UNIVERSITY OF SOUTHAMPTON
Faculty of Physical and Applied Sciences
School of Electronics and Computer Science

Secure Provenance-based Auditing of Personal Data Use

by Rocío Aldeco-Pérez

Supervisors: Prof. Luc Moreau
Examiners: Prof. Vladimiro Sassone,
Dr. Mike Kirton

A thesis submitted in partial fulfillment for the degree of Doctor of Philosophy

May 2012

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF PHYSICAL AND APPLIED SCIENCES
SCHOOL OF ELECTRONICS AND COMPUTER SCIENCE

Doctor of Philosophy

by Rocío Aldeco-Pérez

In recent years, an increasing number of personalised services that require users to disclose personal information have appeared on the Web (e.g. social networks, governmental sites, on-line selling sites). By disclosing their personal information, users are given access to a wide range of new functionality and benefits. However, there exists a risk that their personal information is misused.

To strike a balance between the advantages of personal information disclosure and protection of information, governments have created legal frameworks, such as the Data Protection Act, Health Insurance Portability & Accountability Act (HIPAA) or Safe Harbor, which place restrictions on how organisations can process personal information. By auditing the way in which organisations used personal data, it is possible to determine whether they process personal information in accordance with the appropriate frameworks.

The traditional way of auditing collects evidence in a manual way. This evidence is later analysed to assess the degree of compliance to a predefined legal framework. These manual assessments are long, since large amounts of data need to be analysed, and they are unreliable, since there is no guarantee that all data is correctly analysed. As several cases of data leaks and exposures of private data have proven, traditional audits are also prone to intentional and unintentional errors derived from human intervention.

Therefore, this thesis proposes a provenance-based approach to auditing the use of personal information by securely gathering and analysing electronic evidence related to the processing of personal information. This approach makes three contributions to the state of art.

The first contribution is the Provenance-based Auditing Architecture that defines a set of communication protocols to make existing systems provenance-aware. These protocols specify which provenance information should be gathered to verify the compliance with the Data Protection Act. Moreover, we derive a set of Auditing Requirements by analysing a Data Protection Act case study and demonstrate that provenance can be used as electronic evidence of past processing.

The second contribution is the Compliance Framework, which is a provenance-based auditing framework for automatically auditing the compliance with the Data Protection Act's principles. This framework consist of a provenance graph representation (Processing View), a novel graph-based rule representation expressing processing rules (Usage Rules Definition) and a novel set of algorithms that automatically verify whether information was processed according to the Auditing Requirements by comparing the Processing View against the Usage Rules Definition.

The third contribution is the Secure Provenance-based Auditing Architecture that ensures any malicious alteration on provenance during the entire provenance life cycle of recording, storage, querying and analysis can be detected. This architecture, which relies on cryptographic techniques, guarantees the correctness of the audit results.

Contents

List of Figures

List of Tables

List of Algorithms

Declaration of Authorship

I, *Rocío Aldeco-Pérez*, declare that the thesis entitled *Secure Provenance-based Auditing of Personal Data Use* and the work presented in this thesis are both my own, and have been generated by me as the result of my own original research. I confirm that:

- this work was done wholly while in candidature for a research degree at this University;
- where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
- where I have consulted the published work of others, this is always clearly attributed;
- where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
- I have acknowledged all main sources of help;
- where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
- parts of this work have been published in a number of conference and journal papers (see Section 1.3 for a list).

Date: 28 May 2012

Acknowledgements

First, I want to thank the organisations that have sponsored me during my PhD studies, without whose support my research would have been impossible. Thanks to CONACyT for scholarship number 182546, to Programme Alβan, the European Union Programme of High Level Scholarships for Latin America, for scholarship number E06D103956MX and to the University of Southampton for scholarship number 21360103 and all the travel support I have received over the last four years. All three of them have not only provided me with economical support but also with the opportunity to meet a large community of people committed to the development of science.

I want to thank my parents. I have reached this point in my professional life because of the education, values and love that you have given me. You taught me to be committed and passionate about my work. Thanks for all the support that you have given me and especially for taking care of a piece of my heart, Tatty, for one long year.

Omar, thanks for letting me follow this dream and live it with me. For being next to me during stressful times, for leaving your comfortable life, for changing your job, your friends and your language for me. Thanks for being so adaptive and for understanding that this experience made us different but in essence we will be always the same.

Natalia, my daughter, my Tatty, thanks for being so brave and mature to understand that even though we could not spend a lot of time together my love for you is always present. Your smile, your kisses, your hugs are the best reward after hard work. I LOVE YOU!

Special thanks to *Professor Luc Moreau*. Thanks for teaching me how to do research, how to properly write, how to start a research career. I really appreciate the time that you spent reading my work, discussing research directions, and pointing me in the right direction.

To the complete *MexSoc* society, the current and former members, without you guys this time would have been very difficult. The parties, the travels, the movies, the festivals, the food! All of this made me never forget about who I am and where I am from.

I want to especially thank *Aida*. Your friendship has been an enormous support in difficult times. I can discuss any topic with you, I can complain about everything with you and your answers always lift my spirit. Thanks for all those long chats, for being my house mate (I know is not easy), and for just being there. You are and always will be a very special friend. Finally, thanks for helping me to print this document.

Lulu, thanks for the lunch talks, for your very contagious happiness and optimism. Your way of enjoying life is unique and you share that with me every day at 12pm. Finally, thanks for helping me to print this document.

A big "thank you" to *Allan*, for being there every day at 12pm to listen to this bunch of crazy PhD women students. Your funny comments, your mature comments, your great jokes (including pictures) were something that I will never forget.

Zinovi, my lovely house mate, thanks for the long chats, the funny conversations, the tea, the nice chocolate, the wonderful wine, the poker nights, the movie nights, the dinners together, for being an excellent friend.

Ruben, you have been a very special person in this process. Your unconditional support in every aspect of my life, your unique way of seeing life and problems helped me to achieve this moment. I have a very long list of thanks for you but there is not enough space, so I will try to sum up. Thanks for being my house mate, for the Dutch classes, the emotional support, the dinners, the coffee, the wine, the chocolate, the "stroopwafels", for everything. Specially, thanks for the writing support. Thanks for every single moment. You are and always will be a very special person in my life.

Finally, thanks to the IAM group, especially to the "agents people". The chats with espresso coffee were a great support in rough times. I already miss you guys.

Para la luz de mi vida, mi eterna motivación, mi todo: ***Tatty****.*

Nomenclature

$\mathcal{P}$	Universal Set of Purposes
P	Set of Purposes
p	A Purpose
$\mathcal{T}$	Universal Set of Tasks
T	Set of Tasks
t	A Task
$\mathcal{Types}$	Set containing all Data Types
$type$	A Type
$\mathcal{Instances}$	Multiset containing all Data Instances
$instance$	An Instance
$\mathcal{Classes}$	Set containing all Data Classes
$class$	A Class
C_Q	Set of Requested Data Classes
C_U	Set of Used Data Classes
$C_\mathcal{G}$	Set of Generated Data Classes
D_A^p	Set of Collected Data Instances for a Purpose
d_A	A Collected Data Instance
R	Set of Generated Data Instances
r	A Generated Instance
D_U	Set of Used Data Instances
d_U	A Used Data Instance
$\mathcal{Rel}$	Universal Set of Relationship Labels
Rel	Set of Relationships Labels
l	A Relationship Label
G_Γ	Usage Rules Definition
$Rule$	Processing Rule
V	Vertex
E	Edge
G_W	Processing View Graph
B_1	Used Data Compliance Subgraph
B_2	Purposes Validation Subgraph
B_3	Reusing Data Subgraph

C	Relevant Information Subgraph
F	Anonymity Preservation Subgraph
G	Security Subgraph
D	Set of Data
S_A	Signature created by A
$kgen$	Key generation function

Chapter 1

Introduction

Recent years have seen an exponential growth of not only storage and computing power, but also of the speed at which data is transmitted. For the first time in history, these technological advances have made possible the personalisation of services on a large scale[1]. These services are tailored to the specific needs of the recipient, and are capable of creating value where homogeneous services fall short.

For instance, in business, companies can collect information about customers' shopping and travelling habits, as well as their opinions and preferences about particular products. Particularly, the business sector has benefited from the collection of this information to improve and expand existing services, as well as to create new ones.

Similarly, for governments, the benefits of personalised services are also evident: they increase effectiveness, reduce cost and increase satisfaction. For example, governments increasingly offer on-line services that simplify the way in which citizens pay their taxes, apply for pensions and benefits, track their children's educational progress and check their medical records [58, 59]. Moreover, the availability of detailed medical information can aid national health institutions to detect an epidemic and take necessary measures in a timely fashion. In terms of medical research, the availability of these records enables researchers to identify trends in disease growth, and may potentially speed up the development of cures.

In both the private and public sector, the development of these new personalised services highly depends on the availability of detailed personal information. The continual improvements to information processing techniques make the use and modification of this personal information happen at an increasingly faster pace. The benefits of these technological advances are evident and the opportunities are innumerable.

However, these advances also come with great risks. While the proper use of personal information can lead to many benefits, its misuse can result in great harm. For example,

[1] For example, Amazon recommendation service or iGoogle customised Google page

most people are willing to share their medical records with their doctor. However, they are likely to be less content if it turns out their family doctor also uses these records to determine hiring policies. Similarly, on a smaller scale and perhaps more familiar, the misuse of email addresses can lead to the receipt of unwanted email.

To prevent misuse, it is important to ensure that this information does not fall into the wrong hands, and that data is used according to the purposes for which it was disclosed. Two solutions have been proposed in an attempt to address these issues.

The first solution addresses the first part of the problem, i.e. allowing access to information only to whom it was disclosed. This led to the development of access control and information retrieval techniques that protect sensitive information. However, there are two potential difficulties associated with this solution. First, while access control is a reliable method of restricting access to information, it does not specify who should receive those access rights, nor does it ensure the proper use of information to authorised users. Second, as Weitzner *et al.* [146] have pointed out, access control alone cannot properly solve the problem of correct use of personal information on the Web, where information is widely and publicly available from different sources. In some cases, personal information that was initially restricted can be inferred by aggregating different pieces of publicly available information [133]. With new technologies, such as Linked Data [26], it has become easier to aggregate, automatically correlate and infer data across multiple sources of information. By so doing, sensitive data can be disclosed that was properly protected by access control.

The second solution was offered by governments, which decided to legislate the way in which organisations are allowed to manage personal information. For example, the UK saw the approval of the Data Protection Act (DPA) [7], which is the implementation of an EU directive [5] that protects personal information in both the private and public sectors. Similarly, the US government created the US Safe Harbor [8], which protects information transferred for commercial purposes between the US and the EU, and the Health Insurance Portability & Accountability Act (HIPAA) [6] to protect information managed by health institutions. The main problem with this solution is one of enforcement: how can one make sure that the multitude of decentralised and open information systems used by a multitude of people are processing data in compliance with these legislative frameworks.

In light of this problem, both the DPA and the US Safe Harbor define procedures for investigating and resolving complaints from individuals who suspect that their personal data is being misused. Audits are currently the main method to implement these procedures. In detail, an *audit* is an evaluation of an organization or system to ascertain whether personal data is stored and processed according to the principles of the applicable legislation. For example, in this context, audits focus on whether personal data was processed according to the purpose for which it was collected. An audit is performed

by a qualified third party known as *auditor*, who issues a report on the results of the audit. This process is long, as it relies heavily on manual collection and analysis of large amounts of data from different sources, and needs to follow many rules and regulations. Thus, there is no guarantee that all data is properly analysed, and as such, the final conclusions are based on samples rather than the whole set of available data. Moreover, because this is an inherently manual process, these audits are prone to intentional and unintentional errors and omissions. The consequences of the ineffectiveness (of the threat) of these audits are perhaps best illustrated by the highly publicised cases of information misuse (data leaks and exposures of personal data) [116, 109, 108] and the review of different legislation [12, 13].

Thus, both access control and legislation fall short of solving the problem of protecting personal information. As a result, users who give out their personal information have no guarantees that it is used properly, nor do they know how this information is used and by whom—information processing remains non-transparent.

1.1 Provenance: Making Information Processing Transparent

It is this lack of transparency that is the main weakness of legislation, or rather systems implementing this legislation. Without knowing how information was processed, it is impossible to determine whether this processing was in compliance with the law, and to take appropriate measures in case of non-compliance.

To address this shortcoming, auditors need the ability to determine which processes were involved in managing personal information, to analyse these processes, and to make organisations accept responsibility for these processes. This ability is called *information accountability*. Information accountability gives users insight into the way their information was used. Moreover, it allows users and society in general to decide—by means of *automatic audits*—whether information is used appropriately, i.e. in compliance with the corresponding data protection framework.

In order to perform automatic audits, it is necessary to make information processing transparent. This means that the way in which personal information is created, modified and stored should be made explicit. By doing this, an equivalent audit process is carried out to the evidence collection performed by auditors when carrying out manual audits. However, in contrast to manual audits, automatic audits require evidence in electronic machine-readable form.

In computational systems, this electronic evidence that documents the events performed by applications at execution time is also referred to as *provenance*. In more detail, provenance consists of the causal dependencies between data and events explaining what

contributed to a result in a specific state [64]. As such, it not only represents data derivations but also processes derivations, i.e. which processes were involved in the creation of a piece of data. For that reason, it is regarded as a very promising means of providing information accountability [146] in open and distributed systems (such as the Internet). Thus, if the provenance of data is available, processing becomes transparent since the provenance of data can be audited to verify that information was used properly.

In order to achieve this, systems need to record provenance about processing of personal information, store this provenance in a so-called Provenance Store, make it available by offering query functionality, and allow auditors (a role that might be fulfilled by a component in the system itself) to analyse it. Due to the open and distributed nature of these systems, these steps involve communication between multiple components over potentially unsecured networks. Consequently, without additional means of protection, there are multiple ways in which provenance might be altered, tampered with, or fall into the hands of untrusted parties. More specifically, several attacks can be performed in an attempt to break one of the four basic security properties: confidentiality, integrity, authentication and non-repudiation. For example, a confidentiality attack might aim to get unauthorised possession of the provenance of personal information; an integrity attack involves the alteration of provenance in an attempt to cover up misuse of personal information; an authentication attack aims to record provenance in the name of others; and a non-repudiation attack aims to deny the use of personal information, by attempting to refute the creation of the corresponding provenance. If any of these attacks are successful, they can undermine the reliability and trustworthiness of provenance, making it unsuitable for use in audits.

As in any information system, provenance information must be available when needed, a property referred to as *availability*. However, unlike the aforementioned properties, ensuring the availability of provenance does not require provenance specific techniques; existing techniques to protect systems from attacks that target their availability can be readily implemented in systems that store provenance. Therefore, we consider the availability property beyond the scope of this thesis.

1.2 Securing Provenance

Against this background, we identify the need for *securing* provenance. This is one of the central problems we address in this thesis. As mentioned above, a crucial issue in the development of transparent systems is the security of the collected provenance. If this provenance has not been collected and stored in a secure way, results can be altered by malicious parties, and the integrity of the results obtained from audits cannot be ensured.

To address this problem, in this thesis we develop a novel secure architecture that protects not only provenance but also the related application data during its collection, storage, communication, query and analysis phases. By considering the entire life cycle of provenance and application data, this architecture is designed to minimise the risk of undetected alterations of provenance by (malicious) actors (for example, in order to hide improper alterations to data), which would result in incorrect or corrupt audits.

In addition, using the provenance captured by this secure architecture, we develop a novel set of algorithms that automatically verify the compliance of data processing against a set of *processing rules* that specify which operations can be performed on personal data. In this thesis, we derive these rules from the Data Protection Act. Now, to perform automatic verification, the algorithms need both provenance and the processing rules in machine-readable format. To do this, the former is intuitively represented as a *Provenance Graph*, which captures the causal dependencies between data and processes. The latter is also represented as a graph, called a Usage Rules Definition graph, which captures the relations between the data classes that can be used, and the allowable operations that can be performed on that data.

1.3 Thesis Statement and Contributions

Thus, our solution to the problem of auditing processing of personal information in IT systems can be summarised as follows:

By securing provenance we ensure the reliability of audits and enable auditors to verify and analyse whether personal information was processed in compliance with a set of processing rules.

In more detail, the contributions made in this thesis are as follows:

1. Our first contribution is the Provenance-based Auditing Architecture, which consists of a set of protocols designed to augment existing systems to make them provenance-aware, i.e. enabling them to capture provenance at execution time. Moreover, we derive a set of requirements from a case study of the Data Protection Act and demonstrate that these requirements can be verified using provenance. Based on this case study, we identify which provenance should be gathered by the architecture to verify these requirements. With our proposed architecture and the corresponding protocols, the necessary provenance can be collected and visually analysed, demonstrating that provenance can be effectively used to support information transparency.

2. Our second contribution is the Compliance Framework which automatically verifies the requirements derived from the DPA case study. It uses a novel graph-based representation for analysing collected provenance information and verifying whether information was processed according to the identified requirements. Within this representation, provenance is encoded as a so-called *Provenance Graph*, which captures the relations between data and the processes that acted upon it. Similarly, the rules that indicate how personal information should be processed are represented by the *Usage Rules Definition Graph*, a graph that simplifies the analysis of past processing. Finally, compliance is decided by checking whether the Provenance Graph matches a Usage Rules Definition Graph.

3. Our third contribution is the Secure Provenance-based Auditing Architecture that secures the Provenance-based Auditing Architecture in order to minimise the risk of undetected alteration of provenance (for example, by malicious actors) during the stages of creation, storage, querying and analysis. This architecture, which relies on cryptographic techniques, ensures the trustworthiness of the evidence on which audits are based. By securing provenance, we guarantee that the entire system (i.e. provenance and application data) exhibits the properties of confidentiality, integrity, authentication and non-repudiation. These security characteristics are formalised and verified using an automatic model checker.

These contributions have led to the publication of the following papers:

- Aldeco-Pérez, R. and Moreau, L. (2010). Securing Provenance-based Audits. In D. L. McGuinness., J. R. Michaelis and L. Moreau (Eds.), International Provenance and Annotation Workshop (IPAW 2010) (LNCS 6378, pp. 148-164). Troy, NY: Springer-Verlag.

- Aldeco-Pérez, R. and Moreau, L. (2010). A Provenance-based Compliance Framework. In A. J. Berre, A. Gómez-Pérez, K. Tutschku and D. Fensel (Eds.), Future Internet Symposium (FIS 2010) (LNCS 6369, pp. 128-137). Berlin, Germany: Springer-Verlag.

- Aldeco-Pérez, R. and Moreau, L. (2009). Information Accountability supported by a Provenance-based Compliance Framework (Poster). UK e-Science All Hands Meeting 2009. Oxford, UK.

- Aldeco-Pérez, R. and Moreau, L. (2008). Provenance-based Auditing of Private Data Use. International Academic Research Conference, Visions of Computer Science (BSC 2008) (pp. 141-152). London, UK: BCS.

1.4 Thesis Structure

The remainder of this thesis is organised as follows.

