of simply duplicating records and hiding original records behind these duplications. In order to address both of these issues, we introduced the concept of provenance networks. A provenance network is a network of linked provenance contracts, i.e., smart contracts, which are designed for storing provenance data. These provenance contracts allow each other to be linked in form of cyclic, directed, weighted graphs, as commonly used by trust networks. This way, a provenance network provides a searchable space where provenance data can be searched for. By introducing the notion of trust on top of this search space, we are able to create a generalized approach for solving the duplication issue. Furthermore, our provenance network concept is model-agnostic, which makes it possible for different other solutions to solve their specialized requirements on top of our provenance networks. This allows us to extend the usability of our search space beyond the duplication issue to also solve the searchability issue. Thus providing a search space across different domains, use cases, provenance models, granularities, and storage models. One major factor that makes all of this possible is, the fact that for utilizing our solution, no single server has to exist since the backend of every provenance network is in the blockchain. The blockchain is highly distributed by nature and thus a perfect choice for supporting provenance networks.

With a scenario-based evaluation, we showed that our concept of provenance networks can support a variety of different scenarios and most importantly allows for finding duplication attempts within the same network. At the same time, it fosters collaboration between different domains by allowing them to query a more complete provenance picture across different domains. Collaboration is further supported by our model-agnostic approach and our light backend allowing for domains to build their use case specific clients and even works across different provenance models. Furthermore, we showed during our evaluation, how important off-chaining is with respect to keeping the provenance system cheap and useable. This is an aspect that many other solutions neglect by saving their provenance directly into the chain. We also showed that even big provenance networks can be queried rather quickly by using local nodes, which can be of huge importance for auditors. At the same time, we built small and simple clients which allow to use and create provenance contracts without the need for a complex setup. Finally, we discussed non-functional properties – most prominently security properties – and showed that our solution is arbitrarily extendable to supports complex provenance scenarios also with respect to security properties.

6.1 Future Work

This thesis merely introduced the very basic concept of provenance networks. Now that there exists a basic framework to work with, it would be interesting to evaluate how different trust propagation strategies behave in combination with data provenance. Another interesting point is how we can extend our solution to be able to link provenance networks across different chains. Since the idea of provenance networks is in general independent of the used chain, as long as the chain provides smart contracts. Beside cross-chain linking, we should investigate provenance migration. This is interesting for different use cases. One is a technical use case since the current prototype has no evolution strategy implemented. Meaning when the backend contracts get extended with new functionality, we should have a strategy how to link or to migrate the provenance in the old contracts to the new provenance contracts without invalidating any of the security assurances. Another is cross-chain migrations. The next generation of blockchains could have even better properties as the current one and we should think of ways how we could migrate provenance across chains without losing the security assurances. Finally, we have to make some improvements to the current backend to be even more flexible and customizable, which would be an ideal moment to include an evolution strategy at least for the Ethereum-based solution.