Chapter 2 presents an overview of related work. First, we overview the state of the art in provenance, analyse the strengths and weaknesses of existing approaches, and identify the techniques we adopt in our work. Then, we discuss existing approaches for auditing IT systems, and argue that they are unsuitable for open and distributed systems. After that, we discuss the four security characteristics of confidentiality, integrity, authentication and non-repudiation and give an overview of existing techniques that ensure these characteristics hold. Finally, we focus on legislation frameworks for data protection, and explain the necessary conditions for compliance.

In Chapter 3, we study the Data Protection Act and derive a set of Auditing Requirements that specify the conditions for data processing to be in compliance. With this goal in mind, we also present a practical scenario and identify the protocols that involve the exchange of personal information.

In Chapter 4, we develop the Provenance-based Auditing Architecture. This architecture is based on a provenance-aware version of the protocols presented in Chapter 3, augmented with additional components to support automatic audits. Then, we identify which provenance needs to be captured in order to automatically verify the Auditing Requirements stated in Chapter 3.

In Chapter 5, we develop the Compliance Framework, which is capable of automatically verifying the Auditing Requirements described in Chapter 3, using the provenance captured by the architecture from Chapter 4. This framework consists of three components. The first component is a graph based representation of the provenance captured in Chapter 4. The second component is a graph based representation of the processing rules, that state how personal information should be processed. These processing rules are derived from the Auditing Requirements. The third component is a set of algorithms for automatically verifying whether data processing was in compliance with the processing rules.

Chapter 6 discusses the steps that need to be taken to make the Provenance-based Auditing Architecture secure. Here, we explain how to guarantee confidentiality, integrity, authenticity and non-repudiation in each of the stages of the provenance life cycle: recording, storage, querying and analysis. This is first achieved by securing the provenance graphs, and second, by formalising the architecture protocols using UML. Later, the protocols are verified with an automatic model checker. The result of this chapter is the Secure Provenance-based Auditing Architecture.

In Chapter 7, we present a methodical analysis of the security of systems designed according to the Secure Provenance-based Auditing Architecture. This analysis is carried out by using *Attack Trees*, which examine the security of a system under realistic conditions by analysing threats and their corresponding countermeasures.

Finally, Chapter 8 summarises the contribution of this thesis, outlines future work and offers some concluding remarks.

Chapter 2

Related Work

In Chapter 1, we discussed the benefits and risks involved in disclosing personal information. While the proper use of personal information has created value in the form of new services, its misuse can lead to harm. In an attempt to address this problem, governments have proposed legislation to restrict the ways in which personal information can be used. However, legislation merely transforms the original problem into a problem of enforcement. We argued that *information accountability* — making organisations responsible for proper management of personal information — is an essential step towards successful enforcement. To do this, it is necessary to make information processing transparent, making it possible to conduct audits to verify whether this information processing was in compliance with the applicable laws. This transparency can be achieved by documenting the operations that were applied to data, which constitutes the *provenance* of that data. Finally, we argued that, in order to ensure the trustworthiness and reliability of audits, the provenance itself on which these audits are based needs to be trustworthy and reliable. Thus, we identified the problem of *securing* provenance.

Before addressing this problem and developing solutions to it, we first give an overview of the background of this thesis. To do this, we discuss the four areas identified in the central problem summarised above: provenance, electronic audits, security and data protection legislation. In this chapter, we study each of these topics in more detail, discuss the state of the art, and identify the techniques that we adopt in our own work.

Thus, this chapter is organised as follows. Section 2.1 presents an overview of provenance approaches. Here, we discuss the different ways in which provenance is captured, stored, queried and analysed. Section 2.2 presents the main approach used to audit IT systems, audit trails. Section 2.3 formalises the concept of security by discussing the four main properties that need to hold in secure systems. Section 2.4 presents the most important data protection legislations, and explains their structure, concepts and principles. Finally, Section 2.5 summarises the discussion in this chapter.

2.1 Provenance

The word *provenance* is used in diverse areas, such art, archaeology and palaeontology, for describing the history of custody of an object since its creation, including any successive changes made to it. To establish that this object has not been altered and it is not a forgery or a reproduction, it is necessary to have documented evidence of such events. The main purpose of this evidence is to prove that a specific object actually comes from where it is thought to originate from, in other words, that it is authentic.

Similarly, in Information Technology, provenance makes it possible to determine the origin of a computational result and its event history. Electronic provenance can be used to evaluate information quality [112, 47], ascertain information attribution [77], support reproducibility [87, 137], or as in this thesis, as audit evidence to establish information accountability. Due to its extended use in diverse domains, different techniques for capturing, storing and analysing provenance have been developed. These techniques vary according to the specific requirements of each domain, where provenance is defined and represented in different forms.

In order to present a systematic general overview of the most representative provenance approaches, we explain the techniques and approaches used throughout its life cycle to capture, store, query and analyse provenance. This analysis is based on the characterisations presented by Simmhan *et al.* [128], Freire *et al.* [53] and Moreau [103] in their corresponding surveys.

The aim of this analysis is to identify the necessary characteristics of provenance for supporting audits.

2.1.1 Computer Systems Provenance Definition

In computer systems, provenance is defined in several different ways depending on the domain where it is used. A generic definition is given by Simmhan *et al.* [128] who state that: "data provenance is information that helps determine the derivation history of a data product, starting from its original sources". This definition is similar to the one found in most dictionaries in the sense that provenance helps to determine how a data product originated. Here, a data product can be any type of data, such as files, tables, collections, etc. However, more specialised definitions exist, which we briefly discuss below.

Buneman *et al.* [34] define data provenance in database systems as "where a piece of data came from and the process by which it arrived in the database". In that sense, they make a distinction between *why-provenance*, which refers to the source data (tuples) that influence the existence of a query result, and *where-provenance*, which refers to the location (databases) from which the data was copied.

In the scientific domain, Greenwood *et al.* [60] define provenance as "a kind of metadata, recording the process of experiments for e-Science, the purpose and results of experiments as well as annotations and notes about experiments by scientists". This definition states that a type of data (metadata) is captured from a specific area (e-Science). This metadata can be enriched by standard "annotations" indicating when an object was created or updated, who owns it and what its format is. These annotations can be created automatically, but also manually by users.

In a different context, Groth *et al.* [64] define provenance as "[...] the process that led to that piece of data". Using this definition, they argue that provenance not only represents data derivations but also processes derivations, i.e. which processes were involved in the creation of a piece of data. In the context of our work, this definition is attractive, because it includes the processes that were performed on data. This is vital information for establishing the process transparency needed for verifying that personal information was used in accordance with the law. For this reason, this definition by Groth *et al.* is the one used in this thesis.

Now that we have presented the different definitions of provenance, we proceed by reviewing the different existing computational approaches for managing provenance. This review is organised according to the provenance life cycle, which is presented in Figure 2.1.

The cycle starts when provenance is captured and recorded in a storage component called *Provenance Store*, in a stage we refer to as *recording*. The techniques used to record provenance are discussed in Section 2.1.2. Provenance is maintained in the Provenance Store during the *storage* stage. The different approaches used to store provenance are described in Section 2.1.3. Provenance is made available by querying the Provenance Store, during the *query* stage, which is presented in Section 2.1.4. Finally, the results of these queries are analysed, during the *analysis* stage. The different approaches used during this stage are studied in Section 2.1.5.

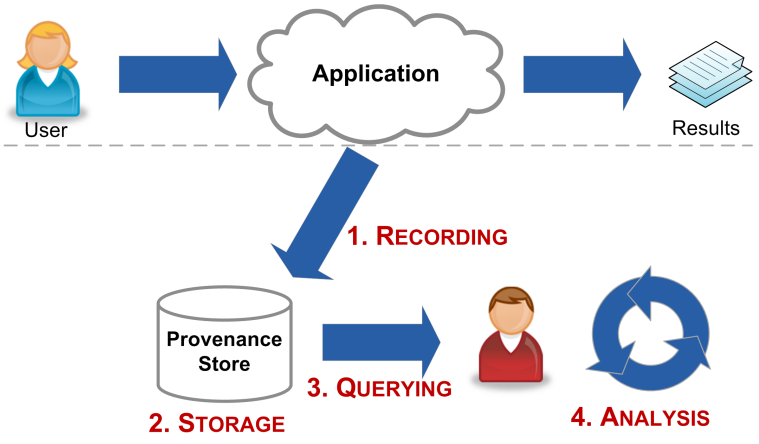


FIGURE 2.1: Provenance Life Cycle

2.1.2 Recording Provenance

As Freire *et al.* argues, there are three different mechanisms used to capture provenance: workflow-based, process-based and Operating System-based [53]. The characteristics, advantages and disadvantages of these mechanisms are discussed below.

2.1.2.1 Workflow-based Systems

Workflow-based Systems systematically capture provenance of complex processes, and represent this provenance as the workflow of data processes that were performed. Processes are specified by well-defined languages allowing the system to support automation, reproducibility and result sharing. One advantage of these systems is that the capture functionality is integrated into the workflow system, and as such, provenance is captured through system APIs. These systems normally provide simple programming models and visual programming interfaces that allow visualisation of provenance workflows as graphs, which can be explored and queried. Examples of this type of system are Kepler [29], Taverna [149] and REDUX [23].

2.1.2.2 Process-based Systems

In process-based systems, each service involved in a computational task is adapted to capture its own provenance. This is achieved by wrapping existing services in a provenance-aware adapter, which is capable of recording and storing the data and data-process dependencies that constitute provenance. To obtain the complete provenance of a piece of data, provenance information is queried by reconstructing the dependencies that exist between services. Similar to workflow-based systems, provenance information can be visualised as graphs.

An example of this type of system is PASOA (Provenance Aware Service Oriented Architecture) [64], which captures extra information that describes what actually occurred at execution time. Such extra information, which is referred to as *process documentation* (which we call provenance), can be seen as documented evidence of the events performed by an application in execution time.

In order for applications to implement PASOA, they first need to be made *Provenance Aware*, by recording process documentation at execution time, in addition to performing the task for which they were designed.

The provenance collected by PASOA consists of a set of assertions generated at execution time, called *p-assertions* (Provenance Assertions), which are asserted by the components of an application. There are three types of p-assertions that document different actions:

Interaction P-Assertion is a description of the contents of a message by an actor that has sent or received such a message.

Relationship P-Assertion is a description of how an actor obtained output data sent in an interaction, by applying some function or algorithm to input data from other interactions.

Actor State P-Assertions is a description provided by an actor about its internal state in the context of a specific interaction.

2.1.2.3 Operating System-based Systems

Operating System-based systems capture provenance at the Operating System (OS) level, and as such, are not aware of the specific processes executed within the system. These systems capture data and data-process dependencies created by system calls and file access. They rely on OS functionalities only, so there is no need for modifying existing applications that are executed on the system. Similar to process-based systems, they obtain provenance information through provenance queries that reconstruct the existing dependencies between system calls and files accesses. One disadvantage of these systems is the low level and quantity of captured provenance, which makes querying and analysing this provenance very challenging. Examples of this type of system are PASS [70] and ES3 [54].

Another important issue about capturing provenance is the granularity level in which it is captured. In the next section we describe the different approaches used.

2.1.2.4 Granularity of Provenance

The granularity of the captured provenance depends on the specific domain and application in which provenance is used. For example, in the databases context, provenance is used to track tables [144], rows (Trio [147], Perm [56]) and cells [33]. In OS-based Systems provenance is used to track provenance of files, such as PASS [70] and ES3 [54], the same as some workflow-based systems such as VDS [150]. Other workflow-based systems, such as Kepler [29] and Taverna [149], track provenance of collections and collections' elements. The main disadvantage of workflow-based systems is that the granularity of provenance is not decided according to a predefined methodology. For example, it does not take into account the analysis that should later be applied to the provenance.

In order to obtain a more flexible method, Miles *et al.* [100] propose a different vision. They argue that the granularity of provenance can be chosen depending on the specific use of this information. Specifically, it is possible to decide which provenance is

recorded based on the questions that we want to answer using provenance information. With that purpose, they introduce the *Provenance Incorporating Methodology* (PrIMe), which is a software engineering methodology for adapting existing applications to record provenance about their execution. PrIMe is divided into three phases: provenance question capture and analysis, actor based decomposition and adapting the application. As this methodology is used in this thesis (see Chapter 4), more information about it is presented below:

Phase 1: Provenance Question Capture and Analysis In the first phase of PrIMe, an analysis of the application identifies the provenance related questions to be answered about the application. Likewise, the information for which provenance is sought is characterised as well as the answer's scope.

Phase 2: Actor Based Decomposition In the second phase of PrIMe, the application is conceptually decomposed into a set of actors, which record process documentation. Then, their interactions are analysed to find the relationships between these interactions. This analysis exposes the information flow within the application. For this purpose, PrIMe uses a graph-based representation of application interactions. The next step is to determine which application actors are involved in the provenance of a given data item: these actors are called knowledgeable actors. If the necessary information for answering the provenance related questions is available from the current actors, then it is time to apply Phase 3. If not, Phase 2 has to be repeated until the correct level of granularity to answer provenance questions is reached.

Phase 3: Adapting the Application This phase involves the modification of the application in order to make explicit the required information items that are currently unavailable, thereby giving the application the necessary functionality to record process documentation, which in turn allows actors to perform queries on the documentation in order to answer provenance questions.

Now that we have discussed techniques for recording provenance, we next focus on the second stage of the provenance life cycle: storage.

2.1.3 Storing Provenance

There are three principal storage strategies: the no-coupling, the tight-coupling and the loose-coupling recording strategy [57].

In the no-coupling strategy provenance is stored in a central provenance repository or in distributed provenance repositories that are dedicated to store provenance only. The main advantage of this strategy is that it can be used in heterogeneous environments with

limited control over the execution (open systems). However, the existence of a centralised provenance repository creates a single point of failure. If, instead, the repository is distributed, retrieving provenance becomes more challenging as the provenance of a single item can be distributed in various repositories.

Since, in this strategy, provenance is stored in a different repository from the one in which application data is stored, there exist two ways in which the relation between application data and its provenance is established. The first one maintains a copy of the original application data as part of the provenance. The second one maintains a reference to this data. This reference is used to retrieve the original data from the database where it is stored. Note that, to obtain the original data, the corresponding credentials are required. Examples of systems that implement this strategy are PASOA, Kepler and Taverna.

In the tight-coupling strategy, provenance is directly associated with the data whose provenance is captured. Thus, provenance is attached to the original data. The main advantage of this strategy is that we do not need a query functionality, because provenance is obtained together with the data. The disadvantage of this is the lack of protection of provenance. When a piece of data is accessed it is possible to access and manipulate its provenance. An example of a system where this strategy is used is discussed by Hasan *et al.* [68].

Finally, in the loose-coupling strategy, provenance and data are stored in a single storage component but logically separated. The main advantages of this strategy is that, since provenance is contained in the same repository, accessing provenance is faster. Moreover, provenance and application data can be protected by different policies, since both are separated. However, should the database crash, not only the application data but also its provenance is lost. Examples of systems that implement the loose-coupling strategy include those described by [34, 32] and the OS-based systems, such as PASS.

2.1.4 Querying Provenance

The fundamental goal of provenance systems is to enable analysis of the results produced by their systems. Therefore, provenance needs to be queried after it has been captured and stored in a provenance store. To this end, the provenance store should be able to respond to *provenance queries*. A provenance query [105] is a user-tailored query over captured provenance aimed to obtain the provenance of a specific data item.

The intent of a provenance query is to select a set of provenance assertions (p-assertions), which we refer to as *provenance query result*, that provides the provenance of a specific piece of data. A p-assertion expresses the relation between a piece of data and a process that used or produced it. The piece of data for which provenance is sought is referred

to as the *data item*. Since not all p-assertions related to the data item are relevant to a query result, only those that belong to a predefined context called *scope* are returned.

In most systems, such as Taverna, PASS, PASOA, and Kepler, provenance query results are represented by Directed Acyclic Graphs (DAGs). These DAGs represents the way in which a piece of data was derived by showing the dependencies between data and processes. The nodes of such DAGs represent data items and edges represent data derivations. Thus, by following the relationships in this DAG, it is possible to ascertain how a data item was produced.

In an effort to standardise this widely used provenance representation, Moreau *et al.* [104] developed the Open Provenance Model (OPM). OPM is a model designed to support the sharing of provenance between different systems. Different from the aforementioned DAGs, in this model, the nodes of the DAG represent not only artifacts (pieces of data), but also processes (actions resulting in new artifacts), while the edges represent causal dependencies between them.

2.1.5 Analysing Provenance

After provenance has been queried using one of the aforementioned methods, it can be analysed to enforce norms in multi-agent systems, or to verify that proper procedures were followed in experimental and business workflows.

In workflow systems such as VisTrails, Kepler and Taverna, the actual workflow can be compared to manually constructed rules that specify so-called acceptable workflows, which are represented as workflows themselves. These systems provide visual tools for manually comparing the actual execution of workflows (which is represented as provenance) with the acceptable workflows [35]. Thus, these systems do not offer functionality for automatic compliance checking.

Miles *et al.* [101] capture provenance of experimental workflows. This provenance is later analysed to determine the semantic validity of past experiments. They determine whether the inputs and outputs of the operations applied in these experiments are of the correct type. They achieve this by analysing the underlying provenance representation (XML) against the expected types (represented as XML Schema). The main disadvantage of the model used in their technique is its strong dependence on the implementation details of XML; it is not an implementation independent theoretical model.

Curbera *et al.* [44] use provenance in a business context (so-called business provenance), which provides information about how data was produced, which resources were involved and which tasks were executed. Business provenance is captured in the form of a DAG that is analysed to detect compliance with business process regulations. To do so, they create *business control points*, which are patterns that business processes should

adhere to in their execution. Using these control points, users can detect (the cause of) problems in business processes by manually analysing the corresponding DAGs against the control points. The main disadvantage of this method is that, despite the availability of provenance in electronic form, audits are essentially performed manually.

Vázquez-Salceda and Alvarez-Napagao [142] use provenance to regulate the interactions of web services in a multi-agent context. In their work, provenance is also represented as a DAG, which provides information about the states an agent may enter into. In this context, a norms is represented as a partial order, specifying valid sequences of agent states. To enforce these norms, an enforcement component analyses the stored provenance against the norms and takes actions whenever violations are observed.

In this section, we gave an overview of the state of the art in provenance, and argued it is a promising way of making data processing transparent. As such, it can support audits to achieve information accountability. However, currently, audits are performed without the availability of provenance. Therefore, in the next section, we study the state of the art in IT auditing, and explain the benefits of using provenance instead.

2.2 Auditing IT Systems

An *audit* is an evaluation of an organization or system performed to ascertain the validity and reliability of its information and to provide an assessment of its operations [65]. The goal of an audit is to express an opinion about the organisation or system under evaluation, based on accepted standards, legislative frameworks, or mutual policies. An audit is performed by a third party known as an *auditor*, who issues a report on the results of the audit.

The traditional way of auditing involves a qualified auditor who assesses the degree of compliance of a predefined set of specifications by collecting evidence that is analysed later. This assessment is a human-driven process that follows procedures established by, among others, International Organisation for Standardisation (ISO) standards. These procedures are long, as auditors have to manually analyse large amounts of data. Thus, in the end, there is no guarantee that all data has been analysed, since the final conclusions are based on samples, rather then on all available data. Moreover, because of the human intervention, the manual audits are also prone to intentional and unintentional errors and omissions. For these reasons, many researchers [65, 124, 106, 25] have proposed to automate the auditing process so as to increase efficiency and allow for continuous auditing. The main computational auditing techniques are reviewed below.