APPENDIX A

Evaluation Results

A.1 Results Scenario 1

Listing A.1: Scenario 1 results for finding provenance in the TU domain.

```
{
    "search": {
        "Search16": "0x330c2646ea6be38625ce3b15957738820b31370a",
        "Search64": "0x7487d90e1faff5c569ae99b1453adceaabd87c8a",
        "Search32": "0x1747bae0546a80818e974eb81e99c7846099dd12",
        "Search128": "0x82a6f0217348ce9af9ea6d7d54aa41ddda06de3e",
        "Search256": "0x239249e1bb5859cb5214555e06dfb08260ee64ae"
    },
    "domains": {
        "TU": "0x55526b860d8fc67bef7440e236c02231acb12d90",
        "Uni16": "0x75735b7a532108ea0760ba4b4841d399e16f1fd7",
        "Uni32": "0x78Af41B30Bd48d94965A8fc3fA563FE2950fD638",
        "Uni64": "0xfd840eca0bdf85a350f0cea0f5b5a9ecd0322793",
        "Uni128": "0x9fa8d22001c79d4c0d2607c97541ef46cfc0ee15",
        "Uni256": "0x188b3f7579695501dddb5e155dcae3d187e73cb7"
    },
    "blue_domain": {
        "InfoSys": "0xcf9efa13aa7b5600fe263676bf7d6d19e7bda56e",
        "DSG": "0xb0ab80a55113eeb9c5ee65fb332ed8b57c191196",
        "Proj-Cloud": "0xb54cb4313f6fa7bf51d7250209b324efe26b8984",
        "InfoSys-Group0": "0x45374493b34fce298bfcd8f53074dc7798e9ed1d",
        "InstX": "0x1d1d28c13a0816948dc64fa3184b845cd80aa49e",
        "InstX-Group0": "0x93ba312f68a08c9d7be0ac603baf4123f6c99827",
        "InstX-Group1": "0x385134ff0e51505f7f109ce170b8763c76ed845c"
    },
    "orange_domain": {
        "Uni16-Inst0": "0x36c0322cc017dad799bbfd10968f91b1e52591b4",
        "Uni16-Inst1": "0xcfcca9d7cf92178787410073 9be986488ce0ec9c",
        "Uni16-Inst0-Group0": "0x07c431f898ec376cb51348092eda10a5f4d21159",
        "Uni16-Inst0-Group1": "0xef12512047eb269fa69a8aaffe888e7ded12ea5f",
        "Uni16-Inst1-Group0": "0x8358f3f2b3a975c3f772b0c5f5abd784ca1fd98f",
        "Uni16-Inst1-Group1": "0x936738f2daad1ecbd823ca550429439972c128ad"
    },
    "green_domain": {
        "Uni32-Inst0": "0xa5631e23a8215b4dedc660052146a8146cd4a55a",
        "Uni32-Inst1": "0x2f53d70e1ba5cd94fa98ed97ccae3bfde8c0b4e4",
        "Uni32-Inst0-Group0": "0xdc6ecb9bf091304b5dbd2e57d8e9d082a0bbeb79",
        "Uni32-Inst0-Group1": "0x08604c0b0dce181fe9764d8bccb4324ad6704a23",
        "Uni32-Inst0-Group2": "0xe71bdff910c73a8436704cb697be429ce2f87d3b",
        "Uni32-Inst0-Group3": "0x886634923f31c128cd9084155331a70ea5eec806",
        "Uni32-Inst0-Group4": "0x82babd4a986951dee21ca53c41643cbf20a33de7",
        "Uni32-Inst0-Group5": "0x329d47434a7041885a81e026165dc87b4fccb7a8",
        "Uni32-Inst1-Group0": "0xfa2e2390039595fbc1cb7d9ffc51924aeac6fb04",
        "Uni32-Inst1-Group1": "0x31dfd5b73628338b67fd7421a49ad8c45525d87b",
        "Uni32-Inst1-Group2": "0x902dd87cdd4037b5209ad4fd4f69464fbc194d0e",
        "Uni32-Inst1-Group3": "0xf91fca430f60a30bc4f4378caf925da2bef8875f",
        "Uni32-Inst1-Group4": "0x9382a3f6083650ebada45292b65dce9762175caf",
        "Uni32-Inst1-Group5": "0x6a24294d3279fdb0250fb06068d1869be1a27441"
    },
    "violet_domain": {
        "Uni64-Inst0": "0x31e91fb905d8fbf32b7491c21640c57a14dc0372",
        "Uni64-Inst1": "0x8ce0daf8512027337b4b895533602c658ba07e0f",
        "Uni64-Inst0-Group0": "0x6aa348c583e47635672c1829098d5eeef10b692b",
        "Uni64-Inst0-Group1": "0x3b6cf5ebf16dccfbc6272aad9242e9ed48dc57ae",
        "Uni64-Inst0-Group2": "0x4e4fa3829cb9b08d33a194bfc5eb46c0c542b43a",
        "Uni64-Inst0-Group3": "0x4f01e833272911a2b2ce8e048866169fb736a0ae",
        "Uni64-Inst0-Group4": "0x9fc19ea263313dc7373107a5919831b08395151e",
        "Uni64-Inst0-Group5": "0x42d32e19aecea1b8c7ba8dd6e4469f5dcbeb5ab7",
        "Uni64-Inst0-Group6": "0xe5f5b23fc365aeaa3ae99d1fecda0a29b5216f68",
        "Uni64-Inst0-Group7": "0xda4957e502e621630cca0ca883e26d26bc74d716",
        "Uni64-Inst0-Group8": "0x06f2f06ab8ee1489277091c3f5f94920a6926ed1",
        "Uni64-Inst0-Group9": "0x35e2043e609490304909D1883BAbb2a577a8313F",
        "Uni64-Inst0-Group10": "0x61f5c4b8614bd892771149da2da10d911295bbb2",
        "Uni64-Inst0-Group11": "0x324a039343fa5028e7aabce7abab27dbcf5c8c3c",
        "Uni64-Inst0-Group12": "0x70a0e88f73f1444d7f0c7aafda53aec9ead04f48",
        "Uni64-Inst0-Group13": "0xa19c556bae1298026417ca84142ba3fbff4714e3",
        "Uni64-Inst1-Group0": "0xa14283052c33fe4658eff5f3e2b176daff66bb0f",
        "Uni64-Inst1-Group1": "0x8ac22fee88b6c1abb6c360e38b63d3a959f2cb44",
        "Uni64-Inst1-Group2": "0x43d6988b0910938c260249f383b2dd22773119cb",
        "Uni64-Inst1-Group3": "0xf0f761ecdcb7faad212a11eb421572774f0f0b5c",
        "Uni64-Inst1-Group4": "0x7c3ace277b70107c1947aaeee1ff40a1e7393f31",
        "Uni64-Inst1-Group5": "0x847e07e202cfd278c7eed8cd51eb58df14c0cff0",
        "Uni64-Inst1-Group6": "0x29413f5768f194d5a7d6680cbb44d0ff0d44230d",
        "Uni64-Inst1-Group7": "0x1f6d920b5ac13b9c9f0ed57b98edda65839bb72a",
        "Uni64-Inst1-Group8": "0x05ee15dacc9c55b2dbafd6b8260a867567ab85ce",
        "Uni64-Inst1-Group9": "0xe44a17530dc897f713259dd35ce2d60b0e6f22fe",
        "Uni64-Inst1-Group10": "0x7be66ed3dc4e0b50aad410583f6e11e23e935894",
        "Uni64-Inst1-Group11": "0xa9fd8e06619c7a6bc83852ba24ff9b5f28da84ed",
        "Uni64-Inst1-Group12": "0x17d825146cbb9eb895b8fb10630e30eb47626307",
        "Uni64-Inst1-Group13": "0x36699f19d77d77401062982ce7bfe8ee9dc58fcb"
    },
    "yellow_domain": {
        "Uni128-Inst0": "0x7b7fb11557807db7256c75917e60e9892e95ce2c",
        "Uni128-Inst1": "0x0fd380ba625edc8b86ae09d8e8fdf50e823124ea",
        "Uni128-Inst0-Group0": "0x8eade39e64cb0d1035966a2147de7971aa08571f",
        "Uni128-Inst0-Group1": "0x287ce5250cda9fdb22925d97601b5b4d7b6ceecf",
        "Uni128-Inst0-Group2": "0xdd59d748a33e8897aad6f7f3212a31993479ec9f",
        "Uni128-Inst0-Group4": "0xafab4d841d86fd5d029ca2f3b6a41c56002d7ad4",
        "Uni128-Inst0-Group3": "0x0933f168829653ed4a62c258834073c5d53ac8db",
        "Uni128-Inst0-Group5": "0x495e745fb593f4501c603c2299d15b9e45287081",
        "Uni128-Inst0-Group6": "0xd0aa3e92bdd4b9e1f6de13f4c4f4b7adff5f52af",
        "Uni128-Inst0-Group8": "0x207d0461c60e8a4c8fb7cc7b7ed1b0623d546f97",
        "Uni128-Inst0-Group7": "0x56b6cca90d73f479998707658b8dc88533835ab8",
        "Uni128-Inst0-Group9": "0x2c884ccb2a021920bfeabf614fb75416d60b8b23",
        "Uni128-Inst0-Group10": "0x4470351a92d89cc511447c8bf230f7315658bfba",
        "Uni128-Inst0-Group11": "0x11e9970b812dae51c40b49da87d88a5a45cab922",
        "Uni128-Inst0-Group12": "0x267930f5946361aaa37e885a48ce8057bde10cff",
        "Uni128-Inst0-Group13": "0xdd07a0f63a3dff728846d03eb1a856e68ec1ac96",
        "Uni128-Inst0-Group14": "0x6c4b79e49de3711d76e8437b81aeb4cfe39601c0",
        "Uni128-Inst0-Group15": "0x86a881bc06ec6b50cae968e6090109d6b420015b",
        "Uni128-Inst0-Group16": "0x406d0e1e36a93ff9bf48a8c59ea7ebae0c85b08d",
        "Uni128-Inst0-Group17": "0xfe9d2bbaacd094d6bb015ac6d6e8ae334e4db002",
        "Uni128-Inst0-Group18": "0x5fa9c80aed795f2ce0f3adee45fa91246675b260",
        "Uni128-Inst0-Group19": "0xbbded04817d6b68b12face06be24c95e7bfd3c73",
        "Uni128-Inst0-Group20": "0xa68b82ddc9fcff23ab6ed614a5618634cc55d1ca",
        "Uni128-Inst0-Group21": "0xdd9edc44a6b86c358456dcf802059479cd681f9d",
        "Uni128-Inst0-Group22": "0x082de57050b743c47bc18407588aa214b590f943",
        "Uni128-Inst0-Group23": "0xc72e9480e074ab13f65cfbab01dc25d5bf37cf1f",
        "Uni128-Inst0-Group24": "0x410fc16dca2f43682f75417b6553be8e444c603b",
        "Uni128-Inst0-Group25": "0x9a70400c5b1111d5ae6b0628054ce74aade6db5c",
        "Uni128-Inst0-Group27": "0x7f525c4a675b8f4fed86c29beabdce111cc67862",
        "Uni128-Inst0-Group26": "0xba61bd1df7259903149ef527ab2684a1a83422e0",
        "Uni128-Inst0-Group29": "0xd84e1b19889859f524bc66f0c229da6f073722c5",
        "Uni128-Inst1-Group0": "0xe17bd3cbbb84703081895255b8b236a0398f6dd1",
        "Uni128-Inst0-Group28": "0x26ed854c81bd29ee8561ccced0236402dc2556a3",
        "Uni128-Inst1-Group1": "0xd13fed33a1f045b84abfc6dac0c6fe6a38c0b7f6",
        "Uni128-Inst1-Group2": "0xc74d7d1aa727ce06a14a7411dbfdf494eea73913",
        "Uni128-Inst1-Group3": "0x13bf7a6c0118d7876e02f2c4bb53a73908d1aa67",
        "Uni128-Inst1-Group4": "0xf679f574dd0152b549012f9c6b0c3d2774f8ddb8",
        "Uni128-Inst1-Group5": "0x8e0ac3fd05ea5f15e79872e94e5f103a6395e796",
        "Uni128-Inst1-Group6": "0x9aa390c2d99b1a6ee5f2960953a0ff43413820d0",
        "Uni128-Inst1-Group7": "0x683209f17595e0ca9c7187d5cea059a303e127bc",
        "Uni128-Inst1-Group8": "0xb5655b305c3ea686cfb858a2b9e91e753e0a158c",
        "Uni128-Inst1-Group9": "0x06ed06a891f016a59e70dc5e80c6956f07697b56",
        "Uni128-Inst1-Group10": "0x3ffbec35e78edc69cbc0f4f495d9f98d01dab896",
        "Uni128-Inst1-Group11": "0x60311e2536bcd519ff8a2a41096700e234d4ea73",
        "Uni128-Inst1-Group13": "0x2703ceea40eac29ad79752dcd349b89e7a0508e8",
        "Uni128-Inst1-Group12": "0x963afa09720ff83de7f37d671f45bb38a5a8c7cd",
        "Uni128-Inst1-Group14": "0x0373ed67ef4900a8649667d3c7aee23990835f1a",
        "Uni128-Inst1-Group16": "0x11454be244ef25bf1827925c825fbe682e999e13",
        "Uni128-Inst1-Group15": "0xd477ec19000f07dd440de69c6a41db5f967fb91f",
        "Uni128-Inst1-Group17": "0x9e260be7b0e33bd4f785aff653754e93b23ccee6",
        "Uni128-Inst1-Group18": "0x8ea8859836badf7181f74e5457142d6e812050d9",
        "Uni128-Inst1-Group19": "0x1bab39f4ca6df73d1f70cefe5c23dee16505da4e",
        "Uni128-Inst1-Group20": "0xab8731d18a939113dc985ad59fa3422e4b3de91b",
        "Uni128-Inst1-Group21": "0xf9d56c665d4932a9e6eb114de239214bd89db504",
        "Uni128-Inst1-Group22": "0xa662eb5a762e1688acc4f047d90e4b8e34ae15f4",
        "Uni128-Inst1-Group23": "0x70dbac85f49091837c65fd8bc7de291e1556b17c",
        "Uni128-Inst1-Group24": "0x01f3db8abcf093f00330b51dcc30a0e6fc4262fa",
        "Uni128-Inst1-Group25": "0x13f338a5eafcea936458d0bc68607a88c16e53a4",
        "Uni128-Inst1-Group26": "0xf2b4ad25aa8ccbfe4c5bda8f2ace10c57131316e",
        "Uni128-Inst1-Group27": "0x933ca788bc21644d073c7f77199903d6744d044e",
        "Uni128-Inst1-Group28": "0xc63348398f5f9283abd1b635a853313fa106f71b",
        "Uni128-Inst1-Group29": "0xa828f7d62fed20ffad2672bf1a0e9a9aee7d9ca4"
    },
    "red_domain": {
        "Uni256-Inst0": "0x34cec3eca25b6f4d13c964684f2d9c87de37fae1",
        "Uni256-Inst1": "0x4E2002A194073147f7ee1024B31b857BCDDA3Db8",
        "Uni256-Inst0-Group0": "0x01ed11f0224eb67151e6cb3036e30cf317269342",
        "Uni256-Inst0-Group1": "0x0300d2d88dff0e5803284e85beed5a2c84582719",
        "Uni256-Inst0-Group2": "0x9e15460a7b7e7fac9f5b3e55ea0639a6afcb9d69",
        "Uni256-Inst0-Group3": "0x66eb7e1f29e65ab8ebf3ba8187bec352e798a280",
        "Uni256-Inst0-Group5": "0x596fbd88c3617321dddf541f8a6b51ba6836888c",
        "Uni256-Inst0-Group6": "0x6d419eb35e7e7dbbe7b6b1108f8347fbdea15592",
        "Uni256-Inst0-Group7": "0xf24dae763dfa882ce87567e714d382fcc585bae2",
        "Uni256-Inst0-Group8": "0x54f130c49d2e2fb68f518c0b2341f7e6e4feaded",
        "Uni256-Inst0-Group4": "0x4d5053d72f19d7b22b61283edcdd0e643b9998ea",
        "Uni256-Inst0-Group9": "0x960b739d013b504675e2ab327a628444b4602f48",
        "Uni256-Inst0-Group10": "0xff1f0ecc46b3145fcbfc004ba0384a381c1fcd09",
        "Uni256-Inst0-Group11": "0x2f64c6e356771800de92d7edfa1c4ef21bf4db30",
        "Uni256-Inst0-Group12": "0xf1e070e473c01b8e4259d770c12ab1844669439f",
        "Uni256-Inst0-Group13": "0x30c553115ae4bb2cd699e4717c90661262acc04f",
        "Uni256-Inst0-Group14": "0xdf229f4e358869baaedcd158fa1c2fe74fa531d0",
        "Uni256-Inst0-Group15": "0xbe485620d444bfbae9f7569be17db39055817b19",
        "Uni256-Inst0-Group16": "0x2bbbc043322f1d29a8d257a2d28f26723efa9797",
        "Uni256-Inst0-Group17": "0xcf36238c22327c67ee0aab0840dbb242074264b8",
        "Uni256-Inst0-Group18": "0xdbc4140e67ff116e434cc437798595a77fd07264",
        "Uni256-Inst0-Group19": "0x1cd821635b5df9064f15b6508850ea0820a807f4",
        "Uni256-Inst0-Group20": "0xb78b0618572b5cc2bd90bb9c7a33b894d6022ccc",
        "Uni256-Inst0-Group21": "0x05e4625fce32dc28b1b00662035a65dc429b1e68",
        "Uni256-Inst0-Group22": "0x25a08c5193a3de0e7cc135cd44b46d505a5fdbbc",
        "Uni256-Inst0-Group23": "0x137a3bb878798004dc73a945ffa5ebf199f473f9",
        "Uni256-Inst0-Group24": "0x189fc3fac39640fa18e5624a18786a726c7a515a",
        "Uni256-Inst0-Group25": "0x9f7bada170a9a9150a276c670d96eab64ab72fd2",
        "Uni256-Inst0-Group26": "0x078e3b56a203b1875cb695ed951776abf3667b16",
        "Uni256-Inst0-Group27": "0xeb40269fafe526cf41ff7365a676038809eda85d",
        "Uni256-Inst0-Group28": "0xe18d87f8ade2e6e1c64a2a74371a89037ae28b00",
        "Uni256-Inst0-Group30": "0x633b6d4f49eea3d44c686bff6b83e63d91f65856",
        "Uni256-Inst0-Group31": "0xd43efe6c69f7c3fbe905b360a3a3a35963c875ca",
        "Uni256-Inst0-Group29": "0xd77fc789b61e7d33bc4259ab37944ac924b31e64",
        "Uni256-Inst0-Group32": "0x32a66d122d17fd77ac6e7e4052069afa2d5094b6",
        "Uni256-Inst0-Group33": "0xc57383ee38dda7b964875e81f54a0a79234c471f",
        "Uni256-Inst0-Group34": "0xb2e994ebf8dfa765435899586d5423d505ab3e71",
        "Uni256-Inst0-Group35": "0x33e8b1a4a1881d3f83da59b72516df4344b2771a",
        "Uni256-Inst0-Group37": "0x0d870eb48ad25d963674086e9e5d5c7c8bc8eb4d",
        "Uni256-Inst0-Group36": "0x1f7bfc57418bf8685bc7f8de308d9ac6b04ad5e4",
        "Uni256-Inst0-Group38": "0x771e0f1e95eb45fc55199476d77b5945c2b0932b",
"Uni256-Inst0-Group39": "0xe459af55ed5b40f17ab79cd75e448c942ba8fc22",
"Uni256-Inst0-Group40": "0x9b59abe3e4f34348e428223b2455844c6f2531a9",
"Uni256-Inst0-Group42": "0xf35a3cbfc3fbc23c3aff187f6eb5f95b3623e85b",
"Uni256-Inst0-Group43": "0x160eff84290bb657374a707e4c14f1978c8ef924",
"Uni256-Inst0-Group44": "0x62f7ca0f59c97d22de6454f85deda598c7a3b415",
"Uni256-Inst0-Group45": "0x70f8c52909541485f378fc058af9fe498c6ca757",
"Uni256-Inst0-Group41": "0x367a0c29587adb9a8067daf4959325fb956a04eb",
"Uni256-Inst0-Group46": "0xb8852c4ee359102ec10ecbcaf76727b5796d227e",
"Uni256-Inst0-Group47": "0xdd92a4452fe51e0bd2d1ddeb2c80fc9ef90d60a7",
"Uni256-Inst0-Group48": "0xc22bd979d8e01230a48f6bf5c8d2182ca2183449",
"Uni256-Inst0-Group49": "0x9b7cf85ee7d5d488d1f33ca74f7571c286500cd3",
"Uni256-Inst0-Group51": "0xd40218d98974e24720dcb84c0988d4a288324808",
"Uni256-Inst0-Group52": "0x7265e9e196f39f8a3ad31b75dbf2dbc4fd7d4564",
"Uni256-Inst0-Group53": "0x082e0e4764568261a41dbaae46310f8d371d8adc",
"Uni256-Inst0-Group54": "0x1825eb8690c810281e0e435e6b69ceb794b70e3c",
"Uni256-Inst0-Group55": "0x354e51493e4adb69d837bf050cdc5dc1f88e2ef7",
"Uni256-Inst0-Group50": "0xe6afe932bfea01f7e48a87a8230f17327af6fcad",
"Uni256-Inst0-Group56": "0x071868db6313a955e284665c1fe7cca48843af76",
"Uni256-Inst0-Group57": "0xdb351c23fc091e98eef75a186e959c12ad8b6478",
"Uni256-Inst0-Group58": "0x56a0d557822946a3583d4fae54f357e6cdb44197",
"Uni256-Inst0-Group59": "0x10ef26128a08a707b0f3b1b7166d4d3fcac29249",
"Uni256-Inst0-Group60": "0x6df0e59d5eefd7093cb9a2da2889417808773698",
"Uni256-Inst0-Group61": "0xb6a78b82f91a784a18c91e7ef07e3dc672712e4a",
"Uni256-Inst1-Group0": "0xacd6600f0cebccbbf3106eae87542f515c7b9133",
"Uni256-Inst1-Group1": "0x549c80201869aaff1982bb9a2e7fb2274c3174ab",
"Uni256-Inst1-Group2": "0xcc2f8e363a4ff31681a0b0075fc524e73aae14b6",
"Uni256-Inst1-Group3": "0x5ea9a3ced2f1a94f73cdcdf37d49ba1fd576847c",
"Uni256-Inst1-Group4": "0x3884614eff95d621f457906d2004e5ae8545c7f3",
"Uni256-Inst1-Group5": "0x188d2a2516cff3b4f63f418fcd963ab37d3981c8",
"Uni256-Inst1-Group6": "0x585ce873ee5b4466fcb6e2e190b1a55847a87d36",
"Uni256-Inst1-Group7": "0x76a91e8063b886f61af98c339b62347bcb4f2428",
"Uni256-Inst1-Group8": "0xf38ffb54f6b9169560196aac653ca13cf2fcc4b8",
"Uni256-Inst1-Group9": "0x4ed6e5127e2c102a363b4bce8682cc0991fef76a",
"Uni256-Inst1-Group10": "0xaa0e454897cabb67c44b4a7139f9615b2c58b7b1",
"Uni256-Inst1-Group11": "0x63564df7cad0a26afae866d836b34ac5e77dde0d",
"Uni256-Inst1-Group12": "0x1c75c5c6b3cf86577a8d96c82053390bb8a5c3fa",
"Uni256-Inst1-Group13": "0xe4fde86d7cf326f298defcb427fd688e9976d612",
"Uni256-Inst1-Group14": "0xdb899f467de80b77434f23981ceed5f4836bd55f",
"Uni256-Inst1-Group15": "0x647f2809223397490cbb735bf2b866766cc02641",
"Uni256-Inst1-Group16": "0x381665c716e97899e91915c1d3edeae5ffe2b7f7",
"Uni256-Inst1-Group17": "0xfe1102b243d30dd61e4373444f9c768a475150e9",
"Uni256-Inst1-Group18": "0x772b1ee841875be672a63f2c9e971e49e622ddd1",
"Uni256-Inst1-Group19": "0xf24a6f589e03f28a542d9dfccb88d65fae629198",
"Uni256-Inst1-Group20": "0x40b0383381ea9c7afbe148b59e96a51b9792408f",
"Uni256-Inst1-Group21": "0x3722f18b1df20d6975c88bbfeded4d89b4c04afc",
"Uni256-Inst1-Group22": "0x8f2a7e64420b9c6b15c5e7a037e88dfbacc9b3b8",
"Uni256-Inst1-Group23": "0x861cb1df3444db34d880d3d75d5cd2a99888d949",
"Uni256-Inst1-Group24": "0x9fbc46f6020478bdb40687234781391c7bdc6c04",
"Uni256-Inst1-Group25": "0x970886684e9fc6de2da2b17b9dc70c3e12b7e75b",
"Uni256-Inst1-Group26": "0xd7a7e46f62eb31f1c9199bdd5036440e82a769ac",
"Uni256-Inst1-Group27": "0x3248b93e9a628baad3f298b97f94bed77e1c46f1",
"Uni256-Inst1-Group28": "0xfb7c344fb259e400de1dad9973ef40cb314d88b4",
"Uni256-Inst1-Group29": "0xd9e2ad281ea0a3f6f3ed2a7987400035c719b321",
"Uni256-Inst1-Group30": "0x4b49ddaf97a86ce74f5cfbcab8ce1c01b8c9f318",
"Uni256-Inst1-Group31": "0xbad998eaa527fbce4cb79f0eff6eb5f5833f2032",
"Uni256-Inst1-Group32": "0x3d1fe3afe0b87be611f89b097d722a44eabfcbc7",
"Uni256-Inst1-Group33": "0xad21f674f9f4b600116a02bdf2cdd152b13bb378",
"Uni256-Inst1-Group34": "0xa41ce67044e1f4ceb4b5d7d4e03b53ffcd461d4a",
"Uni256-Inst1-Group35": "0xb3648dae389cc3cb2581b606bea4433ea1ab5aff",
"Uni256-Inst1-Group37": "0x352a62a0280fa90adc9a7cfc03bb269d9e90bce8",
"Uni256-Inst1-Group36": "0x3b551af684f2c0162626126b54ce55daf6ec60d9",
"Uni256-Inst1-Group38": "0x5d853391d607139597b1950921ddf0046f49eef3",
"Uni256-Inst1-Group39": "0xae8d32bf9e248bc18d5c339a7eb97a1cf726d9ab",
"Uni256-Inst1-Group40": "0x344ef1c5e75c3431a96804a79cc30c18661a283b",
"Uni256-Inst1-Group41": "0x62ce4cc95a76c1540a2afd81e066fe34b3038036",
"Uni256-Inst1-Group42": "0xf281d5de991bd993046f6583d0bcd272e25f29e6",
"Uni256-Inst1-Group43": "0x4b31d5c541f3c058a282b096e7ddef729edaf9d5",
"Uni256-Inst1-Group44": "0xf950503b3616bb1be9b17232e652bdda431ef0a3",
"Uni256-Inst1-Group45": "0xD3E0d52c33E1d58b5090cdC9be3Df3a67F42C2d9",
"Uni256-Inst1-Group46": "0xef7445143555c89c320ad536e75e8c89c8a51bc8",
"Uni256-Inst1-Group47": "0x6f0ac011b89d16fe59a8246dc43ca385d5e197c4",
"Uni256-Inst1-Group48": "0x7dce6107ee23f2f5535b9e87cc61fe454a8d9287",
"Uni256-Inst1-Group49": "0xfb06490ca2855ef0adb9c89b04c73e066f6ba5bf",
        "Uni256-Inst1-Group50": "0x734e6d82680df7186030df23403f4d100bebc29f",
        "Uni256-Inst1-Group51": "0x03a62e0c246421ef1b6c998bc62885a2478aa72a",
        "Uni256-Inst1-Group52": "0xf999c4c3773a7d2b3f359cda6314f96925d77c91",
        "Uni256-Inst1-Group53": "0xc504adf1e7e27823c7a4259c511ca06215f2b834",
        "Uni256-Inst1-Group54": "0x73616c014343ef8a4677058dbda259cda2647f2b",
        "Uni256-Inst1-Group55": "0x3f41d9c7bc7df615ab9645d3ccf90aeeb93a1bde",
        "Uni256-Inst1-Group56": "0xc1df435825853e66c5e1ab66f25ae32bfb26f9a1",
        "Uni256-Inst1-Group57": "0xe1099153f18baea6a9b8bcc2a2e2f82736a115d2",
        "Uni256-Inst1-Group58": "0x3e9290692dbbf7377b261eb30f275c814ee8b96e",
        "Uni256-Inst1-Group59": "0xc6e060ac7167082b91734e1a493681a4a9d23153",
        "Uni256-Inst1-Group60": "0x4d66a4b14f05c5836b10a4b2ca9115f1aefe1b8e",
        "Uni256-Inst1-Group61": "0x7cffb51e4f6205b9980354b6ae76a6bb079af0d7"
    }
}
```

A.2 Results Scenario 2

Listing A.2: Scenario 2 results for cooperating domains.

```
{
    "request": {
        "url": "http://localhost:3001/contracts/0x55526b860d8fc67bef7440e236c02231acb12d90/search",
        "body": {
            "target": "http://thesis.eval/scenario1",
            "links": [
                "trusted"
            ]
        }
    },
    "results": {
        "count": 1,
        "time": 1350,
        "list": {
            "0xb54cb4313f6fa7bf51d7250209b324efe26b8984": "Proj-Cloud"
        }
    },
    "searched": {
        "count": 8,
        "list": {
            "0x55526b860d8fc67bef7440e236c02231acb12d90": "TU",
            "0x1d1d28c13a0816948dc64fa3184b845cd80aa49e": "InstX",
            "0x93ba312f68a08c9d7be0ac603baf4123f6c99827": "InstX-Group0",
            "0x385134ff0e51505f7f109ce170b8763c76ed845c": "InstX-Group1",
            "0xcf9efa13aa7b5600fe263676bf7d6d19e7bda56e": "InfoSys",
            "0x45374493b34fce298bfcd8f53074dc7798e9ed1d": "InfoSys-Group0",
            "0xb0ab80a55113eeb9c5ee65fb332ed8b57c191196": "DSG",
            "0xb54cb4313f6fa7bf51d7250209b324efe26b8984": "Proj-Cloud"
        }
    }
}
```