2.2.1 Audit Trails

An *audit trail*, also referred to as an *audit log*, *activity log* or *system log*, is a chronological series of records of computer events pertaining to operating systems, applications, or user activity that is generated by an auditing system to monitor system activity [134].

Audit trails offer a way to track the actions of an individual in a system, thereby making users personally accountable for their actions. Moreover, using audit trails, it becomes possible to detect attempts of penetrating a system and gaining unauthorized access, to assess the damage of an incident or to find out how, when, and why such an incident occurred [65].

Audit trail-based systems are systems that are able to collect audit trails of system activity, which constitute evidence in an audit. These audits consist of two stages:

1. Audit Trail Collection. This process involves capturing audit trail data, which typically contains result from activities such as transactions or communications performed by individuals, systems, accounts or other entities [122].
2. Audit Trail Analysis. During this stage, the relevant audit trails are selected. These audit trails are then presented to a user (auditor) who analyses them and takes the necessary measures depending on the nature of the analysis. At this stage, the collected data is normally reduced using specialised tools to select only those trails that are important in the context of the goal of the audit.

In what follows, we first analyse the different techniques used in the first stage. Then in Section 2.2.1.2, we present an overview of the techniques used in the second stage, audit trail analysis.

2.2.1.1 Audit Trail Collection

Most audit trail-based systems collect trails at the OS level, at varying level of granularity and in different formats. These properties make the audit trails tightly coupled to the operating system in which they were created. For example, the Compartmented Mode Workstation (CMW) [115] is an extension of the UNIX 4.2 BSD operating system with added auditing capabilities. CMW collects data about command information and access to objects by users in a format that is dependant on the specific routines present in the system. Another example is the SunOS MLS System [113], which is a variant of the SunOS 4.0 distributed operating system that includes audit trail generation capabilities. This system defines a standard audit file format, however, this format is still tied to the UNIX family of operating systems, since it specifies routines names that only appear in UNIX-based systems. Moreover, in the SunOS MLS System, user names and file

names are directly referenced in the audit trails, so care must be taken to ensure these descriptors have meaning across the components (such as terminals and databases) of a distributed system.

In contrast, systems such as the VAX security kernel [124], a virtual machine monitor with audit generation capabilities, not only collects audit trails based on user names but also based on objects. In this context, object means files, devices and processes.

The main problems in these systems are interoperability, since the format of the audit trails is dependant on the OS, and scaling the quantity of recorded information for analysis can grow quickly. To address the first issue, standard formats such as Bishop's Standard Audit Trail Format [25], Normalised Audit Data Format (NADF) [106] and Common Audit Trail Interchange Format for UNIX (svr4++) [129] have been proposed.

To address the second issue, preprocessing techniques that reduce the quantity of collected trails have been developed. These techniques select a subset of trails that are in scope of the main goal of the auditing system. For example, if the goal is intrusion detection the preprocessing module selects the trails that are related with a set of suspicious activities that can lead to intrusion. Another reduction technique is to select only the trails related to a specific object or user, for example, to track files that are considered fundamental to system security. A different technique is the creation of criteria that state which trails should be collected. Examples include the DoD Trusted System Evaluation Criteria [1] and the Security Criteria for Distributed Systems [95], which explicitly define the activities and events that need to be tracked by an audit trail-based system.

When audit trails are collected in a distributed environment, these issues are even more problematic, as audit trails not only need to be collected locally but also transmitted to a central location for analysis. These issues are explored in the Distributed Auditing System (DAS) [22] and the Distributed Audit Service (DAX) [136], in which audit entities collect audits trails locally, send them to an audit manager entity that merges them, reduces them and, if necessary, transforms them to the chosen format.

To supplement these approaches, some systems collect audit trails of high-level events, in addition to those related to low-level OS events. For example, these high-level events can pertain to applications [85, 91], networks [69] or system configuration [41]. As a result, these techniques collect large amounts of information which needs to be merged with the OS-based audit trails in order to facilitate analysis.

Another area in which audit trails are used is databases. Here, audit trails are used to monitor users' actions, identify malicious behaviour, maintain data quality, and improve system performance. In this context, an audit log is a complete record of the operations on a client table over time. These audit logs can be queried to select the necessary logs to, for example, check the operations performed by a user on one or more tables [90].

2.2.1.2 Audit Trail Analysis

Various tools exist for performing automatic analysis of audit trails. As previously mentioned, the analysis tools are applied to a set of audit trails which might be preprocessed first. These trails are analysed to search for patterns that are previously established by designers. These patterns can be events that are considered suspicious, such as attempts to access files or execute commands without proper authorisation. If one or more of such events are found in the audit trails, the users related to them are flagged as suspicious and an auditor can manually verify what actually happened.

These suspicious events can be defined in files containing the different activities that can trigger suspicious behaviour. An example is the Distributed Intrusion Detection System (DIDS) [130], which creates a list of "notable events" that are collected from host monitors and sent to a director component. This director component uses an expert system to analyse the audit trails and identify suspicious activity. The expert system defines the suspicious activities based on statistical analysis of users' behaviour. If a user does something that he or she normally does not do, this event will be reported.

Another technique for defining behavioural patterns is state-modelling, which defines a system intrusion as a number of different states, each of which has to be present in the audit trails for it to be considered to have taken place. The work in this area can be divided into two subclasses [21]. The first focusses on state transitions in which the states of an intrusion form a simple chain that has to be traversed from beginning to end. The State Transition Analysis Tool (STAT) [73] uses this technique, where the actions an attacker can perform to achieve a security violation are represented by state transition diagrams. These actions are later searched for in the audit trails. If the audit trails are matched by one of these state transition diagram, a violation is detected and flagged for review by an auditor.

The second subclass uses Petri nets, which form non-linear state transition structures. For example, the Intrusion Detection in our Time (IDIOT) [86] uses a particular type of Petri nets called Coloured Petri Nets (CPN). In IDIOT, security violations are seen as sequence of actions that lead to an intruder successfully attacking the system. In this context, each violation is represented as an instantiation of a CPN, whose states and transitions represent relationships between suspicious events. These CPNs are then matched against the audit trail system in search of security violations. Using this technique, the sequence in which the events occurred is very important and needs to be carefully considered in the design of behavioural patterns and in the collection of audit trails.

In contrast to IDIOT, the Network Anomaly Detection and Intrusion Reporter (NADIR) [69] does not require a specific order of events in the collected audit trails. Rather, NADIR, which is a misuse detection system, employs an expert system to identify misuse

scenarios of user activity based on statistical analysis [74]. This is achieved by creating summary profiles of user activity that are analysed in search of the presence of suspicious events. Even though this approach is time independent, it does not collect information about the context in which the events happened, which can lead to false positives.

Another intrusion detection system is Snort [121], which is an open source network-based intrusion detection system. Snort performs real-time traffic analysis and packet logging on Internet Protocol (IP) networks. Snort can be configured in three main modes: sniffer, packet logger, and network intrusion detection. In packet logger mode, Snort logs packets that later can be analysed against a rule set defined by the user.

A different statistical technique used in the analysis of audit trails is the creation of rules based on the past behaviour of users. In this technique, misuse scenarios and suspicious activities are encoded in rules that are used by an expert system to analyse the audit trails. Systems such as the Next-generation Intrusion Detection Expert System (NIDES) [18] use this technique that combines statistical analysis and pattern matching. NIDES creates descriptive statistics rules based on the statistical analysis of users behaviour. After that, the audit trails are analysed in search of the violation of the rules to flag potential suspicious behaviour.

Now, since audit trails are sensitive to tampering and they constitute evidence of possibly wrongful conduct, they need to be properly managed and secured. This is especially important and challenging in distributed systems, where audit trails need to be transmitted over possibly insecure networks. Therefore, in the next section, we address the problem of securing audit trails.

2.2.2 Securing Audit Trails

From the perspective of security, it is important to securely collect, store and analyse distributed audit trails. Schaen and McKenney [122] explain how security affects the auditing process in distributed environments, and suggest which issues need to be addressed. However, no practical solution is given. In the same area, Huh and Martin [72] discuss the problem of having trustworthy services for auditing and logging (i.e. trusted logging). They suggest that, in distributed environments, it is possible to use remote attestation to enforce procedures and policies in a secure and interoperable way. They argue that logs should offer integrity and confidentiality through the implementation of access control. The logs should also be combined and analysed in a distributed fashion to support multiple administrative domains.

Another important security issue related to audit trails is privacy. When audit logs fall into the wrong hands, they can be used to analyse users' actions, and view personal information that is referenced in these audit logs. Both constitute a violation of users'

privacy. A suggested solution to this problem involves the implementation of access control that restricts user access to query results [107]. A similar privacy problem exists in the management of provenance, which to date has not been addressed. We come back to this issue in Chapter 8, when we discuss future work.

2.2.3 Disadvantages of Audit Trails

The focus of this thesis is the collection and analysis of electronic evidence in open and distributed systems, with the ultimate aim of verifying that personal information was used in compliance with the law. In this context, audit trails are less suitable, for a number of reasons.

First of all, few audit trail systems are capable of integrating audit trails across different platforms in a distributed way. The main problem here is a lack of standardisation of audit trail formats. Some efforts have been made by creating translators, however, this hampers systems performance.

The second problem is caused by the fact that the granularity of audit trails is too fine, i.e. audit trails are too low-level for our purpose. As a result, a lot of information needs to be collected, which leads to a problem of scalability. While attempts have been made to solve this problem by filtering the events that are tracked within the system, some researchers instead have argued that audit trails contain too little information, and more (instead of less) information is necessary.

Thirdly, by analysing audit trails it is not evident which events where triggered by the operating system, and which were triggered by user applications. Some researchers have sought to address this problem by collecting application level trails. The problem of this solution is the difficulty of merging OS trails with application trails, and to subsequently analyse the large amount of information that results.

Finally, and perhaps most importantly, audit logs are evidence of "independent events" that can be ordered by the time of their occurrence (if the corresponding time stamps were in place). In contrast, provenance is not organised according to the creation time but according to the *relationships* between data and processes that acted upon it. Thus, where audit trails fall short in explaining how a piece of data was used and how it originated, provenance is capable of accurately capturing this vital information, and thereby of achieving *information transparency*. In light of this, provenance is more suitable in the context of the central problem of this thesis, i.e. for determining whether personal information was used according to the data protection legislation.

2.3 Security

As argued in the introduction of this thesis, an important issue in the collection and analysis of any type of digital evidence is security. This importance is evident given the many efforts of the audit trail research community to protect electronic evidence from being tampered. Provenance, a different type of electronic evidence, is no exception to this and despite its novelty, some work towards protecting it has already been done.

In light of its importance, we present an analysis of the techniques used to protect electronic evidence. Specifically, we formally define security by introducing several properties that need to hold in order to help secure electronic evidence. Furthermore, we give a brief overview of the techniques used in computer systems to ensure that these properties hold.

This section is organised as follows. In Section 2.3.1 we discuss the so-called *basic security properties*, and an extra property, anonymisation, that is important to preserve privacy. Then, in Section 2.3.2 we discuss the techniques for formalising and validating the security of a system. Specifically, we discuss UMLSec, which is the approach we adopt in this thesis for automatic validation of the protocols involving the exchange of personal information.

2.3.1 Security Properties

In this section we define a set of security properties which need to hold during collection, storage and analysis of electronic evidence. Specifically, the properties defined in the ISO/IEC 7498-2 standard [3]: confidentiality, integrity, authentication, non-repudiation and access control. These properties are specially important in open systems to ensure secure communication and, consequently, protect the personal information that is transported. Beside these properties, we add a property that is decisive in preserving the privacy of information, called anonymisation. Recalling from Chapter 1, the availability property is out of the scope of this thesis and, therefore, is not described in this section. Below, we formally define each of these properties.

2.3.1.1 Confidentiality

Definition 2.1 (Confidentiality)**.** Confidentiality protects sensitive data against non-authorised disclosures by ensuring that information is accessible only to those authorized to have access [96].

A commonly used technique for ensuring information confidentiality is encryption schemes, in which only those parties that own a secret key can access sensitive data that

is transported over an insecure channel. There exist two different types of encryption schemes: symmetric-key encryption [48] and public-key encryption [96]. In the former, two parties secretly choose or secretly exchange a key, which is called *symmetric key*. To ensure confidentiality, the sender encrypts the sensitive information, which is subsequently transmitted over a (possibly) insecure network. To recover the information, the receiver decrypts it using the same key. In the latter, both the receiver and sender party create a key pair consisting of a *private key* and a *public key*. Here, the information is encrypted using receiver's public key. Then, when the receiver gets the encrypted message it can be decrypted using the receiver's private key.

2.3.1.2 Integrity

Definition 2.2 (Integrity)**.** Data integrity is the state that exists when computerized data has not been exposed to accidental or malicious alteration or destruction [2].

There are two forms of integrity. The first allows us to detect accidental data modifications, such as transmission errors due to the underlying network. The second allows us to detect malicious data modifications, insertions or deletions [96]. In terms of security, the second is our main concern, since we assume open and distributed systems, in which data is transmitted over unsecured networks.

The most commonly used technique to ensure this property is hash functions. A hash function is a computationally efficient function that maps binary strings of arbitrary length to binary strings of some fixed length, called hash-values [96]. The hash-value associated with a binary string is unique, so if one bit of such a string is changed, its corresponding hash-value will be different. Moreover, hash functions are chosen to be collision resistant, which means that it is very difficult to find two different binary strings whose hash values are the same. Using this property, an entity receiving a piece of data and its corresponding hash-value is able to verify the integrity of such a piece of data by re-calculating its hash-value and comparing it with the given one.

There exist two types of hash functions. The first type is called "keyed hash functions", which is also know as Message Authentication Codes (MAC), because its specific purpose is message authentication. These functions require a secret key, and consequently, they not only provide data integrity but also data-origin authentication, because it is assumed that apart from the recipient, only the sender knows the secret key necessary to compute the MAC [96]. An examples of these a keyed hash function is HMAC [24].

The second type is called "unkeyed hash functions", which are also known as Modification Detection Codes (MDC). In this type, the hash function provides a short message digest of a (possibly large) piece of data. If the data is (maliciously) altered, the digest of this altered data should be different from the original digest [96], enabling the detection of the alteration. Examples of these functions are MD5, SHA-1 and its predecessor

SHA-2 [117]. The unkeyed hash functions are also used in digital signatures schemes by hashing long messages (using a publicly available hash function) and signing just the created hash-value, instead of the entire message.

2.3.1.3 Authentication

Definition 2.3 (Authentication)**.** Authentication verifies the supposed identity of an actor (software or user) and ensures that the information exchanged originated from the correct source.

The type of authentication in Definition 2.3 is also known as *Entity Authentication* or *identification* (as opposed to *message authentication* discussed above). Entity authentication assures one actor that the identity of the second party involved is correct. If the second party is authenticated to the first one, and vice versa, the authentication is called *Mutual Entity Authentication*.

Entity authentication can be implemented using password authentication or login protocols [143]. In these protocols a password, which is associated with a user, serves as a shared secret between the user and system. The system authenticates the user by asking the user to enter his user ID and password. The system checks that the password matches the corresponding user ID.

Zero-knowledge identification protocols [96, 46] are also used to provide authentication. These protocols are specifically designed to allow a prover (the party who wishes to authenticate itself) to identify itself to a verifier (the party who wishes to establish the prover's identity) by demonstrating the possession of a secret without revealing any information whatsoever about the secret.

Another method used to support entity authentication is a challenge-response identification protocol [30, 96]. In these protocols one entity (the claimant) proves its identity to another entity (the verifier) by demonstrating knowledge of a secret. This secret is associated with that entity only, so proof of possession of the secret identifies its owner. The demonstration is performed without revealing the secret itself to the verifier. To do so, the claimant provides a response to a time-variant challenge (such as a random number), where the response depends on both the entity's secret and the challenge. Exemplar protocols of this type are the Transport Layer Protocol (TLS) [27] and all its variants (such as [19]).

The TLS protocol is a very well-known authentication protocol that is currently in use for protecting most Internet traffic that involves sensitive information (i.e. online banking and shopping, email, etc.). Because of its importance to this thesis — we use a variant of this protocol in Chapter 6 — we describe it in detail below.

Transport Layer Security (TLS) is a cryptographic protocol that secures Internet communications by preventing eavesdropping, tampering and message forgery. TLS is a widely-used IETF (Internet Engineering Task Force) standard that allows client/server applications to negotiate a secure connection using a handshaking procedure in which the participants agree on various parameters for establishing the connection. In doing so, the server authenticates itself to the client and the client and server together create the shared session key that is used to encrypt and decrypt the data contained in messages. The TLS protocol proceeds as follows:

- The handshake begins when a client connects to a TLS-enabled server by requesting a secure connection. It presents a list of supported cipher suites, which contains the supported ciphers and hash functions. This is done in the clientHello message shown in Figure 2.2.
- From this list, the server chooses the strongest cipher and hash function that it supports and notifies the client.
- The server identifies itself using a digital certificate. The certificate usually contains the server name, the trusted certificate authority (CA) that signed the certificate and the server's public encryption key. All this is achieved by sending the serverHello message in Figure 2.2.
- When the client receives this information, it may contact the certificate authority to confirm the validity of the server's certificate before continuing the execution of the protocol.
- In order to generate the session key used for securing the connection, the client encrypts a random number with the server's public key and sends the result to the server. If the private keys is securely kept by the server, only it is able to decrypt the message with its private key. This is achieved by the sending of the keyExchange message in Figure 2.2.
- From the random number, both parties use the same algorithm to generate the session key that is used for the encryption and decryption of application data.
- Finally, the server sends a finished message to indicate that the handshake phase was been completed successfully.

Once these steps have been completed, the handshake phase terminates, and a secure connection is established. Using this connection, all application data is protected because it is encrypted by the sender and decrypted by the received using the created session key until the connection closes. If any of the above steps fail, the TLS handshake fails and no connection is established. Note that, for reasons of clarity, the description above

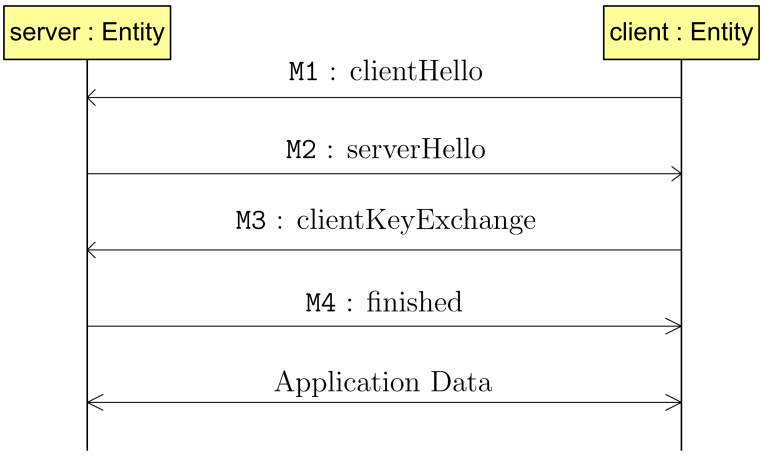


FIGURE 2.2: TLS Protocol

and in Figure 2.2 is presented at a high level of abstraction. For more details about this protocol, including implementation details, we refer the reader to [27].