A.3 Results Scenario 3

Listing A.3: Scenario 3 results for malicious provenance in TU domain.

```
{
    "request": {
        "url": "http://localhost:3001/contracts/0x55526b860d8fc67bef7440e236c02231acb12d90/search",
        "body": {
            "target": "http://thesis.eval/scenario2",
            "links": [
                "trusted",
                "known"
            ]
        }
    },
    "results": {
        "count": 2,
        "time": 757,
        "list": {
            "0x07c431f898ec376cb51348092eda10a5f4d21159": "Uni16-Inst0-Group0",
            "0xb54cb4313f6fa7bf51d7250209b324efe26b8984": "Proj-Cloud"
        }
    },
    "searched": {
        "count": 15,
        "list": {
            "0x55526b860d8fc67bef7440e236c02231acb12d90": "TU",
            "0x75735b7a532108ea0760ba4b4841d399e16f1fd7": "Uni16",
            "0x36c0322cc017dad799bbfd10968f91b1e52591b4": "Uni16-Inst0",
            "0xef12512047eb269fa69a8aaffe888e7ded12ea5f": "Uni16-Inst0-Group1",
            "0x07c431f898ec376cb51348092eda10a5f4d21159": "Uni16-Inst0-Group0",
            "0xcfcca9d7cf921787874100739be986488ce0ec9c": "Uni16-Inst1",
            "0x936738f2daad1ecbd823ca550429439972c128ad": "Uni16-Inst1-Group1",
            "0x8358f3f2b3a975c3f772b0c5f5abd784ca1fd98f": "Uni16-Inst1-Group0",
            "0x1d1d28c13a0816948dc64fa3184b845cd80aa49e": "InstX",
            "0x93ba312f68a08c9d7be0ac603baf4123f6c99827": "InstX-Group0",
            "0x385134ff0e51505f7f109ce170b8763c76ed845c": "InstX-Group1",
            "0xcf9efa13aa7b5600fe263676bf7d6d19e7bda56e": "InfoSys",
            "0x45374493b34fce298bfcd8f53074dc7798e9ed1d": "InfoSys-Group0",
            "0xb0ab80a55113eeb9c5ee65fb332ed8b57c191196": "DSG",
            "0xb54cb4313f6fa7bf51d7250209b324efe26b8984": "Proj-Cloud"
        }
    }
}
```

A.4 Results Scenario 4

A.4.1 Query 1

Listing A.4: Scenario 4 results for TU domain.

```
{
    "request": {
        "url": "http://localhost:3001/contracts/0x55526b860d8fc67bef7440e236c02231acb12d90/search",
        "body": {
            "target": "http://thesis.eval/scenario3",
            "links": [
                "trusted"
            ]
        }
    },
    "results": {
        "count": 2,
        "time": 904,
        "list": {
            "0xb0ab80a55113eeb9c5ee65fb332ed8b57c191196": "DSG",
            "0x93ba312f68a08c9d7be0ac603baf4123f6c99827": "InstX-Group0"
        }
    },
    "searched": {
        "count": 8,
        "list": {
            "0x55526b860d8fc67bef7440e236c02231acb12d90": "TU",
            "0xcf9efa13aa7b5600fe263676bf7d6d19e7bda56e": "InfoSys",
            "0x45374493b34fce298bfcd8f53074dc7798e9ed1d": "InfoSys-Group0",
            "0xb0ab80a55113eeb9c5ee65fb332ed8b57c191196": "DSG",
            "0xb54cb4313f6fa7bf51d7250209b324efe26b8984": "Proj-Cloud",
            "0x1d1d28c13a0816948dc64fa3184b845cd80aa49e": "InstX",
            "0x93ba312f68a08c9d7be0ac603baf4123f6c99827": "InstX-Group0",
            "0x385134ff0e51505f7f109ce170b8763c76ed845c": "InstX-Group1"
        }
    }
}
```

A.4.2 Query 2

Listing A.5: Scenario 4 results for Uni16 domain without propagating trust from TU domain.

```
{
    "request": {
        "url": "http://localhost:3001/contracts/0x55526b860d8fc67bef7440e236c02231acb12d90/search",
        "body": {
            "target": "http://thesis.eval/scenario4",
            "links": [
                "trusted"
            ]
        }
    },
    "results": {
        "count": 1,
        "time": 672,
        "list": {
            "0xb0ab80a55113eeb9c5ee65fb332ed8b57c191196": "DSG"
        }
    },
    "searched": {
        "count": 8,
        "list": {
            "0x55526b860d8fc67bef7440e236c02231acb12d90": "TU",
            "0xcf9efa13aa7b5600fe263676bf7d6d19e7bda56e": "InfoSys",
            "0xb0ab80a55113eeb9c5ee65fb332ed8b57c191196": "DSG",
            "0xb54cb4313f6fa7bf51d7250209b324efe26b8984": "Proj-Cloud",
            "0x45374493b34fce298bfcd8f53074dc7798e9ed1d": "InfoSys-Group0",
            "0x1d1d28c13a0816948dc64fa3184b845cd80aa49e": "InstX",
            "0x385134ff0e51505f7f109ce170b8763c76ed845c": "InstX-Group1",
            "0x93ba312f68a08c9d7be0ac603baf4123f6c99827": "InstX-Group0"
        }
    }
}
```

A.4.3 Query 3

Listing A.6: Scenario 4 results for Uni16 domain with propagated trust from TU domain.

```
{
    "request": {
        "url": "http://localhost:3001/contracts/0x75735b7a532108ea0760ba4b4841d399e16f1fd7/search",
        "body": {
            "target": "http://thesis.eval/scenario4",
            "links": [
                "trusted"
            ]
        }
    },
    "results": {
        "count": 1,
        "time": 1280,
        "list": {
            "0x07c431f898ec376cb51348092eda10a5f4d21159": "Uni16-Inst0-Group0"
        }
    },
    "searched": {
        "count": 7,
        "list": {
            "0x75735b7a532108ea0760ba4b4841d399e16f1fd7": "Uni16",
            "0x36c0322cc017dad799bbfd10968f91b1e52591b4": "Uni16-Inst0",
            "0xef12512047eb269fa69a8aaffe888e7ded12ea5f": "Uni16-Inst0-Group1",
            "0x07c431f898ec376cb51348092eda10a5f4d21159": "Uni16-Inst0-Group0",
            "0xcfcca9d7cf921787874100739be986488ce0ec9c": "Uni16-Inst1",
            "0x936738f2daad1ecbd823ca550429439972c128ad": "Uni16-Inst1-Group1",
            "0x8358f3f2b3a975c3f772b0c5f5abd784ca1fd98f": "Uni16-Inst1-Group0"
        }
    }
}
```

A.5 Results Scenario 5

A.5.1 Query 1

Listing A.7: Scenario 5 results for TU domain.

```
{
    "request": {
        "url": "http://localhost:3001/contracts/0x75735b7a532108ea0760ba4b4841d399e16f1fd7/search",
        "body": {
            "target": "http://thesis.eval/scenario4",
            "links": [
                "trusted",
                "known"
            ]
        }
    },
    "results": {
        "count": 2,
        "time": 4114,
        "list": {
            "0xb0ab80a55113eeb9c5ee65fb332ed8b57c191196": "DSG",
            "0x07c431f898ec376cb51348092eda10a5f4d21159": "Uni16-Inst0-Group0"
        }
    },
    "searched": {
        "count": 15,
        "list": {
            "0x75735b7a532108ea0760ba4b4841d399e16f1fd7": "Uni16",
            "0xcfcca9d7cf921787874100739be986488ce0ec9c": "Uni16-Inst1",
            "0x936738f2daad1ecbd823ca550429439972c128ad": "Uni16-Inst1-Group1",
            "0x8358f3f2b3a975c3f772b0c5f5abd784ca1fd98f": "Uni16-Inst1-Group0",
            "0x55526b860d8fc67bef7440e236c02231acb12d90": "TU",
            "0xcf9efa13aa7b5600fe263676bf7d6d19e7bda56e": "InfoSys",
            "0x45374493b34fce298bfcd8f53074dc7798e9ed1d": "InfoSys-Group0",
            "0xb0ab80a55113eeb9c5ee65fb332ed8b57c191196": "DSG",
            "0xb54cb4313f6fa7bf51d7250209b324efe26b8984": "Proj-Cloud",
            "0x1d1d28c13a0816948dc64fa3184b845cd80aa49e": "InstX",
            "0x385134ff0e51505f7f109ce170b8763c76ed845c": "InstX-Group1",
            "0x93ba312f68a08c9d7be0ac603baf4123f6c99827": "InstX-Group0",
            "0x36c0322cc017dad799bbfd10968f91b1e52591b4": "Uni16-Inst0",
            "0xef12512047eb269fa69a8aaffe888e7ded12ea5f": "Uni16-Inst0-Group1",
            "0x07c431f898ec376cb51348092eda10a5f4d21159": "Uni16-Inst0-Group0"
        }
    }
}
```

A.5.2 Query 2

Listing A.8: Scenario 5 results for fake domain without propagation of trust by TU main contract.

```
{
    "request": {
        "url": "http://localhost:3001/contracts/0x55526b860d8fc67bef7440e236c02231acb12d90/search",
        "body": {
            "target": "http://thesis.eval/scenario5",
            "links": [
                "trusted"
            ]
        }
    },
    "results": {
        "count": 1,
        "time": 728,
        "list": {
            "0xb0ab80a55113eeb9c5ee65fb332ed8b57c191196": "DSG"
        }
    },
    "searched": {
        "count": 8,
        "list": {
            "0x55526b860d8fc67bef7440e236c02231acb12d90": "TU",
            "0x1d1d28c13a0816948dc64fa3184b845cd80aa49e": "InstX",
            "0x93ba312f68a08c9d7be0ac603baf4123f6c99827": "InstX-Group0",
            "0x385134ff0e51505f7f109ce170b8763c76ed845c": "InstX-Group1",
            "0xcf9efa13aa7b5600fe263676bf7d6d19e7bda56e": "InfoSys",
            "0x45374493b34fce298bfcd8f53074dc7798e9ed1d": "InfoSys-Group0",
            "0xb0ab80a55113eeb9c5ee65fb332ed8b57c191196": "DSG",
            "0xb54cb4313f6fa7bf51d7250209b324efe26b8984": "Proj-Cloud"
        }
    }
}
```

A.5.3 Query 3

Listing A.9: Scenario 5 results when scanning entire network.

```
{
    "request": {
        "url": "http://localhost:3001/contracts/0x78Af41B30Bd48d94965A8fc3fA563FE2950fD638/search",
        "body": {
            "target": "http://thesis.eval/scenario5",
            "links": [
                "trusted"
            ]
        }
    },
    "results": {
        "count": 1,
        "time": 2127,
        "list": {
            "0xdc6ecb9bf091304b5dbd2e57d8e9d082a0bbeb79": "Uni32-Inst0-Group0"
        }
    },
    "searched": {
        "count": 15,
        "list": {
            "0x78af41b30bd48d94965a8fc3fa563fe2950fd638": "Uni32",
            "0x2f53d70e1ba5cd94fa98ed97ccae3bfde8c0b4e4": "Uni32-Inst1",
            "0x31dfd5b73628338b67fd7421a49ad8c45525d87b": "Uni32-Inst1-Group1",
            "0x902dd87cdd4037b5209ad4fd4f69464fbc194d0e": "Uni32-Inst1-Group2",
            "0xf91fca430f60a30bc4f4378caf925da2bef8875f": "Uni32-Inst1-Group3",
            "0xfa2e2390039595fbc1cb7d9ffc51924aeac6fb04": "Uni32-Inst1-Group0",
            "0x6a24294d3279fdb0250fb06068d1869be1a27441": "Uni32-Inst1-Group5",
            "0x9382a3f6083650ebada45292b65dce9762175caf": "Uni32-Inst1-Group4",
            "0xa5631e23a8215b4dedc660052146a8146cd4a55a": "Uni32-Inst0",
            "0x82babd4a986951dee21ca53c41643cbf20a33de7": "Uni32-Inst0-Group4",
            "0xe71bdff910c73a8436704cb697be429ce2f87d3b": "Uni32-Inst0-Group2",
            "0x886634923f31c128cd9084155331a70ea5eec806": "Uni32-Inst0-Group3",
            "0x329d47434a7041885a81e026165dc87b4fccb7a8": "Uni32-Inst0-Group5",
            "0x08604c0b0dce181fe9764d8bccb4324ad6704a23": "Uni32-Inst0-Group1",
            "0xdc6ecb9bf091304b5dbd2e57d8e9d082a0bbeb79": "Uni32-Inst0-Group0"
        }
    }
}
```