As suggested by researchers [43, 19], TLS is an expensive protocol, not only in terms of the number of messages that are exchanged, but also because of the expensive cryptographic operations that the server needs to perform, which can potentially become a centralised bottleneck in the system.

For our work, these properties of TLS have two negative implications. First, entities need to mutually authenticate each other in our system (Requirement A). Secondly, our system gathers provenance about the protocol's execution in order to keep track of the security protocols that are used to protect sensitive application data in our system. The more messages are exchanged, the more provenance will have to be recorded, which slows down the process of storing, querying, and processing provenance.

Hence, we opt for an improved and more efficient version of TLS, called the Optimised TLS Handshake Protocol [19, 79, 71] or OTLS[1]. We emphasise that this choice is by no means essential in our work, and that TLS or other variants in the TLS family [127, 43, 126] can also be used. However, OTLS effectively addresses the aforementioned drawbacks of TLS in the context of our proposed system.

Now, in more detail, the difference between OTLS and TLS is that the former is more efficient in terms of communication overhead since it requires three messages to perform mutual client-server authentication, while the latter requires four [19]. In addition, it reduces server side processing (without compromising security), by reversing the roles of client and server so that the generation of the pre-master secret[2] is performed by the server instead of the client. This role reversal achieves the aforementioned reduction

[1]This abbreviation is not used by the authors, we use it in this thesis to distinguish OTLS from "standard" TLS.

[2]The pre-master secret is a randomly generated number that will be used to generate a master secret from which the encryption keys will be derived. This secret is also called "shared secret" as it is known by the client and the server after the protocol execution.

of the number of messages of a TLS handshake to from 4 to 3 and allows the client to perform the expensive private key operations, thereby reducing the load on the server (as compared to TLS). In contrast, in the TLS protocol, the public key used by the client to encrypt the shared pre-master secret is obtained from the server certificate. With OTLS, the server chooses the pre-master secret and sends it to the client encrypted with the client's public key. Therefore, the expensive decryption operation is performed by the client.

Crucially, OTLS has been proven to be as secure as TLS in terms of providing security to Internet communications by preventing eavesdropping, tampering and message forgery [71]. Thus, the use of OTLS has the advantage of providing the same security properties as TLS with lower overhead in terms of computation and communication.

Now, the OTLS protocol proceeds as follows (see Figure 2.3):

- The handshake begins when a client connects to a OTLS-enabled server by requesting a secure connection, and identifies itself by presenting its certificate and a list of supported cipher suites and hash functions.
- When the server receives this information, it may contact the certificate authority to confirm the validity of the client certificate. At this point, the client has authenticated itself to the server.
- From the list of supported functions, the server chooses the strongest cipher and hash function it supports and notifies the client. The server also identifies itself using a digital certificate, which contains the server name and the server's public encryption key signed by the trusted certificate authority (CA). The server also sends a server certificate containing the pre-master secret, the client's random number and the client's public key. This certificate is signed with server's private key and encrypted with client's public key.
- When the client receives this information, it may contact the certificate authority to confirm the validity of the server's certificate before continuing with the protocol. At this point, the server has authenticated itself to the client.
- The client decrypts the pre-master secret with its private key and verifies the signature by extracting the public key from the temporary certificate. Later, the client encrypts the server's name with the pre-master secret and sends the result to the server. Only the server is able to decrypt this using the pre-master secret.
- Finally, from the pre-master secret, both parties use the same algorithm to generate the session key that is used for the encryption and decryption of application data.

Identical to TLS, after a successful handshake a secure connection is established, in which all application data is encrypted and decrypted with the session key until the

connection closes. If any of the above steps fail, the OTLS handshake fails and no connection is established.

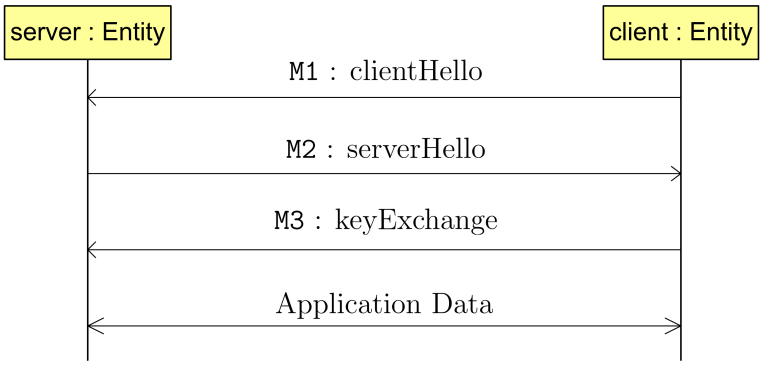


FIGURE 2.3: OTLS Protocol

In sum, in our work we use OTLS instead of standard TLS, because it is more efficient in terms of computation and communication. Specifically, since OTLS requires less messages to be exchanged, less provenance needs to be collected, processed and stored, resulting in a better performance of our system.

We return to the OTLS protocol in Chapter 6. Here, we use it to provide mutual authentication between the entities of our proposed architecture. In addition, we make the necessary alterations to record provenance of its execution, in order to keep track of the security measures that were used when processing sensitive application data.

However, before ending the discussion of authentication protocols in general, and (O)TLS in particular, note that no matter how well designed and tested a protocol may be, a flawed implementation can cause vulnerabilities. These vulnerabilities have the potential of compromising the security properties of our proposed system (or any system in general). Specifically, it has been shown by many authors [118, 140, 140] that TLS has a number of vulnerabilities. For example:

- A design flaw in the protocol could allow an attacker to successfully inject data in an encrypted session using a man-in-the-middle attack. The problem occurs during the renegotiation of the TLS channel, specifically, when client certificates are used [118]
- By eavesdropping messages sent over a TLS channel, it is possible to decrypt some of the data contained in these messages (such as passwords). This vulnerability is due to the way error handling is implemented in certain applications [140].
- In some cases, authentication tokens and cookies used in many HTTPS requests, which uses TLS, can be recovered from the messages (encrypted) messages sent over a TLS channel [51]. These tokens and cookies can be subsequently used to gain unauthorised access to websites.

Needless to say, the vulnerabilities of implementation of authentication protocols need to be taken into account when implementing our system. Practically, this means that the libraries that implement these protocols need to be updated regularly, and that additional monitoring systems (that, for example notify human operator of suspicious behaviour) should be put in place. Whilst important, an in-depth discussion of these issues is outside the scope of this thesis.

2.3.1.4 Access Control

Definition 2.4 (Access Control)**.** Access control, which is also called Protection or Authorisation, is a security property that protects shared resources against unauthorised access according to some access control policy [139].

Access control mechanisms are implemented to mitigate the risks of unauthorized access to data, resources, and systems. Some examples of these mechanisms are classic access control techniques, such as Access Control Lists (ACLs). An ACL is the most basic form of access control in which each resource in a system has a list associated with it that specifies which entities are allowed access to it, and which actions they can perform [89]. An example of this mechanism is the Discretionary Access Control (DAC) in which the access policies to a resource are determined by the owner of such a resource.

Another access control mechanism is the Role-Based Access Control (RBAC) mechanism. With this mechanism, the right of access to a resource is determined by the relationship between the requester and the owner of the requested resource, i.e. the requester's role determines whether access to a specific resource will be granted or denied [89].

Attribute Based Access Control (ABAC) is another access control method in which the access control decisions are made based on a set of characteristics, or attributes, associated with the requester, the environment, or the resource itself. [89].

Finally, Mandatory Access Control (MAC) is an access control mechanism determined by systems, not owners. Users and resources have a set of security attributes associated with them. When users request access to a resource, an authorization rule enforced by the system examines these security attributes and decides whether access is granted [89].

Whilst access control is a vital aspect of system security, this thesis does not focus on the implementation of access control or the verification of its presence. Rather, we assume that one of the aforementioned techniques is correctly implemented.

2.3.1.5 Non-Repudiation

Definition 2.5 (Non-Repudiation)**.** Non-repudiation makes it impossible for an entity to deny the sending or receiving of a given message [96].

The most commonly used techniques for ensuring non-repudiation are digital signature schemes. In these schemes, the sender and the receiver of a message have a pair of keys: one private and one public. The sender of a message signs it using its private key. As the private key is only known by the sender, this signature can only be created by the sender. If the receiver of the message wishes to prove a message was indeed sent by the sender, the signature can be verified using the publicly available sender's public key.

In order to properly offer the non-repudiation property, a digital signature scheme should have two properties. First, a signature generated from a given message and a given private key should allow for the verification of the authenticity of that message using the corresponding public key (trapdoor function). Secondly, it should be computationally infeasible to generate a valid signature by a party who does not possesses the private key (one-way function) [96].

Digital signature schemes can be classified in two types. The first type, digital signature schemes with appendix, requires the original message to verify the signature. This type of signature scheme appends the message signature to the message itself. However, by just creating a signature and adding it to the original message the signature is vulnerable to a substitution attack, i.e. given a valid signature for a message it is easy to modify the given signature in such a way that is valid for another known message. This type of attack can be prevented by applying a cryptographic hash function (See Section 2.3.1.2) to the original message before being signed. To verify the validity of a signature, the hash-value of the message is computed first and a new signature is calculated from this hash-value. By comparing this new signature with the one the appended to the message, the validity of the signature can be established. An example of this type of digital signature is DSA [84], which is used in the FIPS-186-3 standard [16].

The second type is digital signature schemes with message recovery. This type of scheme does not require the original message to verify the signature. Instead, the original message is recovered from the signature itself. Therefore, the process to verify a signature in these type of schemes is also called extraction. After the original message has been extracted from the signature, there is no need of additional steps to establish the validity of the message. This scheme can also be used along with a cryptographic hash function. In such a case, the message is hashed and the obtained hash-value is signed. Then, the corresponding hash-value can be extracted from the given signature and the authenticity of the message is verified by comparing the extracted hash-value with the locally

computed hash value of the same message. This is the process that most of current applications implement [110]. Examples of this type are RSA [120] and the ISO/IEC1991 [4] signature schemes.

In practical terms, digital signatures with appendix are applied to messages of arbitrary length, while digital signature schemes with message recovery are applied to messages of a fixed length. In order to obtain messages of a fixed length, these can be hashed. Another option is to break the message into blocks of a fixed length that later can be signed. A problem of this approach is that signature generation is relatively slow making the signing and extraction of one message time consuming. Therefore, the preferred method is to hash [96]. However, in formal methods as the one presented in Section 2.3.2.2, signature schemes with message recovery make use of this scheme.

Due to their evident advantages, the digital signature schemes with message recovery are the most widely-used. Hence, in the rest of this document we use these type of schemes. When the security characteristics of our system requires the use of a cryptographic hash function, we use it and clearly indicate it. We come back to this in more detail in Section 2.3.2.2.

A good survey that presents detailed information about these schemes is found in [102].

2.3.1.6 Anonymisation

Anonymisation is another important property that should hold when information that is regarded as *sensitive* is processed. Sensitive data can be defined as any information, which through loss, unauthorized access, or modification could adversely affect the interests of its owner. For example, information relating to race or ethnic origin, political opinions, religious beliefs, physical/mental health, trade union membership or criminal activities. Through anonymisation, the identity of the person associated with this sensitive information has to remain hidden in the result of processing, if this result is made available to users who are unauthorised to access sensitive information.

As an initial attempt to hide individuals' identities when sensitive data is being processed, a technique called *data de-identification* was proposed [138]. In this technique, information that could be used to identify an individual is removed or replaced by *pseudo-data*. Pseudo-data is an identifier, which is used instead of information that can identify an individual. For example, instead of presenting the name of an individual a sequence number can be used.

However, data de-identification provides no guarantee of anonymity as processing results might contain other data that can be linked to publicly available information to re-identify individuals or to infer other sensitive information [42]. This is known as a linking attack, in which external data is combined with a set of de-identified data in

order to infer hidden private information [37]. To effectively hide individuals identity, the anomymisation property should hold:

Definition 2.6 (Anonymisation)**.** Anonymisation ensures that sensitive personal information can not be recovered by unauthorised users through the analysis of (publicly) available processing results [135].

Data can be anonymised by implementing different techniques, such as Differential Privacy [52], k-Anonymity [133, 75], ℓ-Diversity [93] or Perfect Privacy [97]. These techniques are mainly used to avoid disclosure of sensitive information from databases that manage personal data.

Now that we have defined the properties that need to hold in a secure system, we now shift our focus to techniques for ensuring that systems indeed exhibit these properties.

2.3.2 Formalising Security Properties

Security should be considered during each phase of the software development life cycle. This way, it is possible to model and analyse the security properties of a system at the software architecture design level. Recently, some approaches have been proposed that support this modelling and analysis. These approaches can be classified as semi-formal, formal and aspect-oriented. Semi-formal approaches, such as UML, use modelling languages for specification and visualisation to define a vocabulary for annotating these models and include information about security properties on this model. Formal approaches use formal methods (mathematical modelling techniques) to specify, develop, and verify computer system (software and hardware) design. These methods offer a way to verify security properties in the systems or models before its implementation. Finally, aspect-oriented techniques approach security as being orthogonal to the application's functionality. Using so-called aspects, i.e. parts of software that are relevant to a particular concern, non-functional security properties can be modelled independently from the core application code.

In this thesis, we use both semi-formal and formal methods, which, unlike aspect oriented techniques, give us architecture design tools as well as formal methods to verify the correctness of the model. Specifically, we focus on UML and UML-sec, which combine the development of models with their verification.

2.3.2.1 Unified Modelling Language

In the next section we present an approach to modelling security requirements using UMLsec, which is an extension of Unified Modelling Language (UML). Thus, we first

briefly discuss UML [9] itself. UML is a general-purpose modelling language that includes a graphical notation used to create an abstract model of a system. It is one of the most commonly used standards for specifying object-oriented software systems, and has been adopted as the industry-standard language for visualising, constructing, and documenting the artifacts of software systems.

UML Elements UML provides a number of diagram types as a mechanism for entering model elements into the model. In this section, we briefly introduce the relevant UML diagrams and components that are used throughout this thesis.

Use case diagrams: are visual descriptions of a system's behaviour pertaining to a business task or requirement, which focus on the relation between the system, external entities, and uses cases. It shows the relations between *actors*, the users, people, or other systems, and the *use cases*, the scenarios in which the actors interact with the system.

Sequence Diagrams: model the communication between objects in a specific process as a sequence of method calls. In these diagrams, the temporal relation between the method calls is shown visually in the form of arrows that represent the exchange of messages between objects.

Tagged Value: is an explicit way to define properties of an element in the model. Its notation is $\{tag = value\}$ where *tag* is the property name and *value* is the corresponding value assigned to the tag. These values are called pseudo-attributes of model elements because the semantics of their description are outside the scope of UML definition.

2.3.2.2 UMLSec

UMLsec provides a formal specification of security elements that can be included in a UML model. We will use UMLsec to include and formalise security characteristics in the architecture presented in Chapter 5. Based on this formalisation, we can automatically verify the security characteristics that are relevant to our work presented in Chapter 6.

In more detail, UMLsec [79] is a UML extension for the development of secure systems that provides a formal specification of security elements in a UML model. In UMLsec, security requirements and assumptions within the system environment are formulated using tagged values. These can be used by an automatic model checker to determine whether a system design meets certain security requirements. Additionally, UMLsec defines a cryptographic notation to model cryptographic protocols and an adversary model to model security attacks against the system. In what follows, we briefly explain both.

Cryptographic Notation The cryptographic notation is a formal notation used in the sequence diagrams to model cryptographic protocols that provide specific security properties to the system, such as those mentioned in Section 2.3.1. This notation contains formal definitions for modelling cryptographic data in a UML specification, which can later be analysed. These definitions are presented below and taken from [79].

The notation defines a set of keys, which is denoted as **Keys**. A key $k \in \mathbf{Keys}$ can be symmetric or asymmetric. A *symmetric key* is used for symmetric encryption schemes and an *asymmetric key* for asymmetric encryption schemes. If $k \in \mathbf{Keys}$ is an *asymmetric key*, its inverse exists and is denoted as k^{-1}. Thus, if k is used as a *public key*, then k^{-1} is the corresponding *private key*. The notation also defines a set of variables **Var** that contains the variable names defined in the diagrams and a set of data values **Data** that contains the message names of the diagrams, including nonces[3] and other secret information. Moreover, UMLsec defines an *algebra of cryptographic expressions*, denoted as **Exp**, which is an algebra generated from the set $\mathbf{Data} \cup \mathbf{Var} \cup \mathbf{Keys}$. The operations and properties of this algebra are presented below and taken from [79].

If $A, B \in \mathbf{Exp}$, UMLsec defines the following operations [79]:

$$A \parallel B \quad A \text{ concatenated with } B \tag{2.1}$$
$$head(A) \quad \text{Head of } A \tag{2.2}$$
$$tail(A) \quad \text{Tail of } A \tag{2.3}$$
$$\{A\}_k \quad A \text{ encrypted using the key } k \tag{2.4}$$
$$Dec_k(A) \quad A \text{ decrypted using the key } k \tag{2.5}$$
$$Sign_k(A) \quad A \text{ signed using the key } k \tag{2.6}$$
$$Ext_k(A) \quad A \text{ extracted using the key } k \tag{2.7}$$
$$\mathtt{h}(A) \quad \text{Hash of A} \tag{2.8}$$

For each $A \in \mathbf{Exp}$, we have the following properties [79]:

$$Dec_{k^{-1}}(\{A\}_k) = A \text{ For all } A \in \mathbf{Exp} \text{ and } k \in \mathbf{Keys} \tag{2.9}$$
$$Ext_k(Sign_{k^{-1}}(A)) = A \text{ For all } A \in \mathbf{Exp} \text{ and } k \in \mathbf{Keys} \tag{2.10}$$

The *Ext* function, the inverse of the *Sign* operation, recovers the data that was signed, i.e. does extraction. Thus, when the *Sign* operation is applied to plain, sensitive data, the *Ext* function can be used to recover it. As a result, when used in isolation, the

[3]Nonces are time-variant parameters which serve to distinguish one protocol instance from another to prevent replay attacks [96]. In these attacks, an adversary records the messages exchanged as part of a security protocol, and reuses these in an attempt to break system security.

Sign operator ensures non-repudiation, but does not ensure confidentiality, since the plain data can be recovered by anyone in possession of the corresponding public key. However, if instead of the plain data, the hash value of the data is signed, the plain data is not recoverable from the signature. Moreover, by doing so, the integrity of the data can be ensured, since the signed hash can not be altered by a third party (i.e. anyone not in possession the private key) without being detected. Therefore, in the remainder of the thesis, if we only need to ensure non-repudiation of non-sensitive data (such as the digital certificates in Figure 6.1), only the *Sign* and *Ext* operations are used. If, however, non-repudiation *and* integrity of data need to be ensured, the *Sign* operation is applied to the hash value of sensitive data (for example, this is done in the Data Request, Task Request and Query Request protocols where sensitive application data is exchanged) and verification is performed by hashing the original data and comparing it to the result of the *Ext* function.

Finally, the laws regarding concatenation [79, page 37]. Note that the concatenation operation is equivalent to the cons function used to handle lists in most programming languages.

For all expressions $E_1, E_2, E_3 \in \mathbf{Exp}$

$$(E_1 \parallel E_2) \parallel E_3 \quad = \quad E_1 \parallel (E_2 \parallel E_3) \tag{2.11}$$

For all $E_1, E_2 \in \mathbf{Exp}$ the operation $head()$ is defined as:

$$head(E_1 \parallel E_2) \quad = \quad E_1 \tag{2.12}$$

For all expressions $E_1, E_2 \in \mathbf{Exp}$ such that there exist no E, E' with $E_1 = E \parallel E'$ the operation *tail* is defined as:

$$tail(E_1 \parallel E_2) \quad = \quad E_2 \tag{2.13}$$

Moreover, for each $E \in \mathbf{Exp}$, we use the following abbreviations [79]:

$$fst(E) \quad = \quad head(E) \tag{2.14}$$
$$snd(E) \quad = \quad head(tail(E)) \tag{2.15}$$
$$thd(E) \quad = \quad head(tail(tail(E))) \tag{2.16}$$

The defined algebra assumes that it is not possible to obtain plain data values from encrypted data without the decryption key. Moreover, it assumes that the symmetric

encryption technique provides integrity by implementing Message Authentication Codes (MAC), which were explained in Section 2.3.1.

Adversary The adversary model represents a network attacker that can eavesdrop, modify or insert messages on the communication channel with malicious intentions. This adversary model relies on an extended Dolev-Yao adversary model [50], in which an adversary can read messages sent over the communication channel, in an attempt to derive secret knowledge and break system security.

In the formalisation process, each sequence diagram is associated with an adversary object that is used to verify that the modelled protocol exhibits the aforementioned security properties. To this end, the adversary object contains three types of predefined values: `secret`, `initial knowledge` and `guard`. The values labelled `secret` are the data items that should be protected from the attacker. The values in the set `initial knowledge` denote the information known by the attacker beforehand (e.g. public keys), and $\texttt{guard}_n$ represents the operations to be performed by the receiver of message n after it has been received, but before it is processed. Finally, the simultaneous execution of the adversary model and the sequence diagram model is undertaken to search for possible attacks. The next section describes how this evaluation is performed.

Using the adversary model, it is possible to model attacks on a specific part of a system. The adversary is modelled by defining the actions it can perform on each part of the system. For example, an adversary may be capable of reading messages exchanged over the communication links of the system, but can not read records in a database. Different attacks to the system are defined in the form of *abstract threats*, which are the common attacks to the information channels:

- `delete` indicates that the adversary can delete messages from the communication channel.
- `read` indicates that the adversary can read messages on the communication channel.
- `insert` indicates that the adversary can insert messages on the communication channel.
- `access` indicates that the adversary has access to the communication channel or to a physical system node.

When an adversary attack takes place in an actual channel, *concrete threats* can be obtained from the real vulnerabilities of such channel. For example, on an Internet connection without implementation of security techniques, the adversary's concrete threats are delete, read and insert messages.

2.3.2.3 UMLSec Automatic Verification

To verify that the security requirements expressed in UMLsec are maintained during the execution of a protocol, the Viki model checker can be used [80]. Viki takes as input a UML sequence diagram and its associated adversary model, and returns the possible attacks that can be performed by the given attacker in the modelled protocol. Viki obtains the security requirements from the UMLsec elements and the predefined values used in sequence diagrams [80]. These requirements are formalised in First-Order Logic and analysed with automatic theorem provers (e-SETHEO [131] and SPASS [145]) to find flaws. If a flaw is found, a Prolog engine can be used to generate the attack trace, which can provide a system designer with valuable insights to solve it.

Using this tool, we can ensure that the modelled attacks are unsuccessful in the model. Since, in this context, a successful attack means that the system does not exhibit one of the security properties [125], Viki enables us to verify system security. Therefore, in Chapters 6, we use the techniques described in this section to model and verify the security of our Provenance-based Auditing Architecture.

2.4 Data Protection Legislation

As we mentioned in the introduction to this thesis, one of the initial attempts to protect personal information from misuse is the creation of *Data Protection Legislation*. Data Protection Legislation protects individuals by making the misuse of personal information illegal. In general, it provides a set of rules that organisations should follow when personal information is handled. Moreover, it defines procedures for verifying that these organisations processed individuals' personal information in compliance with the applicable legislation.

In this section we overview three important examples of Data Protection Legislation: The Data Protection Act (Section 2.4.1), which is the UK data protection legislation that safeguards UK citizens' personal data; Safe Harbor (Section 2.4.2), which is a US framework that protects personal information managed for commercial purposes between the US and the EU; and the HIPAA (Section 2.4.3), which is a legislative framework for protecting patients' medical information.

2.4.1 Data Protection Act

The Data Protection Act (DPA) 1998 [7] provides protection for an individual's personal information by placing restrictions on how organisations can use personal information — including how they should acquire, store, share or dispose of it. The UK's DPA is an implementation of the European directive [5] that enforces the protection of individuals'

personal data as it is processed within or moved between Member States of the European Union. The DPA introduces a new regime for the distribution of information using computer based systems by legislating the computational processing that organisations are allowed to perform on individuals' personal information.

2.4.1.1 Terminology

The DPA defines three entities that are involved in the processing of personal information:

Definition 2.7 (Data Controller (DC))**.** A Data Controller is an individual or organisation that decides the purpose for which, and the manner in which, personal information is to be processed.

Definition 2.8 (Data Subject (DS))**.** A Data Subject is an individual whose information is held by a Data Controller.

Definition 2.9 (Data Processor (DP))**.** A Data Processor is an individual or organisation, other than an employee or subsidiary of the Data Controller, that processes personal data on behalf of the Data Controller.

In the DPA, the information processed by these entities is classified into two types:

Definition 2.10 (Personal Data or Personal Information)**.** Any information related to a living individual that can be used to identify that individual. For example, name, date of birth, address, etc. including opinions about this individual or indications of intent towards him or her.

Definition 2.11 (Sensitive Personal Data)**.** A special subclass of personal data that can only be processed under certain conditions. Examples of this type are: racial or ethnic origin, political opinions, religious beliefs, physical and mental condition, sexual life, offences, etc.

Both types of information could be processed under certain conditions. In the DPA, as it can be seen in the next definition, processing means anything that can be done with data.

Definition 2.12 (Processing)**.** The ways of processing information include obtaining, recording, holding or carrying out any operations, including:

- Organisation, adaptation or alteration.
- Retrieval, consultation or use.
- Disclosure, transmission or dissemination.

- Alignment, combination, blocking, erasure or destruction.

This definition of processing is not precise enough for our purposes. For example, it does not distinguish between the act of collecting and using data, and it does not explicitly specify the purpose for which processing it performed. However, this is vital information for deciding whether processing was conducted according to the DPA principles discussed in the next section. Therefore, we make the following additional definitions:

Definition 2.13 (Purpose)**.** A Purpose is the intention for which the data is to be processed.

Definition 2.14 (Collected Data)**.** A set of Collected Data is the information obtained from Data Subjects.

Definition 2.15 (Used Data)**.** A set of Used Data is the information used in the processing of a result.

2.4.1.2 Principles of the Data Protection Act

The Data Protection Act consists of eight statutory principles that a Data Controller should adhere to in order to process a Data Subject's information. Below, the principles are presented and briefly explained. In Chapter 5, we develop the Compliance Framework that is capable of automatically verifying these principles using the provenance collected by the Provenance-based Auditing Architecture described in Chapter 4.

Principle 1. *Personal data shall be processed fairly and lawfully and, in particular, it shall not be processed unless*:

(a) *At least one of the conditions in Schedule 2 is met.*

(b) *In the case of sensitive personal data, at least one of the conditions in Schedule 3 is also met.*

Principle 1 introduces the concept of *purpose* through Schedules 2 and 3. In this context, purpose is the intention for which the data is to be processed. Thus, a DC should have a purpose for processing DS's personal data. In more detail, in Schedule 2 and 3 the DPA states the legal purposes for which a DS's information can be processed. The former, used in case **(a)**, specifies legal purposes for using personal data (Definition 2.13). Examples of legal purposes include the consent of a DS, a contract between a DC and a DS, a legal obligation, the vital interest of a DS or functions of public interest. The latter, which is used in case **(b)**, specifies legal purposes of processing sensitive personal data. Examples of these legal purposes are the *explicit* consent of a DS, to perform any right or obligation, to protect vital interest of a DS, medical purposes, administration of justice, etc. For more information about Schedule 2 and 3 the reader should refer to [10].

Principle 2. *Personal data shall be obtained only for one or more specified and lawful purposes, and shall not be further processed in any manner incompatible with that purpose or those purposes.*

If a DC requests personal information from a DS, then such information should only be obtained with a specified and lawful purpose, which should be provided to the DS. Furthermore, if a DC obtains information from a DS, such information can only be processed in accordance with the stated purpose.

Principle 3. *Personal data shall be adequate, relevant and not excessive in relation to the purpose or purposes for which it is processed.*

This principle requires that a DC obtains from a DS only the information that is necessary to perform the processes established in the initial purposes. Thus, the data requested from a DS has to be relevant for the stated purposes. Moreover, a DC should not request, collect or use more information than required.

Principle 4. *Personal data shall be accurate and, where necessary, kept up to date.*

The personal information held by a DC should be updated in a way specified by him or her to ensure that the DC only processes correct information. Hence, a DC has to provide the means for a DS to perform an update of his or her information.

Principle 5. *Personal data processed for any purpose or purposes shall not be kept for longer than is necessary for that purpose or those purposes.*

This principle requires the destruction or deletion of personal data that is no longer necessary, i.e. when the processing established in the purpose has been accomplished. The aim of this principle is it must be impossible to identify such an individual through previously processed data, after the elimination of one individual's personal information.

Principle 6. *Personal data shall be processed in accordance with the rights of data subjects under this Act.*

This principle states that any process applied to personal data should be in accordance with the Data Subject seven rights, which are presented in [7]. These rights are:

Access to personal data Data Subjects have the right to access their personal data, which is stored by a DC, whenever the DS wants.

Prevention of processing likely to cause damage or distress The processing of DS's information should not cause damage or distress to him or her.

Prevention of processing for direct marketing A Data Subject's personal information cannot be used for direct marketing purposes.

Prevention of automated decision-taking A Data Subject's personal information cannot be used for automated decision-taking purposes.

Rectification, blocking, erasure, and destruction The DC needs to perform these actions to a DS's personal information if so requested by a DS.

Compensation If the DPA is not followed by an organisation, the affected Data Subject can demand compensation.

Request for assessment Data Subjects can request an audit on the use of their personal information to verify that DCs followed the DPA principles. A report of the audit results has to be sent to the DS involved.

Principle 7. Appropriate technical and organisational measures shall be taken against unauthorised or unlawful processing of personal data and against accidental loss or destruction of, or damage to, personal data.

This principle indicates that a DC should ensure an appropriate level of security for all the processes involving the personal information of a DS. To achieve this, a DC should implement technical and organisational measures to protect personal data against accidental or unlawful destruction, loss, alteration, disclosure or access. These measures are particularly important when processing involves the transmission of personal data over a network.

Principle 8. Personal data shall not be transferred to a country or territory outside the European Economic Area unless that country or territory ensures an adequate level of protection for the rights and freedoms of data subjects in relation to the processing of personal data.

Principle 8 places restrictions on the transfer of personal data outside the European Economic Area. Only the countries that fulfil the requirements of this Act can request personal data from Data Subjects situated in UK. For example, the United States has implemented the *Safe Harbor* [8] to ensure the adequate level of protection required by the DPA. The *Safe Harbor* legislation is introduced in the next section.

2.4.2 Safe Habor

The Safe Harbor [8] is a legislative framework that protects the privacy of personal information transported between the European Union and the United States for commercial purposes. Because of this, Safe Harbor has seven principles that are similar to the ones discussed above. These principles require the following:

Notice Organisations should notify individuals about the purposes for which they collect and use individuals' information. Organisations also should provide their contact information about how individuals can contact them with any enquiries or complaints. Finally, organisations should notify the individual with the types of third parties to which they intend to disclose the personal information and the

choices and means organisations offer for limiting the information use and disclosure.

Choice If individuals' personal information is disclosed to a third party or used for a purpose incompatible with the collection purpose, then organisations should request individuals for authorisation. For sensitive information the same principle applies but the request should be affirmative or explicit.

Onward Transfer (Transfers to Third Parties) Organisations transferring information to a third party should apply the notice and the choice principles. Moreover, if an organisation transfers such information to a third party that is acting as an agent, then such agent should be subscribed to the Safe Harbor or to the European Directive. If not, then the organisation and the agent should have a written agreement requiring that the agent provides at least the same level of privacy protection as is required by the relevant principles.

Access Individuals should have access to their personal information, which is held by organisations, to correct, amend or delete that information where it is inaccurate.

Security Organisations should implement the necessary security measures to protect personal information from loss, misuse and unauthorised access, disclosure, alteration and destruction.

Data integrity The personal information collected by the organisations should be relevant for the purpose it is to be used for. Thus, organisations might ensure that the data collected is reliable (accurate, complete and current) for its intended use.

Enforcement To ensure compliance with the Safe Harbor principles, organisations should

1. Implement the necessary mechanisms to investigate and resolve any individual's complaint.
2. Offer procedures for verifying that organisations have implemented the Safe Harbor principles.
3. Implement mechanisms to remedy problems arising out of a failure to comply with the principles.

2.4.3 HIPAA

Health Insurance Portability & Accountability Act (HIPAA) of 1996 [6] is a legislative framework aimed to improve the efficiency in healthcare delivery by standardising electronic data interchange and, at the same time, protecting the confidentiality and security of such electronic health data through setting and enforcing standards.

HIPAA is composed of four main parts:

1. Standards for Electronic Transactions. The term *Electronic Health Transactions* includes health claims, health plan eligibility, enrolment and disenrolment, payments for care and health plan, claim status, first injury reports, coordination of benefits, and related transactions. Thus, all the health organisations should use one format in the management of such transactions to simplify and improve the efficiency in the use of this information.

2. Unique Identifiers for Providers, Employers, and Health Plans. This rule requires hospitals, doctors, nursing homes, and other healthcare providers to obtain a unique identifier when filing electronic claims with public and private insurance programs.

3. Security Rule. This rule requires that organisations provide a uniform level of protection of all individuals' health information that is stored or transmitted electronically. Organisations have to implement mechanisms to ensure the confidentiality, integrity, and availability of all electronic protected health information in its creation, reception or transmission. Therefore, the implemented technical mechanism should protect networks, computers and other electronic devices from security threats.

4. Privacy Rule. This rule protects the privacy of the individuals that could be identified through their health information held by health organisation, regardless of whether the information is, or has been, in electronic form. For this reason organisations should [6]:

    - Give patients rights to access their medical records, restrict access by others, request changes, and to learn how they have been accessed.
    - Restrict most disclosures of protected health information to the minimum needed for healthcare treatment and business operations.
    - Enable patients to decide if they will authorize disclosure of their health information for uses other than treatment or healthcare business operations.

The frameworks discussed in this chapter demonstrate that governments have acknowledged the importance of protecting personal information. Furthermore, if we compare the DPA, Safe Harbor and HIPAA, we can see significant similarities. All of them state that the purpose for collecting and processing personal information needs to be stated, and then define legal ways in which this information can be processed. However, what is lacking in all three of them, is an effective and concrete means of enforcing these rules. As was noted in Chapter 1, what is missing is *processing transparency*, which makes it possible for an auditor to investigate how information was managed, and determine whether this was in compliance with the rules. In this thesis, we argue that provenance is an effective way of accomplishing this. To that end, in Chapter 3, we study the DPA in

more detail, and translate its principles into a set of Auditing Requirements—concrete conditions for compliance with the DPA which operate directly on collected provenance. Then, in Chapter 5 we develop algorithms for verifying these Auditing Requirements.

2.5 Summary

In this chapter, we presented an overview of related work. First, we studied the state of the art in provenance using the provenance life cycle of recording, storage, querying and analysis. Specifically, we identified PASOA as the primary approach for recording, storing and querying provenance. The reason for this is that PASOA is the only approach currently in existence that records causal relations between data that can be contained in messages, which is an essential requirement of achieving processing transparency in open systems. We also discussed in detail the PrIMe methodology, an effective tool for deciding which provenance needs to be collected and which level of granularity is appropriate. This methodology is used in the development of the application presented in Chapter 4. Moreover, in terms of the final stage of the life cycle (analysis), we concluded that, whilst provenance has been used for many purposes, it has not been used for verifying whether data processing was performed in compliance with the law, which is one of the primary objectives of this thesis.

Second, we studied audit trails, an alternative technique for collecting electronic evidence of the use of personal information. However, the main problem with this approach is that it records the actions that a system or user performs ordered by time. As such, it does not record the aforementioned essential relations between data and the processes that were applied to it. Moreover, it is a technology and domain dependent technique, making it less suitable for use in open and distributed systems. Finally, there exists no principled methodology for identifying which actions need to be recorded, leading to a large amount of data, which makes the analysis particularly challenging. In light of this, we concluded that provenance is the superior technique for our purposes.

Third, we argued the need for securing electronic evidence, provenance or otherwise. We discussed the key requirements of security and showed the importance of being able to verify that a system meets these requirements. To this end, we discussed UMLsec, an approach that allows us to include, formalise and verify security properties in UML models. Ultimately, the main reason to model security in our work is to obtain secure audit results: if we can guarantee that the transportation and storage of information is secure, the results obtained from an audit can be made trustworthy and reliable. The application of UMLsec to the Provenance-based Auditing Architecture defined in Chapter 4 is presented in Chapter 6.

Finally, we discussed three important data protection legislations, and showed that the importance of protecting personal information is acknowledged by governments. These

legislations were created to mandate the requirements under which the processing of personal data in computer systems must be carried out. Therefore, if an authority needs to verify the compliance of some computational processing to a legislative framework through audits, it is desirable to have tools that allow this authority to perform such analysis. For this reason, in the following chapters we present an approach that can be used to audit the processing of personal information in computational systems and verify its compliance with the law.

Chapter 3

Problem Definition

In the introduction of this thesis we highlighted the importance of verifying that personal information was processed correctly and legally. Furthermore, we discussed various types of legislation created by governments and public institutions to address this problem, and specifically focused on the Data Protection Act.

In this chapter, we define the central problem we address in this thesis. In order to do this, we analyse a case study of the Data Protection Act as a model of how legislation is used to protect personal information. The reason for focusing on this specific legislative framework, is that the Data Protection Act (DPA) is the main framework enforcing information privacy in the UK. Moreover, the DPA consists of principles that are well-specified and can therefore be made amenable to computational implementation.

Against this background, the contribution of this chapter is the identification of a set of requirements derived from this case study. The remainder of this thesis addresses the central challenge of automatically verifying these requirements. To accomplish this, we identify computational security techniques and provenance as a key means of accomplishing this.