A.6 Provenance Network

Listing A.10: Deployed contracts on test network Ropsten.

```
{
    "request": {
        "url": "http://localhost:3001/contracts/0x78Af41B30Bd48d94965A8fc3fA563FE2950fD638/search",
        "body": {
            "target": "http://thesis.eval/scenario5",
            "links": [
                "trusted",
                "linkback"
            ]
        }
    },
    "results": {
        "count": 2,
        "time": 9921,
        "list": {
            "0xb0ab80a55113eeb9c5ee65fb332ed8b57c191196": "DSG",
            "0xdc6ecb9bf091304b5dbd2e57d8e9d082a0bbeb79": "Uni32-Inst0-Group0"
        }
    },
    "searched": {
        "count": 256,
        "list": {
            "0x78af41b30bd48d94965a8fc3fa563fe2950fd638": "Uni32",
            "0x1747bae0546a80818e974eb81e99c7846099dd12": "Search32",
            "0x330c2646ea6be38625ce3b15957738820b31370a": "Search16",
            "0x55526b860d8fc67bef7440e236c02231acb12d90": "TU",
            "0x75735b7a532108ea0760ba4b4841d399e16f1fd7": "Uni16",
            "0x36c0322cc017dad799bbfd10968f91b1e52591b4": "Uni16-Inst0",
            "0x07c431f898ec376cb51348092eda10a5f4d21159": "Uni16-Inst0-Group0",
            "0xef12512047eb269fa69a8aaffe888e7ded12ea5f": "Uni16-Inst0-Group1",
            "0xcfcca9d7cf921787874100739be986488ce0ec9c": "Uni16-Inst1",
            "0x936738f2daad1ecbd823ca550429439972c128ad": "Uni16-Inst1-Group1",
            "0x8358f3f2b3a975c3f772b0c5f5abd784ca1fd98f": "Uni16-Inst1-Group0",
            "0xcf9efa13aa7b5600fe263676bf7d6d19e7bda56e": "InfoSys",
            "0x45374493b34fce298bfcd8f53074dc7798e9ed1d": "InfoSys-Group0",
            "0xb0ab80a55113eeb9c5ee65fb332ed8b57c191196": "DSG",
            "0xb54cb4313f6fa7bf51d7250209b324efe26b8984": "Proj-Cloud",
            "0x1d1d28c13a0816948dc64fa3184b845cd80aa49e": "InstX",
            "0x385134ff0e51505f7f109ce170b8763c76ed845c": "InstX-Group1",
            "0x93ba312f68a08c9d7be0ac603baf4123f6c99827": "InstX-Group0",
            "0x7487d90e1faff5c569ae99b1453adceaabd87c8a": "Search64",
            "0xfd840eca0bdf85a350f0cea0f5b5a9ecd0322793": "Uni64",
            "0x31e91fb905d8fbf32b7491c21640c57a14dc0372": "Uni64-Inst0",
            "0x70a0e88f73f1444d7f0c7aafda53aec9ead04f48": "Uni64-Inst0-Group12",
            "0x35e2043e609490304909d1883babb2a577a8313f": "Uni64-Inst0-Group9",
            "0x3b6cf5ebf16dccfbc6272aad9242e9ed48dc57ae": "Uni64-Inst0-Group1",
            "0xa19c556bae1298026417ca84142ba3fbff4714e3": "Uni64-Inst0-Group13",
            "0x06f2f06ab8ee1489277091c3f5f94920a6926ed1": "Uni64-Inst0-Group8",
            "0xe5f5b23fc365aeaa3ae99d1fecda0a29b5216f68": "Uni64-Inst0-Group6",
            "0x9fc19ea263313dc7373107a5919831b08395151e": "Uni64-Inst0-Group4",
            "0x6aa348c583e47635672c1829098d5eeef10b692b": "Uni64-Inst0-Group0",
            "0x324a039343fa5028e7aabce7abab27dbcf5c8c3c": "Uni64-Inst0-Group11",
            "0x4f01e833272911a2b2ce8e048866169fb736a0ae": "Uni64-Inst0-Group3",
            "0x61f5c4b8614bd892771149da2da10d911295bbb2": "Uni64-Inst0-Group10",
            "0xda4957e502e621630cca0ca883e26d26bc74d716": "Uni64-Inst0-Group7",
            "0x4e4fa3829cb9b08d33a194bfc5eb46c0c542b43a": "Uni64-Inst0-Group2",
            "0x42d32e19aecea1b8c7ba8dd6e4469f5dcbeb5ab7": "Uni64-Inst0-Group5",
            "0x8ce0daf8512027337b4b895533602c658ba07e0f": "Uni64-Inst1",
            "0x8ac22fee88b6c1abb6c360e38b63d3a959f2cb44": "Uni64-Inst1-Group1",
            "0xf0f761ecdcb7faad212a11eb421572774f0f0b5c": "Uni64-Inst1-Group3",
            "0xa14283052c33fe4658eff5f3e2b176daff66bb0f": "Uni64-Inst1-Group0",
            "0x43d6988b0910938c260249f383b2dd22773119cb": "Uni64-Inst1-Group2",
            "0x36699f19d77d77401062982ce7bfe8ee9dc58fcb": "Uni64-Inst1-Group13",
            "0x7be66ed3dc4e0b50aad410583f6e11e23e935894": "Uni64-Inst1-Group10",
            "0xe44a17530dc897f713259dd35ce2d60b0e6f22fe": "Uni64-Inst1-Group9",
            "0x29413f5768f194d5a7d6680cbb44d0ff0d44230d": "Uni64-Inst1-Group6",
            "0x17d825146cbb9eb895b8fb10630e30eb47626307": "Uni64-Inst1-Group12",
            "0x05ee15dacc9c55b2dbafd6b8260a867567ab85ce": "Uni64-Inst1-Group8",
            "0x1f6d920b5ac13b9c9f0ed57b98edda65839bb72a": "Uni64-Inst1-Group7",
"0x7c3ace277b70107c1947aaeee1ff40a1e7393f31": "Uni64-Inst1-Group4",
"0xa9fd8e06619c7a6bc83852ba24ff9b5f28da84ed": "Uni64-Inst1-Group11",
"0x847e07e202cfd278c7eed8cd51eb58df14c0cff0": "Uni64-Inst1-Group5",
"0x82a6f0217348ce9af9ea6d7d54aa41ddda06de3e": "Search128",
"0x9fa8d22001c79d4c0d2607c97541ef46cfc0ee15": "Uni128",
"0x7b7fb11557807db7256c75917e60e9892e95ce2c": "Uni128-Inst0",
"0xdd9edc44a6b86c358456dcf802059479cd681f9d": "Uni128-Inst0-Group21",
"0x495e745fb593f4501c603c2299d15b9e45287081": "Uni128-Inst0-Group5",
"0xa68b82ddc9fcff23ab6ed614a5618634cc55d1ca": "Uni128-Inst0-Group20",
"0x7f525c4a675b8f4fed86c29beabdce111cc67862": "Uni128-Inst0-Group27",
"0xafab4d841d86fd5d029ca2f3b6a41c56002d7ad4": "Uni128-Inst0-Group4",
"0xba61bd1df7259903149ef527ab2684a1a83422e0": "Uni128-Inst0-Group26",
"0x9a70400c5b1111d5ae6b0628054ce74aade6db5c": "Uni128-Inst0-Group25",
"0x86a881bc06ec6b50cae968e6090109d6b420015b": "Uni128-Inst0-Group15",
"0x8eade39e64cb0d1035966a2147de7971aa08571f": "Uni128-Inst0-Group0",
"0x267930f5946361aaa37e885a48ce8057bde10cff": "Uni128-Inst0-Group12",
"0x11e9970b812dae51c40b49da87d88a5a45cab922": "Uni128-Inst0-Group11",
"0x287ce5250cda9fdb22925d97601b5b4d7b6ceecf": "Uni128-Inst0-Group1",
"0xd0aa3e92bdd4b9e1f6de13f4c4f4b7adff5f52af": "Uni128-Inst0-Group6",
"0xc72e9480e074ab13f65cfbab01dc25d5bf37cf1f": "Uni128-Inst0-Group23",
"0xd84e1b19889859f524bc66f0c229da6f073722c5": "Uni128-Inst0-Group29",
"0xbbded04817d6b68b12face06be24c95e7bfd3c73": "Uni128-Inst0-Group19",
"0x082de57050b743c47bc18407588aa214b590f943": "Uni128-Inst0-Group22",
"0x5fa9c80aed795f2ce0f3adee45fa91246675b260": "Uni128-Inst0-Group18",
"0x406d0e1e36a93ff9bf48a8c59ea7ebae0c85b08d": "Uni128-Inst0-Group16",
"0xfe9d2bbaacd094d6bb015ac6d6e8ae334e4db002": "Uni128-Inst0-Group17",
"0xdd59d748a33e8897aad6f7f3212a31993479ec9f": "Uni128-Inst0-Group2",
"0x2c884ccb2a021920bfeabf614fb75416d60b8b23": "Uni128-Inst0-Group9",
"0x56b6cca90d73f479998707658b8dc88533835ab8": "Uni128-Inst0-Group7",
"0x4470351a92d89cc511447c8bf230f7315658bfba": "Uni128-Inst0-Group10",
"0x207d0461c60e8a4c8fb7cc7b7ed1b0623d546f97": "Uni128-Inst0-Group8",
"0x26ed854c81bd29ee8561ccced0236402dc2556a3": "Uni128-Inst0-Group28",
"0x0933f168829653ed4a62c258834073c5d53ac8db": "Uni128-Inst0-Group3",
"0x6c4b79e49de3711d76e8437b81aeb4cfe39601c0": "Uni128-Inst0-Group14",
"0x410fc16dca2f43682f75417b6553be8e444c603b": "Uni128-Inst0-Group24",
"0xdd07a0f63a3dff728846d03eb1a856e68ec1ac96": "Uni128-Inst0-Group13",
"0x0fd380ba625edc8b86ae09d8e8fdf50e823124ea": "Uni128-Inst1",
"0xc63348398f5f9283abd1b635a853313fa106f71b": "Uni128-Inst1-Group28",
"0x11454be244ef25bf1827925c825fbe682e999e13": "Uni128-Inst1-Group16",
"0xab8731d18a939113dc985ad59fa3422e4b3de91b": "Uni128-Inst1-Group20",
"0xa828f7d62fed20ffad2672bf1a0e9a9aee7d9ca4": "Uni128-Inst1-Group29",
"0x60311e2536bcd519ff8a2a41096700e234d4ea73": "Uni128-Inst1-Group11",
"0x683209f17595e0ca9c7187d5cea059a303e127bc": "Uni128-Inst1-Group7",
"0x70dbac85f49091837c65fd8bc7de291e1556b17c": "Uni128-Inst1-Group23",
"0x13bf7a6c0118d7876e02f2c4bb53a73908d1aa67": "Uni128-Inst1-Group3",
"0x9e260be7b0e33bd4f785aff653754e93b23ccee6": "Uni128-Inst1-Group17",
"0xe17bd3cbbb84703081895255b8b236a0398f6dd1": "Uni128-Inst1-Group0",
"0x963afa09720ff83de7f37d671f45bb38a5a8c7cd": "Uni128-Inst1-Group12",
"0xd477ec19000f07dd440de69c6a41db5f967fb91f": "Uni128-Inst1-Group15",
"0x1bab39f4ca6df73d1f70cefe5c23dee16505da4e": "Uni128-Inst1-Group19",
"0x2703ceea40eac29ad79752dcd349b89e7a0508e8": "Uni128-Inst1-Group13",
"0x06ed06a891f016a59e70dc5e80c6956f07697b56": "Uni128-Inst1-Group9",
"0x8ea8859836badf7181f74e5457142d6e812050d9": "Uni128-Inst1-Group18",
"0xf679f574dd0152b549012f9c6b0c3d2774f8ddb8": "Uni128-Inst1-Group4",
"0xd13fed33a1f045b84abfc6dac0c6fe6a38c0b7f6": "Uni128-Inst1-Group1",
"0xa662eb5a762e1688acc4f047d90e4b8e34ae15f4": "Uni128-Inst1-Group22",
"0xf9d56c665d4932a9e6eb114de239214bd89db504": "Uni128-Inst1-Group21",
"0xf2b4ad25aa8ccbfe4c5bda8f2ace10c57131316e": "Uni128-Inst1-Group26",
"0x933ca788bc21644d073c7f77199903d6744d044e": "Uni128-Inst1-Group27",
"0x3ffbec35e78edc69cbc0f4f495d9f98d01dab896": "Uni128-Inst1-Group10",
"0x13f338a5eafcea936458d0bc68607a88c16e53a4": "Uni128-Inst1-Group25",
"0x0373ed67ef4900a8649667d3c7aee23990835f1a": "Uni128-Inst1-Group14",
"0xb5655b305c3ea686cfb858a2b9e91e753e0a158c": "Uni128-Inst1-Group8",
"0x01f3db8abcf093f00330b51dcc30a0e6fc4262fa": "Uni128-Inst1-Group24",
"0x9aa390c2d99b1a6ee5f2960953a0ff43413820d0": "Uni128-Inst1-Group6",
"0xc74d7d1aa727ce06a14a7411dbfdf494eea73913": "Uni128-Inst1-Group2",
"0x8e0ac3fd05ea5f15e79872e94e5f103a6395e796": "Uni128-Inst1-Group5",
"0x239249e1bb5859cb5214555e06dfb08260ee64ae": "Search256",
"0x188b3f7579695501dddb5e155dcae3d187e73cb7": "Uni256",
"0x4e2002a194073147f7ee1024b31b857bcdda3db8": "Uni256-Inst1",
"0x772b1ee841875be672a63f2c9e971e49e622ddd1": "Uni256-Inst1-Group18",
"0x970886684e9fc6de2da2b17b9dc70c3e12b7e75b": "Uni256-Inst1-Group25",
"0xdb899f467de80b77434f23981ceed5f4836bd55f": "Uni256-Inst1-Group14",
"0x1c75c5c6b3cf86577a8d96c82053390bb8a5c3fa": "Uni256-Inst1-Group12",
"0xbad998eaa527fbce4cb79f0eff6eb5f5833f2032": "Uni256-Inst1-Group31",
"0x4b49ddaf97a86ce74f5cfbcab8ce1c01b8c9f318": "Uni256-Inst1-Group30",
"0x381665c716e97899e91915c1d3edeae5ffe2b7f7": "Uni256-Inst1-Group16",
"0x9fbc46f6020478bdb40687234781391c7bdc6c04": "Uni256-Inst1-Group24",
"0xaa0e454897cabb67c44b4a7139f9615b2c58b7b1": "Uni256-Inst1-Group10",
"0xd9e2ad281ea0a3f6f3ed2a7987400035c719b321": "Uni256-Inst1-Group29",
"0xfb7c344fb259e400de1dad9973ef40cb314d88b4": "Uni256-Inst1-Group28",
"0x585ce873ee5b4466fcb6e2e190b1a55847a87d36": "Uni256-Inst1-Group6",
"0x188d2a2516cff3b4f63f418fcd963ab37d3981c8": "Uni256-Inst1-Group5",
"0x3e9290692dbbf7377b261eb30f275c814ee8b96e": "Uni256-Inst1-Group58",
"0xc504adf1e7e27823c7a4259c511ca06215f2b834": "Uni256-Inst1-Group53",
"0x3d1fe3afe0b87be611f89b097d722a44eabfcbc7": "Uni256-Inst1-Group32",
"0xe1099153f18baea6a9b8bcc2a2e2f82736a115d2": "Uni256-Inst1-Group57",
"0xc1df435825853e66c5e1ab66f25ae32bfb26f9a1": "Uni256-Inst1-Group56",
"0xf999c4c3773a7d2b3f359cda6314f96925d77c91": "Uni256-Inst1-Group52",
"0xcc2f8e363a4ff31681a0b0075fc524e73aae14b6": "Uni256-Inst1-Group2",
"0x03a62e0c246421ef1b6c998bc62885a2478aa72a": "Uni256-Inst1-Group51",
"0x4b31d5c541f3c058a282b096e7ddef729edaf9d5": "Uni256-Inst1-Group43",
"0x7dce6107ee23f2f5535b9e87cc61fe454a8d9287": "Uni256-Inst1-Group48",
"0x734e6d82680df7186030df23403f4d100bebc29f": "Uni256-Inst1-Group50",
"0x344ef1c5e75c3431a96804a79cc30c18661a283b": "Uni256-Inst1-Group40",
"0x3b551af684f2c0162626126b54ce55daf6ec60d9": "Uni256-Inst1-Group36",
"0xacd6600f0cebccbbf3106eae87542f515c7b9133": "Uni256-Inst1-Group0",
"0x352a62a0280fa90adc9a7cfc03bb269d9e90bce8": "Uni256-Inst1-Group37",
"0xd3e0d52c33e1d58b5090cdc9be3df3a67f42c2d9": "Uni256-Inst1-Group45",
"0xb3648dae389cc3cb2581b606bea4433ea1ab5aff": "Uni256-Inst1-Group35",
"0xd7a7e46f62eb31f1c9199bdd5036440e82a769ac": "Uni256-Inst1-Group26",
"0xf24a6f589e03f28a542d9dfccb88d65fae629198": "Uni256-Inst1-Group19",
"0xe4fde86d7cf326f298defcb427fd688e9976d612": "Uni256-Inst1-Group13",
"0xa41ce67044e1f4ceb4b5d7d4e03b53ffcd461d4a": "Uni256-Inst1-Group34",
"0x63564df7cad0a26afae866d836b34ac5e77dde0d": "Uni256-Inst1-Group11",
"0xfe1102b243d30dd61e4373444f9c768a475150e9": "Uni256-Inst1-Group17",
"0x861cb1df3444db34d880d3d75d5cd2a99888d949": "Uni256-Inst1-Group23",
"0x549c80201869aaff1982bb9a2e7fb2274c3174ab": "Uni256-Inst1-Group1",
"0x8f2a7e64420b9c6b15c5e7a037e88dfbacc9b3b8": "Uni256-Inst1-Group22",
"0x3248b93e9a628baad3f298b97f94bed77e1c46f1": "Uni256-Inst1-Group27",
"0x76a91e8063b886f61af98c339b62347bcb4f2428": "Uni256-Inst1-Group7",
"0x7cffb51e4f6205b9980354b6ae76a6bb079af0d7": "Uni256-Inst1-Group61",
"0xc6e060ac7167082b91734e1a493681a4a9d23153": "Uni256-Inst1-Group59",
"0x73616c014343ef8a4677058dbda259cda2647f2b": "Uni256-Inst1-Group54",
"0x3f41d9c7bc7df615ab9645d3ccf90aeeb93a1bde": "Uni256-Inst1-Group55",
"0xfb06490ca2855ef0adb9c89b04c73e066f6ba5bf": "Uni256-Inst1-Group49",
"0x40b0383381ea9c7afbe148b59e96a51b9792408f": "Uni256-Inst1-Group20",
"0xf950503b3616bb1be9b17232e652bdda431ef0a3": "Uni256-Inst1-Group44",
"0x3884614eff95d621f457906d2004e5ae8545c7f3": "Uni256-Inst1-Group4",
"0x62ce4cc95a76c1540a2afd81e066fe34b3038036": "Uni256-Inst1-Group41",
"0xae8d32bf9e248bc18d5c339a7eb97a1cf726d9ab": "Uni256-Inst1-Group39",
"0xf281d5de991bd993046f6583d0bcd272e25f29e6": "Uni256-Inst1-Group42",
"0xad21f674f9f4b600116a02bdf2cdd152b13bb378": "Uni256-Inst1-Group33",
"0x647f2809223397490cbb735bf2b866766cc02641": "Uni256-Inst1-Group15",
"0x4ed6e5127e2c102a363b4bce8682cc0991fef76a": "Uni256-Inst1-Group9",
"0x3722f18b1df20d6975c88bbfeded4d89b4c04afc": "Uni256-Inst1-Group21",
"0xf38ffb54f6b9169560196aac653ca13cf2fcc4b8": "Uni256-Inst1-Group8",
"0x5ea9a3ced2f1a94f73cdcdf37d49ba1fd576847c": "Uni256-Inst1-Group3",
"0x6f0ac011b89d16fe59a8246dc43ca385d5e197c4": "Uni256-Inst1-Group47",
"0x5d853391d607139597b1950921ddf0046f49eef3": "Uni256-Inst1-Group38",
"0xef7445143555c89c320ad536e75e8c89c8a51bc8": "Uni256-Inst1-Group46",
"0x4d66a4b14f05c5836b10a4b2ca9115f1aefe1b8e": "Uni256-Inst1-Group60",
"0x34cec3eca25b6f4d13c964684f2d9c87de37fae1": "Uni256-Inst0",
"0x960b739d013b504675e2ab327a628444b4602f48": "Uni256-Inst0-Group9",
"0x6d419eb35e7e7dbbe7b6b1108f8347fbdea15592": "Uni256-Inst0-Group6",
"0x56a0d557822946a3583d4fae54f357e6cdb44197": "Uni256-Inst0-Group58",
"0x596fbd88c3617321dddf541f8a6b51ba6836888c": "Uni256-Inst0-Group5",
"0x62f7ca0f59c97d22de6454f85deda598c7a3b415": "Uni256-Inst0-Group44",
"0xdb351c23fc091e98eef75a186e959c12ad8b6478": "Uni256-Inst0-Group57",
"0x160eff84290bb657374a707e4c14f1978c8ef924": "Uni256-Inst0-Group43",
"0x367a0c29587adb9a8067daf4959325fb956a04eb": "Uni256-Inst0-Group41",
"0x33e8b1a4a1881d3f83da59b72516df4344b2771a": "Uni256-Inst0-Group35",
"0xb6a78b82f91a784a18c91e7ef07e3dc672712e4a": "Uni256-Inst0-Group61",
"0xb2e994ebf8dfa765435899586d5423d505ab3e71": "Uni256-Inst0-Group34",
"0xdd92a4452fe51e0bd2d1ddeb2c80fc9ef90d60a7": "Uni256-Inst0-Group47",
"0xc57383ee38dda7b964875e81f54a0a79234c471f": "Uni256-Inst0-Group33",
            "0x6df0e59d5eefd7093cb9a2da2889417808773698": "Uni256-Inst0-Group60",
            "0xe459af55ed5b40f17ab79cd75e448c942ba8fc22": "Uni256-Inst0-Group39",
            "0x10ef26128a08a707b0f3b1b7166d4d3fcac29249": "Uni256-Inst0-Group59",
            "0xb8852c4ee359102ec10ecbcaf76727b5796d227e": "Uni256-Inst0-Group46",
            "0x633b6d4f49eea3d44c686bff6b83e63d91f65856": "Uni256-Inst0-Group30",
            "0xd77fc789b61e7d33bc4259ab37944ac924b31e64": "Uni256-Inst0-Group29",
            "0x078e3b56a203b1875cb695ed951776abf3667b16": "Uni256-Inst0-Group26",
            "0xd43efe6c69f7c3fbe905b360a3a3a35963c875ca": "Uni256-Inst0-Group31",
            "0xcf36238c22327c67ee0aab0840dbb242074264b8": "Uni256-Inst0-Group17",
            "0x9f7bada170a9a9150a276c670d96eab64ab72fd2": "Uni256-Inst0-Group25",
            "0xd40218d98974e24720dcb84c0988d4a288324808": "Uni256-Inst0-Group51",
            "0x137a3bb878798004dc73a945ffa5ebf199f473f9": "Uni256-Inst0-Group23",
            "0x30c553115ae4bb2cd699e4717c90661262acc04f": "Uni256-Inst0-Group13",
            "0x0300d2d88dff0e5803284e85beed5a2c84582719": "Uni256-Inst0-Group1",
            "0xe6afe932bfea01f7e48a87a8230f17327af6fcad": "Uni256-Inst0-Group50",
            "0x4d5053d72f19d7b22b61283edcdd0e643b9998ea": "Uni256-Inst0-Group4",
            "0x54f130c49d2e2fb68f518c0b2341f7e6e4feaded": "Uni256-Inst0-Group8",
            "0x66eb7e1f29e65ab8ebf3ba8187bec352e798a280": "Uni256-Inst0-Group3",
            "0xf24dae763dfa882ce87567e714d382fcc585bae2": "Uni256-Inst0-Group7",
            "0x9b7cf85ee7d5d488d1f33ca74f7571c286500cd3": "Uni256-Inst0-Group49",
            "0x082e0e4764568261a41dbaae46310f8d371d8adc": "Uni256-Inst0-Group53",
            "0x189fc3fac39640fa18e5624a18786a726c7a515a": "Uni256-Inst0-Group24",
            "0x071868db6313a955e284665c1fe7cca48843af76": "Uni256-Inst0-Group56",
            "0x70f8c52909541485f378fc058af9fe498c6ca757": "Uni256-Inst0-Group45",
            "0x9e15460a7b7e7fac9f5b3e55ea0639a6afcb9d69": "Uni256-Inst0-Group2",
            "0x354e51493e4adb69d837bf050cdc5dc1f88e2ef7": "Uni256-Inst0-Group55",
            "0x01ed11f0224eb67151e6cb3036e30cf317269342": "Uni256-Inst0-Group0",
            "0x1825eb8690c810281e0e435e6b69ceb794b70e3c": "Uni256-Inst0-Group54",
            "0xc22bd979d8e01230a48f6bf5c8d2182ca2183449": "Uni256-Inst0-Group48",
            "0x32a66d122d17fd77ac6e7e4052069afa2d5094b6": "Uni256-Inst0-Group32",
            "0x25a08c5193a3de0e7cc135cd44b46d505a5fdbbc": "Uni256-Inst0-Group22",
            "0xe18d87f8ade2e6e1c64a2a74371a89037ae28b00": "Uni256-Inst0-Group28",
            "0xb78b0618572b5cc2bd90bb9c7a33b894d6022ccc": "Uni256-Inst0-Group20",
            "0xff1f0ecc46b3145fcbfc004ba0384a381c1fcd09": "Uni256-Inst0-Group10",
            "0xbe485620d444bfbae9f7569be17db39055817b19": "Uni256-Inst0-Group15",
            "0xdf229f4e358869baaedcd158fa1c2fe74fa531d0": "Uni256-Inst0-Group14",
            "0xf1e070e473c01b8e4259d770c12ab1844669439f": "Uni256-Inst0-Group12",
            "0x2f64c6e356771800de92d7edfa1c4ef21bf4db30": "Uni256-Inst0-Group11",
            "0x771e0f1e95eb45fc55199476d77b5945c2b0932b": "Uni256-Inst0-Group38",
            "0x0d870eb48ad25d963674086e9e5d5c7c8bc8eb4d": "Uni256-Inst0-Group37",
            "0xf35a3cbfc3fbc23c3aff187f6eb5f95b3623e85b": "Uni256-Inst0-Group42",
            "0x1f7bfc57418bf8685bc7f8de308d9ac6b04ad5e4": "Uni256-Inst0-Group36",
            "0x05e4625fce32dc28b1b00662035a65dc429b1e68": "Uni256-Inst0-Group21",
            "0x9b59abe3e4f34348e428223b2455844c6f2531a9": "Uni256-Inst0-Group40",
            "0xeb40269fafe526cf41ff7365a676038809eda85d": "Uni256-Inst0-Group27",
            "0xdbc4140e67ff116e434cc437798595a77fd07264": "Uni256-Inst0-Group18",
            "0x7265e9e196f39f8a3ad31b75dbf2dbc4fd7d4564": "Uni256-Inst0-Group52",
            "0x1cd821635b5df9064f15b6508850ea0820a807f4": "Uni256-Inst0-Group19",
            "0x2bbbc043322f1d29a8d257a2d28f26723efa9797": "Uni256-Inst0-Group16",
            "0x2f53d70e1ba5cd94fa98ed97ccae3bfde8c0b4e4": "Uni32-Inst1",
            "0xfa2e2390039595fbc1cb7d9ffc51924aeac6fb04": "Uni32-Inst1-Group0",
            "0x31dfd5b73628338b67fd7421a49ad8c45525d87b": "Uni32-Inst1-Group1",
            "0x6a24294d3279fdb0250fb06068d1869be1a27441": "Uni32-Inst1-Group5",
            "0x902dd87cdd4037b5209ad4fd4f69464fbc194d0e": "Uni32-Inst1-Group2",
            "0x9382a3f6083650ebada45292b65dce9762175caf": "Uni32-Inst1-Group4",
            "0xf91fca430f60a30bc4f4378caf925da2bef8875f": "Uni32-Inst1-Group3",
            "0xa5631e23a8215b4dedc660052146a8146cd4a55a": "Uni32-Inst0",
            "0xdc6ecb9bf091304b5dbd2e57d8e9d082a0bbeb79": "Uni32-Inst0-Group0",
            "0x82babd4a986951dee21ca53c41643cbf20a33de7": "Uni32-Inst0-Group4",
            "0x329d47434a7041885a81e026165dc87b4fccb7a8": "Uni32-Inst0-Group5",
            "0xe71bdff910c73a8436704cb697be429ce2f87d3b": "Uni32-Inst0-Group2",
            "0x08604c0b0dce181fe9764d8bccb4324ad6704a23": "Uni32-Inst0-Group1",
            "0x886634923f31c128cd9084155331a70ea5eec806": "Uni32-Inst0-Group3"
        }
    }
}
```