The remainder of this chapter is organised as follows. First, we present an exemplar scenario in which the use of the DPA is demonstrated, and which will serve as a running example throughout this thesis. Then, based on the insights gained from this scenario, in Section 3.2, we analyse the DPA notification process and identify the need for performing audits on systems that manage personal information. Next, in Section 3.3, we identify the key application processes that involve the processing of personal information, and, as such, are subject to the DPA. In Section 3.4, we proceed by analysing the main principles of the DPA (see Section 2.4.1) to derive requirements for lawful processing of data, which can be automatically verified. Then, in Section 3.5, we argue that the availability of provenance of data processing is necessary to achieve this. In Section 3.6 we identify the key assumptions under which our algorithms and techniques perform this verification, and provide concluding remarks in Section 3.7.

3.1 Exemplar Scenario

This section presents a practical scenario in which personal information is being processed, and, as a result, to which the DPA legislation needs to be applied. In this example, we focus on the way personal information is processed and how it can be misused by the organisations that are collecting it. The scenario exposes the ways in which personal information can be captured and misused in an on-line e-commerce application. The scenario will later help us to derive various requirements, and is be used as a running example in subsequent chapters.

3.1.1 On-line Sales Scenario

Consider the following scenario. Alice is trying to start a family, and has decided to take a fertility treatment (advised by her doctor who gave her the corresponding prescription). She decides to buy her treatment from an on-line pharmacy. In order to get her treatment, she needs to register by providing her *name*, *address*, *date of birth*, *gender* and *national insurance number*. At the same time, but unrelated to her attempt to get pregnant, she applies for a job at the same pharmacy—but her application is turned down. She suspects that the pharmacy may have accessed her ordering history and realised that she has plans to start a family. As a result, the company has marked her as a high risk employee because of potentially expensive maternity costs. If this is true, the company obviously misused Alice's personal information; when she provided her personal information to the pharmacy, she did this with the purpose of purchasing her treatment. From the point of view of the company, the purpose is "on-line sales". For this purpose, the pharmacy is allowed verify the existence of the medicine, charge the amount to her card, send the medicine to her home and even create a record of the product's sale. Each of these tasks uses a different set of collected data. For example, the company can create a report of the monthly sales, which includes the *medicine's name* and the *quantity sold*. However, such a report cannot contain the name of the people that bought that item, since this might allow to each person to be identified and linked to a specific medication and, therefore, to a specific medical condition.

In this case, the DPA principles should also be followed. The pharmacy should process its clients' information for the purpose that was initially stated and not use more information than necessary. To verify that the pharmacy is following these rules, audits on the performed processes can be carried out.

3.1.2 Scenario Discussion

According to the DPA legislation (presented in Section 2.4.1), in the on-line sales scenario the pharmacy is a Data Controller and the customers, including Alice, are Data Subjects. The Data Processor is the internal stock management department.

In this example, the Data Subjects' personal information should be processed following the DPA principles. In order to verify whether such principles are observed, we need to expose the way in which this information was processed, i.e. make the processing of information *transparent*. When this is achieved, it becomes possible to analyse this processing to decide whether organisations are in compliance with the DPA principles. Such an analysis is called an *audit* and was described in Section 2.2.

To be in compliance with the DPA, organisations shall follow all of its principles. Each of these principles states different requirements, which are derived in Section 3.4.

3.2 DPA Notification Process

The example in the previous section highlights the importance of conducting audits to verify the compliance of information processing with the DPA principles. In the DPA case, audits could be conducted by the Information Commissioner's Office (ICO) [111], which is the UK's independent public body established to protect its citizens' personal information. The ICO has legal powers to ensure that organisations comply with the requirements of the Data Protection Act. Some of these legal powers include:

- Conducting assessments to check that organisations are in compliance with the Act.
- Conducting audits to assess whether organisations' processing of personal data follows good practice.

To be able to perform such audits, it is necessary to specify which processes are allowed to be performed on which data. This specification is later used to verify that applications are indeed correctly and lawfully carried out using personal data. Consequently, the DPA requires that every Data Controller that is processing personal information in an automated form should notify the ICO of the details of such processing through the *Notification Process* [7]. This process consists of the creation of a *register entry* describing the processing that each Data Controller is performing. This register entry is part of a public register of Data Controllers maintained by the ICO. The ICO's register has a well-defined structure categorising the purposes for collecting personal data from Data Subjects, as well as the Data Subjects of each Data Controller. Figure 3.1 shows the general structure of a register entry and Figure 3.2 is the application of this structure

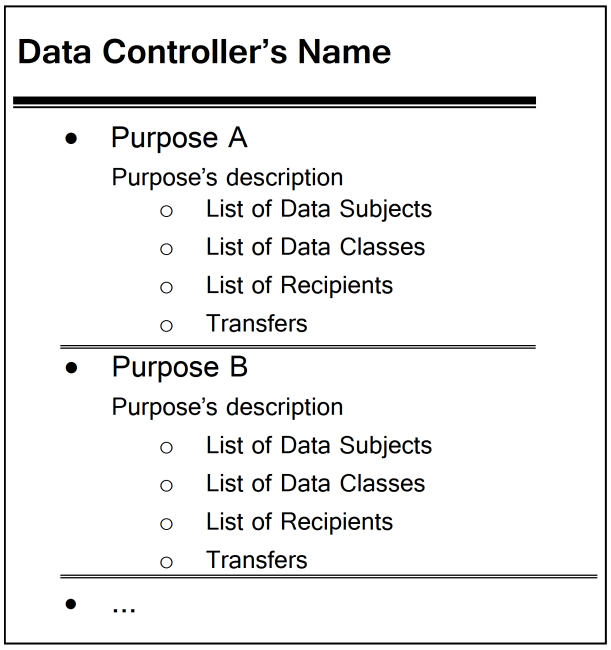


FIGURE 3.1: General Structure of an ICO's register entry

to the On-line Sales Scenario. The ICO's register for an existing UK pharmacy can be found in [88].

Each entry in the ICO's register is organised according to the purposes of a single Data Controller. Each purpose has a description that could include specific tasks related to it. It also includes a list of Data Subjects, Data Classes (types of personal information that are to be processed), and Recipients (individuals or organisations with whom the Data Controller intends or may wish to share data) applicable to the purpose. Data Subjects, Data Classes and Recipients have a numbered classification, as is shown in the example of Figure 3.2. The complete classification can be found in [111]. Further on, this classification model is used to assess whether organisations processed personal data according to the DPA.

The ICO's register can later be used in an audit to verify that the purposes, personal information and Data Subjects that the Data Controller has declared are used according to the DPA principles.

3.3 Processing Personal Data

By analysing the practical scenario described in Section 3.1, we can see that patterns exist in the communication between the Data Controllers who request personal information, and the Data Subjects who provide it. In this section, we identify and generalise these patterns in order to model them as a computational application, which will later form the foundation for the communication protocols we develop in the next chapter.

Pharmacy Example

- Purpose A: Accounts & Records (including on-line sales)
- Purpose's description: Keeping accounts related to any business or other activity carried on by the data controller, or keeping records of purchases, sales or other transactions for the purpose of ensuring that the requisite payments and deliveries are made or services provided by him or to him in respect of those transactions, or for the purpose of making financial or management forecasts to assist him in the conduct of any such business or activity
  - Data Subjects:
    - S101 Customers and clients
    - S102 Suppliers
    - S104 Complainants, correspondents and enquirers
  - List of Data Classes
    - C200 Personal Details
    - C204 Financial Details
    - C205 Goods or Services Provided
  - List of Recipients
    - R400 Data Subjects themselves
    - R405 Business associates and other professional advisers
    - R407 Other companies in the same group as the data controller
    - R408 Suppliers, providers of goods or services
  - Transfers
    - None outside the European Economic Area

FIGURE 3.2: Register entry of the On-line Sales Scenario

As the DPA states, there exist three main entities: Data Controller (Definition 2.7), Data Subject (Definition 2.8) and Data Processor (Definition 2.9). In practice, Data Subjects typically make initial contact with Data Controllers. However, this communication is not modelled because it contains no personal information. As a consequence, we assume that the Data Controller initiates the communication by requesting personal information from Data Subjects. We can model this communication using the following workflow:

1. The Data Controller sends a request for personal information to a Data Subject indicating which information it requires and the purpose for which such information is to be collected.

2. The Data Subject verifies the purpose and, if the purpose is accepted, the Data Subject sends the requested data.

3. The Data Controller receives the personal information and stores it in a local database. At this point, the information should be processed according to the

stated purpose. The Data Controller could perform the processing or outsource it to a Data Processor.

4. The Data Controller sends a request for processing to a Data Processor indicating the task to be performed and sends the information to be processed.

5. The Data Processor receives the information, performs the task and returns the result of the processing to the Data Controller.

In step 4, in order to obtain the necessary information to perform the indicated task, the Data Controller can grant access to the local database to the Data Processor. However, without loss of generality, we assume that the Data Controller explicitly sends a set of data that needs to be processed by the Data Processor. Equivalently, this data could be a reference to a database with the necessary credentials to access to it.

In this step-by-step explanation, two different stages can be identified. The first involves a request for personal information (steps 1, 2 and 3), which we call *data request*. The second involves the processing of information (steps 4 and 5), which we call *task request*.

Using this workflow description, we design a protocol that represents the communication between the three main entities. This protocol is presented in Figure 3.3 in which entities are represented by squares and the communication between them by arrows, which are labelled using the same numbering as above.

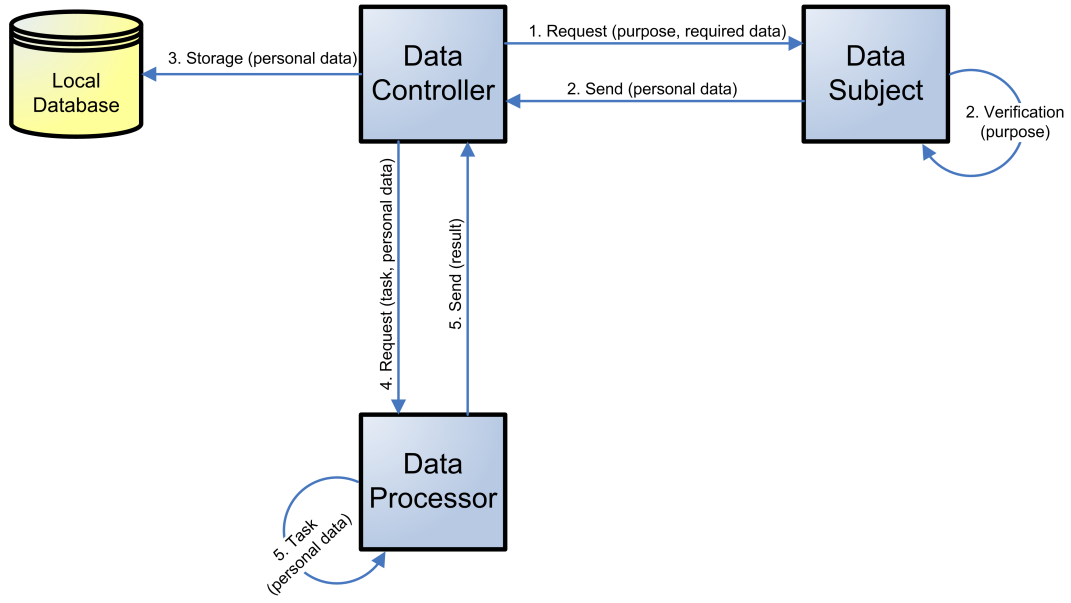


FIGURE 3.3: The *Data Request* and *Task Request* protocols

A different process that is not modelled by the protocol in Figure 3.3, but is mentioned in the DPA, is the update process, which we call *data update*. In this process, the Data Controller has already collected personal information from a Data Subject, but this Data Subject wants to modify some of its personal details (e.g. address). In practice,

Data Subjects inform Data Controllers of their wish using a message which contains no personal information. Thus, again we assume that Data Controllers initiate the update process, which is presented in Figure 3.4 and explained below.

1. The Data Controller sends a request for updating the Data Subject's personal information.
2. The Data Subject sends the updated information to the Data Controller.
3. The Data Controller receives this information and updates the Data Subject's personal information, which is stored in a local database.

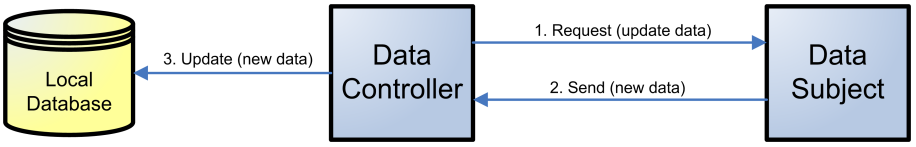


FIGURE 3.4: The *Data Update* protocol

3.4 Requirements Analysis

Thus far, we have presented communication protocols that generalise the communication patterns found in applications that manage personal data. To verify whether the DPA was correctly applied in these protocols, it is necessary to analyse the DPA's principles. In doing so, we can derive requirements that need to be satisfied for the processing of personal information to be in compliance with the DPA.

In this context, this section presents a requirements analysis of the DPA's eight principles, which were presented in Section 2.4.1.2. For each principle, we derive possibly multiple requirements that have to be satisfied, in order for the principle to be followed.

3.4.1 Principle 1: Personal Data Processed Fairly and Lawfully

Principle 1 is focused on the notion that data shall be collected and processed fairly and lawfully. Within the context of collection of information, these concepts are defined in terms of the stated purpose. However, this purpose only has meaning in the context of the identity of the Data Controller (DC). Thus, both the identity of the Data Controller and the reason for collecting the information must be known beforehand. This provides us with the first requirement for information processing:

Requirement. A [Authentication] In the first contact between a Data Subject (DS) and a DC, the DC should send its identity, the set of information requested and the purpose. DS verifies them and decides whether to accept the request.

The fair and lawful concepts are also applied to the processing of data, which is established by the initial purpose. In our case, information is processed lawfully if the purpose by which it was processed appears in the corresponding register file. Additionally, information is processed fairly if its processing is clearly understood by individuals who provide their personal information. The verification of fair and lawful processing is supported by Principle 2, for which we derive a corresponding requirement.

3.4.2 Principle 2: Legal Purpose

Principle 2 states that the DC may only request and process data from a DS for a legal purpose. When DS receives a DC request, DS has to verify the request and decide whether to send the required information. If DS decides to accept the request, then DS is giving his or her consent to process the information only for the given purpose. Thus, this information can only be processed according to the purpose agreed by DS. After processing the information, an auditor can verify that the processing conducted by a Data Processor (DP) was really that stated in the purpose. This provides us with the following auditing requirement:

Requirement. B [Purpose Compliance] For verifying that DC and DP used DS's information only for the stated purpose, we require an explicit description of the processing that was applied to such information. This description has to include which processes were applied and the purposes for which such processes were performed. This description can be analysed by auditors to determine whether these processes were compatible with the stated purposes. Moreover, auditors can verify whether the DC's purposes are legal and whether the ICO was notified.

3.4.3 Principle 3: Collection of Relevant Information

In Principle 3, the data requested from DS has to be relevant for the stated purpose; the DC should not request, collect or use more information than strictly necessary. Hence, when data processing is complete, an auditor should be able to verify that all the data captured and used by DC (or by DP, in the case of an outsourced processing) was relevant. Accordingly, we derive a new auditing requirement:

Requirement. C [Relevant Information Verification] For verifying that DC and DP requested and used only relevant information when processing data, we require a description of the information used to derive a specific result and a description of the information that should have been used. Then, an auditor can use both descriptions to determine whether the information that was actually used was indeed necessary to yield the required result.

3.4.4 Principle 4: Information Integrity

Principle 4 is focused on information integrity, which in the case of our model needs to be guaranteed during the stages of communication and storage of information. In order to support integrity of data transportation and data storage, we derive the following two requirements:

Requirement. D [Integrity] The integrity of information exchanged between the DC, DS, and DP should be maintained, as well as that of any description of the applied processes or information used. In so doing, we can guarantee that information sent by the main entities is not tampered with, that the descriptions represent what really happened to the information and that any unauthorized alteration of the exchanged information will be detected.

Requirement. E [Access Control] When the DS maintains the information in a storage component, which is managed by DC, we require that such a component implements access control measures to prevent undesired alterations. In so doing we can guarantee the integrity of this information.

3.4.5 Principle 5: Identification of Individuals

According to Principle 5, when collected information has been processed (once the initial purpose has been accomplished), it should not be possible to identify specific individuals by means of this information or any related processing result.

Thus, information that could identify a specific individual should not be present in the output of a process, i.e. processing results should be de-identified. De-identification of data can be achieved by removing information that could be used to identify an individual or by creating pseudo-data (see Section 2.3.1.6). For example, in the online sales scenario from Section 3.1, in order to hide Alice's identity, her name should not appear in an inventory. If for some reason, Alice's name needs to be present in the inventory, a sequence number can be used instead. This way, customers cannot be identified, but it is still possible to determine how processing results originated.

However, data de-identification provides no guarantee of anonymity, as processing results might contain other data, such as race, birth date, sex, and post code. This data can be linked to publicly available information to re-identify individuals [42]. To *effectively* anonymise data, various techniques, such as Differential Privacy [52], k-Anonymity [133, 75], ℓ-Diversity [93] and Perfect Privacy [97] can be used. To support Principle 5, and ensure personal data is not exposed, one of the previously mentioned techniques can be employed.

In the remainder of this thesis, we use the term anonymisation to refer to both anonymisation and de-identification, since they are two different techniques with the same goal,

i.e. to avoid the identification of individuals through their personal information. This leads to the following requirement:

Requirement. F [Anonymity Preservation] To verify that personal information cannot be used to identify a specific individual by a third party, we require an explicit description of the data used to generate processing results. An auditor can later analyse this description to check whether personal personal data was correctly anonymised. By anonymising personal data, links to individuals can be hidden so that they cannot be identified, while ensuring the link between data and process is maintained.

3.4.6 Principle 6: Rights of Data Subjects

According to Principle 6, personal data processing should be performed in conformity with the rights of data subjects. These rights are listed below. For each of them, we explain how they are already supported by the protocols from Section 3.3 or by the requirements above.

- The first right, access to personal data, states that a DS should have access to his or her personal data. This data is stored by a DC in a storage component, as the local database presented in Figure 3.3. This component should implement access control techniques and the corresponding interfaces to grant users access to their personal data.
- The second right, prevention of processing likely to cause damage or distress, is part of the authorization given by DS to process its data for the purpose given by DC. The DS decides if the process can cause some kind of damage or distress to him or her and then decides whether or not to give consent to process the information.
- The third and fourth rights state that personal information requested from the DS should not be used for direct marketing or automated decision-taking. This can be seen as a set of processes that cannot be applied to the collected data and, therefore, be part of Requirement B.
- The fifth right, rectification, blocking, erasure, and destruction, is a list of actions that DC can perform on the information at DS's request. The rectification process is part of the update process: DS rectifies the information and, in some cases, updates it. This process is modelled in Figure 3.4. The blocking, erasure, and destruction processes are carried out in order to avoid DS's personal information from being used. The first one refers to blocking access to DS's personal information, which is maintained in a storage component. The second and third refer to the deletion of DS's personal information from the storage component. From an audit point of view, deleting information affects the data gathering process, which,

in turn, affects the quality of the audit result. In order to solve this problem, we assume that instead of deleting information, the techniques explained in Requirement F are used. In this way, DS's identity is safe and the information can still be used to perform the necessary audits.