List of Figures

Bibliography

[1] M. Szomszor and L. Moreau, “Recording and Reasoning over Data Provenance in Web and Grid Services”, in *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE*, vol. 2888, 2003, pp. 603–620. DOI: 10.1007/978-3-540-39964-3_39.

[2] L. Moreau, V. Tan, L. Varga, P. Groth, S. Miles, J. Vazquez-Salceda, J. Ibbotson, S. Jiang, S. Munroe, O. Rana, and A. Schreiber, “The Provenance of Electronic Data”, *Communications of the ACM*, vol. 51, no. 4, pp. 52–58, 2008. DOI: 10.1145/1330311.1330323.

[3] R. Lu, X. Lin, X. Liang, and X. S. Shen, “Secure Provenance: The Essential of Bread and Butter of Data Forensics in Cloud Computing”, in *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, 2010, pp. 282–292. DOI: 10.1145/1755688.1755723.

[4] M. R. Asghar, M. Ion, G. Russello, and B. Crispo, “Securing Data Provenance in the Cloud”, in *Open problems in network security*, vol. 7039, 2012, pp. 145–160. DOI: 10.1007/978-3-642-27585-2_12.

[5] F. Tschorsch and B. Scheuermann, “Bitcoin and Beyond: A Technical Survey on Decentralized Digital Currencies”, *IEEE Communications Surveys & Tutorials*, vol. 18, no. 3, pp. 2084–2123, 2016. DOI: 10.1109/comst.2016.2535718.

[6] Y. L. Simmhan, B. Plale, and D. Gannon, “A Survey of Data Provenance in e-Science”, *ACM SIGMOD Record*, vol. 34, no. 3, pp. 31–36, 2005. DOI: 10.1145/1084805.1084812.

[7] P. Buneman, S. Khanna, and W.-C. Tan, “Data Provenance: Some Basic Issues”, in *FST TCS 2000: Foundations of Software Technology and Theoretical Computer Science*, vol. 1974, 2000, pp. 87–93. DOI: 10.1007/3-540-44450-5_6.

[8] P. Groth, S. Jiang, S. Miles, S. Munroe, V. Tan, S. Tsasakou, and L. Moreau, “An Architecture for Provenance Systems”, EU Provenance project (IST 511085), Project Report, 2006. [Online]. Available: https://eprints.soton.ac.uk/id/eprint/262023 (last access on 02/26/2019).

[9] P. Groth and L. Moreau, *PROV-Overview: An Overview of the PROV Family of Documents*, W3C Working Group Note, 2013. [Online]. Available: https://www.w3.org/TR/2013/NOTE-prov-overview-20130430/ (last access on 02/27/2019).

[10] G. Klyne, P. Groth, L. Moreau, O. Hartig, Y. Simmhan, J. Myers, T. Lebo, K. Belhajjame, S. Miles, and S. Soiland-Reyes, *PROV-AQ: Provenance Access and Query*, W3C Working Group Note, 2013. [Online]. Available: https://www.w3.org/TR/2013/NOTE-prov-aq-20130430/ (last access on 02/27/2019).