- The sixth right states the possibility of obtaining compensation if the rules stated in the DPA are not followed. Whilst this is an important issue, this is not computationally verifiable, and is therefore outside the scope of our work.
- The seventh right, request for assessment, mandates the ability to perform audits. These audits are carried out by an authorized entity to verify the correctness of the information processing and report the results. As mentioned in Section 3.2, within the context of DPA, this entity is the ICO.

3.4.7 Principle 7: Secure Management of Personal Information

Principle 7 states that a DC has to offer technical and organizational measures for managing the data of a DS. For example, this can be achieved by implementing the set of controls comprising the best practice in information security, as defined in ISO/IEC-17799 [11]. However, the measures defined in this ISO standard are beyond the scope of our work. Instead, we focus on the technical measures, specifically on the measures that relate to the information security properties defined in ISO/IEC-7498-2 [3]: confidentiality, integrity, authentication, non-repudiation and access control. These properties are offered by various security techniques that can encrypt and decrypt information, sign messages and verify signatures. These techniques were discussed in Section 2.3. Thus, within the context of this thesis, this principle states that auditors should be able to verify that entities implement information security techniques that offer the five aforementioned properties. This is captured in the next requirement:

Requirement. G [Basic Security Characteristics Verification] For verifying that DS, DC and DP implement security techniques in their communication, we require an explicit description of the techniques that were applied to data. This description can be analysed by auditors to verify that all necessary security techniques were used during information processing.

3.4.8 Principle 8: Overseas Information Transfer

Finally, Principle 8 states that the personal information of a DS cannot be transferred to countries not listed in the list of secure countries defined by the DPA. Thus, to verify this, we derive the next requirement.

Requirement. H [Information Transferred to a Secure Country] DS, DC and DP cannot transfer personal information to a country that is not included in the list of secure countries, which is defined in the DPA.

3.4.9 Requirements Discussion

Table 3.1 summarises the requirements we derived from the principles of the DPA. Most of the principles are supported by a single requirement except Principle 4, which requires two security characteristics, and Principle 6, which supports the seven rights of Data Subjects.

Principle	Requirement	Auditing Requirement	Security Requirement	Chapter
1	A – Authentication		X	6
2	B – Purpose Compliance	X		5
3	C – Relevant Information Verification	X		5
4	D – Integrity E – Access Control		X	6
5	F – Anonymisation Preservation	X	X	5
6	B – Purpose Compliance F – Anonymisation Preservation	X	X	5
7	G – Basic Security Characteristics Verification	X	X	5 and 6
8	H – Information Transferred to a Secure Country	X		5

Table 3.1: Requirements

In order to implement these requirements, we can classify them into two groups. The first group contains requirements for auditing the processing of personal data and includes Requirements B, C, F, G and H. To verify that these requirements are met, an explicit description of this processing is required. This description can later be audited to verify whether these requirements were implemented. For this reason, we call the requirements in this group "Auditing Requirements". In Chapter 5, we develop algorithms for automatically verifying that data processing satisfied these requirements.

The second group contains requirements related to security properties and includes Requirements A, D, E, F and G. These require the implementation of security protocols or cryptographic techniques, so we refer to these as "Security Requirements". These requirements guarantee the authentication, confidentiality, integrity, non-repudiation, access control and anonymity preservation properties. The different ways in which these properties can be implemented were discussed in Section 2.3. In Chapter 4, we develop an architecture that exhibits these properties.

Note that Requirements F and G belong to both groups. The reason for this is that they support characteristics of both groups. Requirement F is used to verify that personal data was correctly anonymised. To do so, we require an explicit description of past

processing. Hence, Requirement F is an Auditing Requirement. However, this requirement also states that data needs to be anonymised by implementing an anonymisation technique, making this requirement a Security Requirement as well. This requirement and its implementation is discussed in more detail in Chapter 5.

Similarly, Requirement G is used to verify that DPA entities implement security techniques in their communication. To do so, it also requires an explicit description of such techniques. Thus, Requirement G is an Auditing Requirement. At the same time, the requirement states that the technical means to support the implementation of such security techniques should be provided. Thus, Requirement G is also a Security Requirement.

3.5 Provenance as a Solution

As discussed in the previous section, the Auditing Requirements need an explicit description of past processing. Using this description, it becomes possible to perform audits and verify whether these requirements were met during the processing of personal information. To obtain such a description, the provenance of data and processing needs to be available.

The PASOA (Provenance Aware Service Oriented Architecture) model (see Section 2.1.2.2) was designed in the context of service-oriented architectures relying on individual services to record their own provenance. This model proposes to capture assertions that encode the relationships between the represented services and data in a generic, customizable and scalable way [63]. PASOA also incorporates a set of security properties [135], as well as a software engineering technique, called PrIMe [100], for making applications provenance-aware.

The PASOA model offers some desirable characteristics that can be useful to support the Auditing Requirements. These characteristics are listed below [63].

- PASOA is platform, domain and technology independent.
- The query functionality offered by this model supports multiple levels of abstraction allowing for the creation of user-tailored queries [98]. This is a crucial characteristic that allows us to audit all the requirements we have derived above.
- The provenance DAG representation can be easily used to analyse past processing against the Auditing Requirements.

By adopting this provenance model, we can process documentation as the explicit description of past processing, which is required to satisfy the Auditing Requirements. Provenance documentation is recorded at execution time, and queried after the processing of personal data has finished. This provenance information, which can be represented

as a DAG, can be analysed to decide whether the application satisfied the Auditing Requirements.

3.6 Assumptions

In order to use the provenance model in the protocols defined in Section 3.3, we need to define a set of assumptions under which the solutions we develop in the remainder of this thesis operate. These assumptions are described below.

1. *Provenance information cannot be deleted.* We assume the provenance information cannot be partially or totally deleted. Thus, when the provenance of a piece of data is obtained, complete information about past executions is always available.

2. *Provenance is persistent and available.* We assume provenance information is persistently stored in a Provenance Store component and is accessible at all times. Consequently, provenance information is accessible when required.

3. *Data and provenance are securely stored.* We assume that all the Local Databases, which contain application data, and the Provenance Store, which contain provenance information, implement the necessary measures to protect the stored information from unauthorised access by means of access control.

4. *Provenance is recorded truthfully.* All actors involved in the creation of provenance record what really happened at execution time. This can be achieved by using a secured library that automatically records the assertions that are made by actors.

5. *Only relevant provenance is recorded.* The actors involved in the processing of information only record information required to verify the Auditing Requirements.

6. *It should be possible to query the provenance store.* We assume the provenance store has a query functionality that can be used to retrieve the provenance of a specific item and supports scoped-provenance queries (see Section 2.1.2.2).[1]

7. *There exists an ontology that defines the way personal information is processed.* We assume that there exists a well-defined ontology that defines personal information and the processes that operate on it. This ontology should be used during the capture of provenance information, and should be used by all the entities involved. Based on this assumption, our solutions can operate in the knowledge that entities consistently create, query and reason about provenance.

8. *Audits are performed off-line.* Audits are performed after the execution of the processes over personal information finishes. Otherwise, it is not possible to analyse the complete processing that was performed on personal information.

[1] This is in contrast with some of the techniques described in 2.1.3, which do not do this.

3.7 Conclusions

In this chapter, we formulated the central problem that we address in this thesis: the automatic verification of correct processing of personal data. To do this, we first introduced an exemplar scenario that illustrates the application of the DPA. We analysed this scenario to demonstrate that performing audits allows us to decide whether data processing is in compliance with DPA principles. We also identified the key protocols that are used by the three main entities of an application that manages personal data: the Data Subject (DS), the Data Controller (DC) and the Data Processor (DP). Based on these protocols and the DPA principles discussed in Section 2.4.1, we identified the requirements for valid and lawful processing of personal information. We grouped these requirements into Auditing Requirements (which can be verified by auditing provenance) and Security Requirements (which require the implementation of security protocols or cryptographic techniques). Then, we discussed the importance of provenance for automatically verifying that these requirements are met. Finally, we identified the underlying assumptions of the solutions we develop in the upcoming chapters.

More specifically, in the following chapters, we address different aspects of the problem formulated here. In Chapter 4, we lay the foundations by creating an architecture for collecting the required provenance, while at the same time securing the exchange of application data. By so doing, we satisfy Security Requirements A (Authentication) and G (Basic Security Characteristics). Then, in Chapter 5, we develop algorithms for automatically verifying the Auditing Requirements. In Chapter 6, we secure the architecture from Chapter 4, including the complete life cycle of provenance—collection, communication, storage and analysis. By doing so, we satisfy the remainder of the Security Requirements. Finally, in Chapter 7 we methodically analyse the security of the complete architecture and verify that we have indeed satisfied all requirements.

Chapter 4

Provenance-Based Auditing Architecture

In the previous chapter, we analysed the principles of the DPA and derived a corresponding set of requirements. These requirements are grouped into Auditing Requirements and Security Requirements. Ultimately, the goal of the thesis is to automatically verify whether data processing has satisfied these requirements and, by so doing, determine whether this processing was in compliance with the DPA.

In this chapter, we take the first step towards this goal. This step involves the creation of the *Provenance-based Auditing Architecture*, which is the contribution of this chapter. This architecture captures and stores the provenance that is later used by the algorithms we develop in the next chapter to perform automatic verification of the Auditing Requirements. At the same time, this architecture secures the communication between its main entities. By doing so, it satisfies Security Requirements A (Authentication) and G (Basic Security Characteristics). In Chapter 6, we secure the remainder of the architecture (including the storage and analysis of provenance) to satisfy the other Security Requirements.

Now, to identify which provenance needs to be captured by the Provenance-based Auditing Architecture, we use the PrIMe methodology (see Section 2.1.2.4). PrIMe allows us to make this architecture *provenance-aware* by deciding which assertions should be recorded during execution time. Recall from Section 2.1.2.2 that assertions document which operations were performed on data, and are the primary way of encoding provenance in the PASOA model (see Section 2.1.2.2) on which our work is based.

To capture the required provenance identified by PrIMe, we extend the Data Request (invoked when a Data Controller requests personal data from a Data Subject) and Task Request (invoked when a Data Controller wishes to delegate processing to a Data

Processor) protocols (see Section 3.3). Furthermore, we define a new protocol, Query Request, that allows an auditor to retrieve the provenance from the architecture.

Since the role of auditor will be played by the algorithms we develop in the next chapter, provenance needs to be machine accessible. Thus, the final step we perform in this chapter is to define the provenance queries, the result of which forms the input of the algorithms in Chapter 5. These answers take the form of provenance DAGs (see Section 2.1.4), which are particularly suitable for automatic analysis.

Thus, in more detail, we perform the following steps. First, in Section 4.1, we present the Provenance-based Auditing Architecture. Then, in Section 4.2, we apply the PrIMe methodology to expose the assertions that need to be recorded. Next, in Section 4.3, we modify the Data Request and Task Request protocols to capture provenance, and define the Query Request protocol. Using this protocol, the algorithms in Chapter 5 can retrieve the necessary provenance. Thus, in Section 4.4, we describe how this is achieved, by defining the necessary provenance queries. Finally, in Section 4.5 we discuss the contributions made in this chapter in the context of existing work, and conclude in Section 4.6.

4.1 Building the Architecture

In this section, we develop the Provenance-based Auditing Architecture. First, we describe its components in Section 4.1.1 using concepts from UML. Then, in Section 4.1.2 we explain how these components interact, i.e. how the architecture operates. Finally, in Section 4.1.3, we secure the communication between its components.

4.1.1 Components

The architecture is based on the Data Request and Task Request protocols presented in Section 3.3. Consequently, the entities that exist within these protocols also appear in the architecture. However, the main difference between these protocols and the ones that exist in the architecture, is that the latter records provenance. Therefore, it is necessary to include new components, which essentially make our architecture provenance-aware.

The architecture is presented as a UML use case diagram in Figure 4.1, in which the Data Subject, the Data Controller and the Data Processor are modelled as actors. Two new actors are introduced—the Provenance Store and the Auditor actors. Furthermore, the Data Request and Task Request protocols are modelled as use cases. Again, we add new use cases, which are invoked by the Auditor to verify Auditing Requirements B, C, F, G and H (Section 3.4).

In what follows, we describe the actors and use cases in further detail.

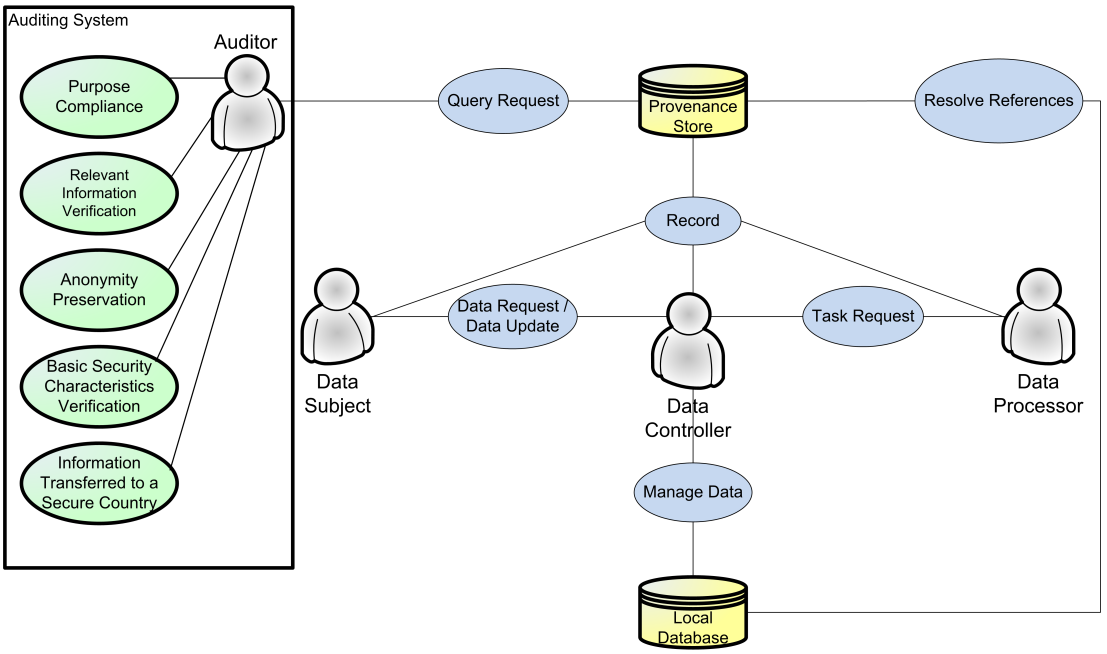


FIGURE 4.1: Provenance-Based Auditing Architecture

4.1.1.1 Actors

The main actors in the architecture are:

The `Data Subject` actor is a software proxy representing a Data Subject that communicates with a Data Controller on behalf of an individual.

The `Data Controller` actor represents a Data Controller that collects personal information from Data Subjects for a specific purpose.

The `Data Processor` actor represents a Data Processor that processes information on behalf of a Data Controller.

The `Auditor` actor represents an auditor who verifies whether the processing of the Data Subject's information was performed in compliance with the auditing requirements derived in Chapter 3.

The `Local Database` subactor represents a local database in which the Data Controller stores the Data Subjects' information. We refer to this component as a subactor to show that is part of the Data Controller actor, who manages the information stored in it.

The `Provenance Store` actor represents the Provenance Store where provenance, in the form of p-assertions (See 2.1.2.2) are stored. This provenance is generated as a result of the interactions between Data Controllers, Data Subjects and Data Processors.

4.1.1.2 Use Cases and Requirements

The following use cases represent the protocols and processes in which the actors participate. These use cases are represented by blue ovals in Figure 4.1.

The `Data Request / Data Update` use case represents a request for personal information or a request for updating personal information issued by a Data Controller to a Data Subject. These use cases represent the protocols described in Figures 3.3 and 3.4.

The `Task Request` use case represents a request for delegating a task issued by a Data Controller to a Data Processor. This use case represents the interaction between the Data Controller and the Data Processor shown in Figure 3.3.

The `Record` use case represents the process of recording p-assertions.

The `Query Request` use case represents the process of querying p-assertions.

The `Resolve References` use case represents the process of resolving references to objects stored in the Local Database, which contains the actual data. As mentioned in Section 2.1.2.2, provenance information can contain references to data instead of a copy of the real data. If this data needs to be accessed, the Provenance Store presents the necessary credentials to the corresponding database, accesses the original data, and executes this use case.

The use cases presented as green ovals in Figure 4.1 represent the Auditing Requirements described in Chapter 3. By obtaining the necessary provenance and invoking the `Query Request`, auditors can verify whether actors who processed personal information were in compliance with these requirements.

4.1.2 Component Interactions

Now that we have presented the components of the Provenance-based Auditing Architecture separately, in this section we explain how the actors interact by participating in use cases, and by doing so, exploit the functionality of the architecture.

The `Data Subject` actor communicates with the `Data Controller` actor through the `Data Request` and the `Data Update` use case. The `Data Request` use case represents a request for personal information issued by the DC to the DS. The `Data Controller` requests information from the `Data Subject`, which is stored by the `Local Database` actor. The `Data Update` use case represents the updating process performed by DS: if the `Data Subject` wishes to change its personal information, the updated information is sent to the `Data Controller`.

The `Data Processor` actor communicates with the `Data Controller` using the `Task Request` use case, which represents a task delegated to DP by DC. The p-assertions generated by the `Data Subject`, `Data Controller` and `Data Processor` actors are recorded in the `Provenance Store`. This process is represented by the `Record` use case. Additionally, in the `Provenance Store`, references to the original data (stored in the `Local Database`) could be recorded instead of the information itself. Thus, the `Provenance Store` has a connection to the `Local Database` through the `Resolve References` use case. Note that if instead of a reference to data a copy to the actual data is maintained, then this use case is not necessary.

Finally, the `Auditor` actor, who is responsible for verifying the compliance of the principles mentioned in Chapter 3, participates in five use cases: `Purpose Compliance`, `Relevant Information Verification`, `Anonymity Preservation`, `Basic Security Characteristics Verification` and `Information Transferred to a Secure Country`. These use cases correspond to the Auditing Requirements (B, C, F, G and H) identified in Chapter 3. These requirements can be verified by querying the p-assertions stored in the `Provenance Store` after information processing has finished. For this reason, the `Auditor` is connected to the `Provenance Store` through the `Query Request` use case.

Thus, the information flow within the Provenance-based Auditing Architecture can be summarised in the following workflows:

Recording of provenance in which components make assertions related to the actions they perform and record them in a Provenance Store.

Storage of provenance in which assertions are persistently stored in a Provenance Store.

Querying of provenance in which process documentation is queried.

Analysis of provenance to determine whether the execution of the system is in compliance with the requirements. The result of this workflow is an audit report.

4.1.3 Securing the Architecture

As the analysis of the DPA principles (Section 3.4) illustrates, security is a key issue in a provenance-aware system. Without having the necessary security protocols in place, which prescribe how messages should be exchanged between actors, the correctness of the information produced, recorded, stored or queried can not be ensured.