[11] S. Nakamoto, *Bitcoin: A Peer-to-Peer Electronic Cash System*, 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf (last access on 12/10/2018).

[12] X. Liang, S. Shetty, D. Tosh, C. Kamhoua, K. Kwiat, and L. 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)*, 2017. DOI: 10.1109/ccgrid.2017.8.

[13] R. Neisse, G. Steri, and I. Nai-Fovino, "A Blockchain-based Approach for Data Accountability and Provenance Tracking", in *Proceedings of the 12th International Conference on Availability, Reliability and Security (ARES '17)*, 2017. DOI: 10.1145/3098954.3098958.

[14] J. Eberhardt and S. Tai, "On or Off the Blockchain? Insights on Off-Chaining Computation and Data", in *European Conference on Service-Oriented and Cloud Computing*, vol. 10465, 2017, pp. 3–15. DOI: 10.1007/978-3-319-67262-5_1.

[15] H. Collins, *Definition of 'provenance'*, COBUILD Advanced English Dictionary, 2018. [Online]. Available: https://www.collinsdictionary.com/dictionary/english/provenance (last access on 12/17/2018).

[16] L. Moreau, P. Missier, K. Belhajjame, R. B'Far, J. Cheney, S. Coppens, S. Cresswell, Y. Gil, P. Groth, G. Klyne, T. Lebo, J. McCusker, S. Miles, J. Myers, S. Sahoo, and C. Tilmes, *PROV-DM: The PROV Data Model*, W3C Recommendation, 2013. [Online]. Available: https://www.w3.org/TR/2013/REC-prov-dm-20130430/ (last access on 02/27/2019).

[17] L. Moreau, P. Missier, J. Cheney, and S. Soiland-Reyes, *PROV-N: The Provenance Notation*, W3C Recommendation, 2013. [Online]. Available: https://www.w3.org/TR/2013/REC-prov-n-20130430/ (last access on 02/27/2019).

[18] T. Lebo and L. Moreau, *PROV Graph Layout Conventions*, 2012. [Online]. Available: https://www.w3.org/2011/prov/wiki/Diagrams (last access on 12/06/2018).

[19] L. Moreau, P. Groth, J. Cheney, T. Lebo, and S. Miles, "The Rationale of PROV", *Journal of Web Semantics*, vol. 35, no. 4, pp. 235–257, 2015. DOI: 10.1016/j.websem.2015.04.001.

[20] B. Lee, A. Awad, and M. Awad, “Towards Secure Provenance in the Cloud: A Survey”, in *2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC)*, 2015, pp. 577–582. DOI: 10.1109/UCC.2015.102.

[21] A. Narayanan, J. Bonneau, E. Felten, A. Miller, and S. Goldfeder, *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction.* 2016, ISBN: 0691171696.

[22] R. C. Merkle, “A Digital Signature Based on a Conventional Encryption Function”, in *Advances in Cryptology (CRYPTO ’87)*, vol. 293, 1988, pp. 369–378. DOI: 10.1007/3-540-48184-2_32.

[23] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography.* 1996, 810 pp., ISBN: 0849385237.

[24] N. Szabo, “Formalizing and Securing Relationships on Public Networks”, *First Monday*, vol. 2, no. 9, 1997. DOI: 10.5210/fm.v2i9.548.

[25] E. F. Community, *Ethereum Wiki: Design Rational*, 2018. [Online]. Available: https://github.com/ethereum/wiki/wiki/Design-Rationale (last access on 12/10/2018).

[26] S. King and S. Nadal, *PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake*, 2012. [Online]. Available: https://peercoin.net/whitepapers/peercoin-paper.pdf (last access on 02/27/2019).

[27] G. Wood, *Ethereum: A Secure Decentralised Generalised Transaction Ledger*, 2014. [Online]. Available: https://ethereum.github.io/yellowpaper/paper.pdf (last access on 05/22/2018).

[28] A. M. Turing, “On Computable Numbers, with an Application to the Entscheidungsproblem. A Correction”, *Proceedings of the London Mathematical Society*, vol. s2-43, no. 1, pp. 544–546, 1938. DOI: 10.1112/plms/s2-43.6.544.

[29] P. Victor, C. Cornelis, M. D. Cock, and M. de Cock, *Trust Networks for Recommender Systems.* 2011, ISBN: 978-94-91216-07-7. DOI: 10.2991/978-94-91216-08-4.

[30] E. Dumbill, *XML Watch: Finding Friends with XML and RDF*, 2002. [Online]. Available: http://www.foaf-project.org/ (last access on 02/27/2019).

[31] J. Golbeck, B. Parsia, and J. Hendler, “Trust Networks on the Semantic Web”, in *Cooperative Information Agents VII*, vol. 2782, 2003, pp. 238–249. DOI: 10.1007/978-3-540-45217-1_18.

[32] C.-N. Ziegler and G. Lausen, “Propagation Models for Trust and Distrust in Social Networks”, *Information Systems Frontiers*, vol. 7, no. 4-5, pp. 337–358, 2005. DOI: 10.1007/s10796-005-4807-3.

[33] R. Guha, R. Kumar, P. Raghavan, and A. Tomkins, “Propagation of Trust and Distrust”, in *Proceedings of the 13th Conference on World Wide Web (WWW ’04)*, 2004, pp. 403–412. DOI: 10.1145/988672.988727.

[34] S. Bowers, T. McPhillips, M. Wu, and B. Ludäscher, "Project Histories: Managing Data Provenance Across Collection-oriented Scientific Workflow Runs", in *International Conference on Data Integration in the Life Sciences*, vol. 4544, 2007, pp. 122–138. DOI: 10.1007/978-3-540-73255-6_12.

[35] R. Hasan, R. Sion, and M. Winslett, "The Case of the Fake Picasso: Preventing History Forgery with Secure Provenance", in *Proccedings of the 7th Conference on File and Storage Technologies*, 2009, pp. 1–14. [Online]. Available: https://www.usenix.org/legacy/event/fast09/tech/full_papers/hasan/hasan.pdf.

[36] R. Hasan, R. Sion, and M. Winslett, "Preventing History Forgery with Secure Provenance", *ACM Transactions on Storage*, vol. 5, no. 4, pp. 1–43, 2009. DOI: 10.1145/1629080.1629082.

[37] V. Tan, P. Groth, S. Miles, S. Jiang, S. Munroe, S. Tsasakou, and L. Moreau, "Security Issues in a SOA-based Provenance System", in *International Provenance and Annotation Workshop*, vol. 4145, 2006, pp. 203–211. DOI: 10.1007/11890850_21.

[38] I. Y. Jung and H. Y. Yeom, "Provenance Security Guarantee from Origin up to Now in the e-Science Environment", *Journal of Systems Architecture*, vol. 57, no. 4, pp. 425–440, 2011. DOI: 10.1016/j.sysarc.2010.04.006.

[39] A. Ramachandran and D. M. Kantarcioglu, *Using Blockchain and Smart Contracts for Secure Data Provenance Management*, 2017. arXiv: http://arxiv.org/abs/1709.10000v1 [cs.CR]. (last access on 02/27/2019).

[40] M. Stoffers, "Trustworthy Provenance Recording using a Blockchain-like Database", Master's thesis, Leipzig University, 2017. [Online]. Available: https://elib.dlr.de/111772/ (last access on 02/26/2019).

[41] T. McConaghy, R. Marques, A. Müller, D. De Jonghe, T. McConaghy, G. McMullen, R. Henderson, S. Bellemare, and A. Granzotto, *BigchainDB: A Scalable Blockchain Database*, 2016. [Online]. Available: https://mycourses.aalto.fi/pluginfile.php/378362/mod_resource/content/1/bigchaindb-whitepaper.pdf (last access on 02/27/2019).

[42] R. Neisse, G. Steri, I. N. Fovino, and G. Baldini, "SecKit: A Model-based Security Toolkit for the Internet of Things", *Computers & Security*, vol. 54, pp. 60–76, 2015. DOI: 10.1016/j.cose.2015.06.002.

[43] H. M. Kim and M. Laskowski, "Toward an Ontology-driven Blockchain Design for Supply-chain Provenance", *Intelligent Systems in Accounting, Finance and Management*, vol. 25, no. 1, pp. 18–27, 2018. DOI: 10.1002/isaf.1424.

[44] U. Javaid, M. N. Aman, and B. Sikdar, "BlockPro: Blockchain based Data Provenance and Integrity for Secure IoT Environments", in *Proceedings of the 1st Workshop on Blockchain-enabled Networked Sensor Systems (BlockSys'18)*, 2018, pp. 13–18. DOI: 10.1145/3282278.3282281.

[45] M. Massi, A. Miladi, A. Margheri, V. Sassone, and J. Rosenzweig, *Using PROV and Blockchain to Achieve Health Data Provenance*, 2018. [Online]. Available: `https://eprints.soton.ac.uk/421292/` (last access on 02/27/2019).

[46] E. Androulaki, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolić, A. Barger, S. W. Cocco, J. Yellick, V. Bortnikov, C. Cachin, K. Christidis, A. D. Caro, D. Enyeart, C. Ferris, and G. Laventman, "Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains", in *Proceedings of the 13th EuroSys Conference on (EuroSys '18)*, 2018. DOI: `10.1145/3190508.3190538`.

[47] V. Buterin, *On Public and Private Blockchains*, 2015. [Online]. Available: `https://blog.ethereum.org/2015/08/07/on-public-and-private-blockchains/` (last access on 02/27/2019).

[48] K. Janowicz, B. Regalia, P. Hitzler, G. Mai, S. Delbecque, M. Fröhlich, P. Martinent, and T. Lazarus, "On the Prospects of Blockchain and Distributed Ledger Technologies for Open Science and Academic Publishing", *Semantic Web*, vol. 9, no. 5, pp. 545–555, 2018. DOI: `10.3233/SW-180322`.

[49] X. Xu, I. Weber, M. Staples, L. Zhu, J. Bosch, L. Bass, C. Pautasso, and P. Rimba, "A Taxonomy of Blockchain-based Systems for Architecture Design", in *IEEE International Conference on Software Architecture (ICSA)*, 2017, pp. 243–252. DOI: `10.1109/ICSA.2017.33`.

[50] ligi, *EIP 831: URI Format for Ethereum*, 2018. [Online]. Available: `https://eips.ethereum.org/EIPS/eip-831` (last access on 02/27/2019).

[51] D. A. Nagy, *EIP 681: URL Format for Transaction Requests*, 2017. [Online]. Available: `https://eips.ethereum.org/EIPS/eip-681` (last access on 02/27/2019).

[52] E. F. Community, *Ethereum Wiki: Decentralized Apps (DApps)*, 2019. [Online]. Available: `https://github.com/ethereum/wiki/wiki/Decentralized-apps-(dapps)` (last access on 02/27/2019).

[53] M. A. Jadhav, B. R. Sawant, and A. Deshmukh, "Single Page Application using AngularJS", *International Journal of Computer Science and Information Technologies*, vol. 6, no. 3, pp. 2876–2879, 2015. [Online]. Available: `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.736.4771&rep=rep1&type=pdf`.