In the architecture, communications between the actors can take place over an insecure network. This means that a third party can eavesdrop on the communications channel or even modify the content of messages without being detected. Hence, it is necessary to implement the security measures required by the Security Requirements (see Section

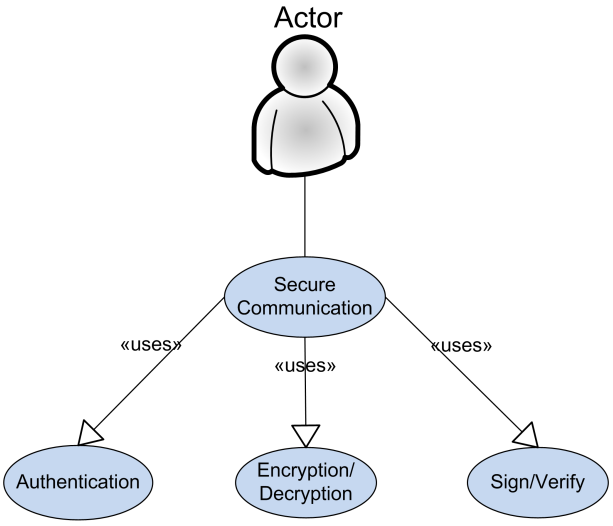


FIGURE 4.2: Securing the Provenance-Based Auditing Architecture. Here `Actor` can be any actor defined in Section 4.1.1.

3.4.9). Moreover, the communication between the actors and the Provenance Store also needs to be secured, in order to ensure that the same security guarantees apply to the assertions generated by actors.

In order to support the Security Requirements, we extend the architecture with a use case called `Secure Communication` (see Figure 4.2). This use case offers functionality that supports the required security characteristics. This functionality includes the authentication of entities (offered by the `Authentication` use case), the encryption of messages (offered by the `Encryption/Decryption` use case) and the signing of messages (offered by the `Signature/Verification` use case).

To establish secure communication between entities, we need to use a challenge-response identification protocol. A protocol of this type provides security over Internet communications preventing eavesdropping, tampering or message forgery. It allows entities to (mutually) authenticate each other through certificates and create session keys that are used to encrypt/decrypt the data contained in messages. In so doing, the confidentiality property is guaranteed to hold for the data that is exchanged. After a secure connection has been established, public and private keys can be created, which form the basis of a digital signature scheme. In this scheme, each message is signed before being sent and the signatures verified after being received. As a result, the properties of integrity, message authentication and non-repudiation are guaranteed to hold. As explained in assumption 3 (see Section 3.6), access control is not considered in this thesis. However, access control can be implemented using public certificates, i.e. an entity is granted to a resources after verifying its identity through the use of certificates. In light of our intention to present a technology independent approach, the specific technical and implementation details of cryptographic techniques are outside the scope of this work.

4.2 Identifying the Required Provenance

In the previous section, we developed the Provenance-Based Auditing Architecture that includes the Provenance Store component. This component is a key element in the implementation of provenance auditing functionality. We also explained how to secure the architecture by supporting the Security Requirements. The next step is to identify which provenance needs to be collected in order to verify compliance of the Auditing Requirements (Table 3.1).

As we discussed earlier, this provenance is encoded as p-assertions. To verify the Auditing Requirements, it is not necessary to record all possible assertions. Rather, we only need to record those that document the processes that are decisive in the use of information. Now, to decide which assertions should be recorded to verify the Auditing Requirements, we use the PrIMe methodology (see Section 2.1.2.4). This methodology consists of three phases:

- For each Auditing Requirement, identify which provenance-related questions need to be answered.
- Decompose the application into a set of actors that record provenance.
- Identify which information is required to answer the provenance questions related to the Security Requirements, but is lacking from the interactions between the actors identified in Phase 2.

In what follows, we go through each of these phases.

4.2.1 Phase 1: Provenance Question Capture and Analysis

In first phase of PrIMe, we must identify which provenance-related questions within the application need to be answered. Tables 4.1, 4.2, 4.3, 4.4 and 4.5 show these provenance questions, which are derived from the Auditing Requirements B, C, F, G and H. Each table presents a provenance question and its corresponding provenance query or queries. In each table, the data item and the scope (see Section 2.1.4) are defined to delimit the results from the provenance query.[1] Furthermore, we define a processing step, which transforms the result of the query into a definitive answer of the provenance question.

Before detailing each provenance-related question, we first define two types of data that we use in this section. The first type, *Collected data* is the information obtained from Data Subjects as a result of the `Data Request` use case. Second, *Used data* is the

[1] Later on, when we represent this query result as a DAG, the data item refers to a node whose ancestors we want to retrieve, and the scope to the type of edges (i.e. type of relationships).

information used in the `Task Request` use case (for delegating the processing of data to a `Data Processor`).

Table 4.1 summarises the provenance question related to **Requirement B**, *Purpose Compliance*, which verifies whether the results of data processing were compatible with the purpose for which data was collected. To answer this question, we need to know which data was used in the process, as well as the purpose for capturing the data. This data can then be compared to the established criteria related to the purpose. Thus, this comparison allows us to conclusively determine whether the audit requirement was observed.

Provenance Question	Was the result obtained by processing personal data compatible with the purpose for which it was captured?
Provenance Query	What was the *used data* to obtain the result, and the *purpose* for which the data was used?
Data item	Result
Scope	Requested and collected data
Processing step	Compare the used data with established criteria related to the purpose.

TABLE 4.1: Provenance related question for Requirement B, Purpose Compliance

Next, Table 4.2 relates to the provenance question that verifies whether all personal data captured from a DS was actually used by a process. Thus, this provenance question is related to **Requirement C**, *Relevant Information Verification*. For answering this provenance question, we need to know which data was used to obtain the result and which data was collected from the DS. Then, the used data can be compared with the collected data.

If these sets are not equal, we have two possibilities. The first possibility is that more personal data was collected than strictly necessary. This means there is some data that was collected but not used. The second possibility is that more data was required than collected. This data might be collected from other entities or might be produced by another process performed by the same entity. The details of these cases are analysed in Chapter 5.

Provenance Question	Was all the collected data used in the processing of result?
Provenance Query	What was the *used data* and the *collected data* used to obtain result?
Data item	Result
Scope	Requested and collected data
Processing step	Compare the used data with the collected data and highlight the differences.

TABLE 4.2: Provenance related question for Requirement C, Relevant Information Verification

Table 4.3 relates to the provenance question that checks whether personal information was anonymised, such that a third party cannot link individuals to their personal information. This question is related to **Requirement F**, *Anonymity Preservation*. To answer this provenance question, we need to know whether the result contains sensitive data, i.e. information that should not be exposed in a processing result. To verify this requirement, we assume that we have a list of data items considered sensitive (such as ethnic origin or medical history). This list can then be compared to data items contained in the result. If one or more sensitive items are found, we can conclude that the information was not properly anonymised.

Provenance Question	Were all used data and associated results properly anonymised?
Provenance Query	What was the produced *result*?
Data item	Result
Scope	Requested and Collected Data
Processing step	Verify that no sensitive items are present in the result

TABLE 4.3: Provenance related question of Requirement F, Anonymity Preservation

Table 4.4 presents the provenance question that checks whether messages exchanged between the entities were encrypted and signed cryptographically. This question relates to **Requirement G**, *Basic Security Characteristics Verification*. To answer this question we need to know which data was used to obtain the result and which security processes were applied to this data. Then, the identified security processes can subsequently be analysed to determine whether they ensure the properties of confidentiality, integrity, authentication and non-repudiation.

Provenance Question	Was all data used to obtain the result encrypted, signed, decrypted, verified, etc.?
Provenance Query	Which *data* was used, and which *security processes* were applied to it?
Data item	Result
Scope	Provenance of result with security processes
Processing step	Check that data was encrypted, decrypted, verified, signed, etc.

TABLE 4.4: Provenance related question of Requirement G, Basic Security Characteristics Verification

Finally, Table 4.5 relates to the provenance question that checks whether the collected information stayed inside countries listed as secure. This question is related to **Requirement H**, *Information Transferred to a Secure Country*. To answer this question we need to know whether collected data crossed a border. The list of countries through which personal information transited can then be compared with the ones listed by the DPA.

Now that we have identified the provenance questions, we can proceed to the next phase.

Provenance Question	Was collected data only sent to "secure countries"?
Provenance Query	What was the *collected data* and to which *countries* was it sent?
Data item	Collected Data
Scope	Provenance of Collected Data including countries it transited through
Processing step	Check if the countries appear in the secure countries list

TABLE 4.5: Provenance related question of Requirement H, Information Transferred to a Secure Country

4.2.2 Phase 2: Actor Based Decomposition

In the second phase of PrIMe, actor-based decomposition, we identify the interactions between actors and the messages they exchange. By recording the act of sending and receiving these messages, we can capture the provenance of the data processing that occurs within the system.

The result of this phase is the diagram shown in Figure 4.3. This figure presents a more detailed formalisation of the Data Request and Task Request protocols from Section 3.3. More specifically, it shows the messages that are exchanged between the main entities as well the *relationships* between these messages.

In this diagram, the main entities are represented by squares: the `DC` (Data Controller), the `DS` (Data Subject) and the `DP` (Data Processor). For the purpose of consistency, the labels of the messages in this diagram correspond to the ones in Figure 4.4, which shows the result of the third phase of the PrIMe methodology.

Now, the information flows in the Data Request and Task Request protocols can be modelled by the exchange of four messages: `M4`, `M5`, `M10` and `M11` (the solid arrows). In `M4`, `DC` sends the collection purpose to `DS`, who responds with message `M5`, which contains the requested data. These two messages are the main part of the Data Request protocol that represents the request of personal data. Then, using message `M10`, `DC` sends a set of data (which is a subset of the data contained in `M5`). This data is processed by `DP`, which informs `DC` of the result using `M11`. Thus, messages `M10` and `M11` are part of the Task Request protocol that formalises the process of outsourcing a job.

As mentioned earlier, Figure 4.3 also shows the *relationships* between these messages, denoted by `R4`, `R9`, `R10` and `R11` (the dashed-arrows). The meaning of these relations is as follows:

- Relation `R4` indicates that message `M5` *was Acquired For* the purpose conveyed in message `M4`.

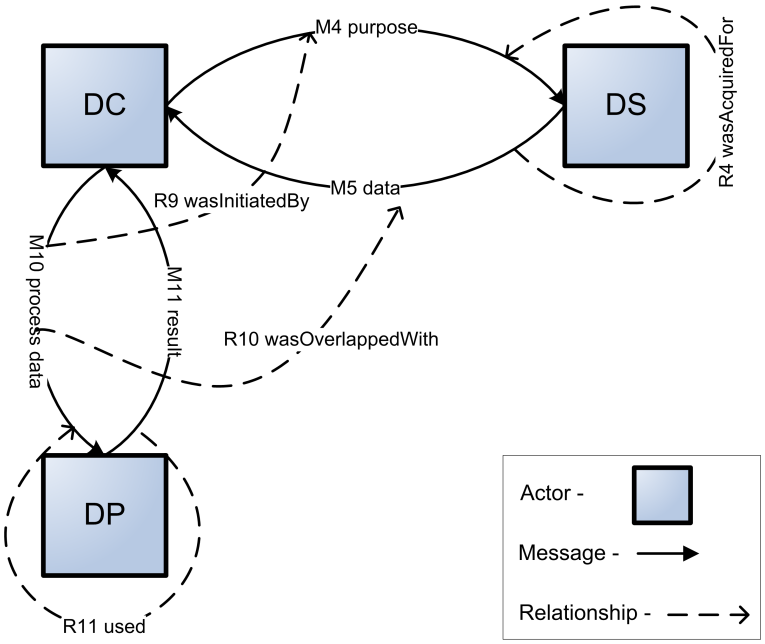


FIGURE 4.3: Phase 2 Diagram

- Relation `R9` indicates that the data processing requested by message `M10` *was Initiated By* the purpose in message `M4`.[2]
- Relation `R10` indicates that the content of `M10` (Process data) *was Overlapped With* the data contained in `M5`.
- Relation `R11` indicates that the content of `M11` (Result) was *used* data in `M10`.

So far, we have identified the key messages exchanged in the Data Request and Task Request protocols, as well as the relationships between these messages. Note that in Figure 4.3 messages related to security do not appear. Thus, as it stands, these protocols are not secure. In phase 3, we address this shortcoming.

4.2.3 Phase 3: Adapting the Application

To secure the messages and relationships identified in Phase 2, shown in Figure 4.3, we require additional information that is currently not conveyed by the messages in this figure. Therefore, in the third phase of PrIMe, we augment the protocols to make this information explicit.

As an example of missing information, the diagram created in Phase 2 (Figure 4.3) does not have explicit information about the security functions applied to data, which are

[2]Relationship `R9` cannot be directly created between messages `M4` and `M10`, as both messages are sent by `DC`. In practice, the `R9` is produced by creating an internal state related to `M4` that later is used to create a relationship with `M10`.

required to support the Security Requirements. This information is necessary to determine whether the Basic Security requirement hold (see Table 4.4 for the corresponding provenance question).

Now, in order to make this information available, we split each actor into a main actor and a subactor. These subactors are called `Secure Communication`, which make available the security functions supported by the `Secure Communication` use case (see Section 4.1.3 and Figure 4.2). Figure 4.4 shows the same protocol as Figure 4.3, but extends it with the `Secure Communication` subactors.

Again, messages are represented by solid arrows and the relations between them by dashed arrows. The local secure communication entity performs the four main security operations described in Figure 4.2: encryption, decryption, signature and verification. Thus, `Secure Communication` subactors associated with each main actor are called `DCSec`, `DSSec` and `DPSec`. This diagram also includes messages related to the implementation of a challenge-response authentication protocol, which is implemented to support Security Requirements A (Authentication) and G (Basic Security Characteristics).

As a result, in Figure 4.4, `M1`, `M2`, `M3` (between `DC` and `DS`), `M7` and `M8`, `M9` (between `DC` and `DP`) are part of the authentication protocol. The messages exchanged between the actors and their corresponding `Secure Communication` subactor represent the encryption, decryption, signing, and verification processes. These processes generate the relationships *verifiedDecrypted* to indicate the verification and decryption of a message, and *encryptedSigned* to indicate the encryption and signature of a message. Note that these messages are exchanged internally by the corresponding actor. Thus, the messages exchanged between the pair `DC`–`DS`, and the pair `DC`–`DP` are transported over a network (and encrypted and signed), while the messages exchanged between the actor and its `Secure Communication` subactor are not (for clarity, these messages, which are related to the encryption, decryption, signing, and verification, are not labelled).

When we compare Figure 4.4 to the protocol from phase 2 (shown in Figure 4.3), we can see that in the communication between `DC` and `DS`, four messages are added. These messages are `M1`, `M2` and `M3`, and `M6`. Of these, the first three implement an authentication protocol. The last one acknowledges the correct receipt of the data contained in message `M5` that was part of the original protocol from phase 2.

Besides these new messages, the extended protocol also contains four new relationships: `R1`, `R2`, and `R3`, and `R5`. The first three of these encode relationships between the messages exchanged by an authentication protocol. The fourth, `R5` encodes the relationship between message `M5` (which contains the data sent by `DS` to `DC`) and message `M6`, which acknowledges the receipt of this message.

As can be seen in Figure 4.4, similar messages and relationships are introduced to extend the protocol that exists between `DP` and `DC`

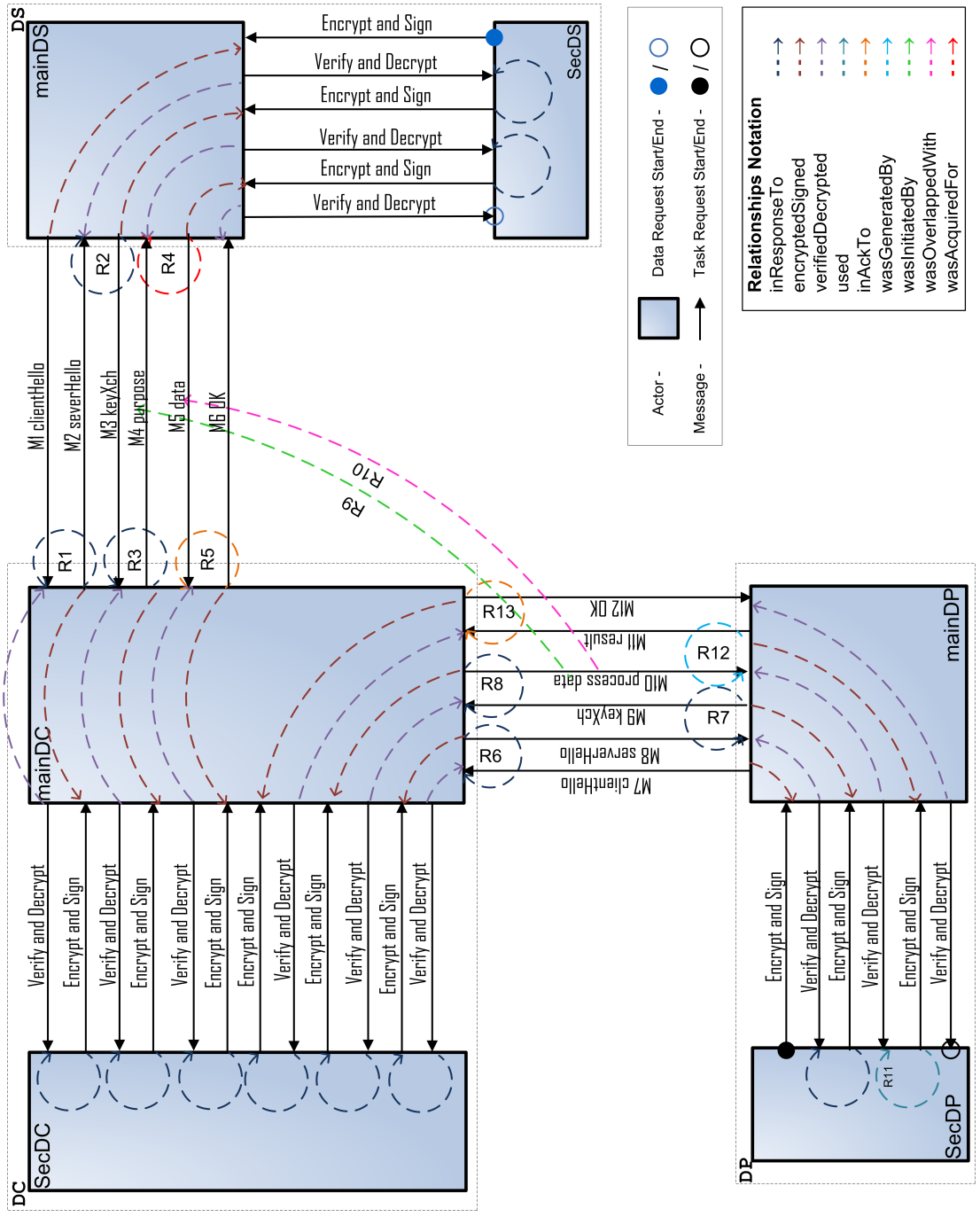


FIGURE 4.4: Phase 3 Diagram

This concludes the application of the PrIMe methodology to the Data Request and Task Request protocols we described in the previous chapter. Using PrIMe, we identified the messages exchanged in these protocols and the relationships that exist between these messages. In addition, we secured these protocols by introducing new `Secure Communication` subactors that implement the necessary basic security functions. In the next section, we show how provenance is recorded by creating assertions associated with the messages and relationships identified in this section.

4.3 Recording Provenance

At this point, we have defined the messages that are exchanged between the actors, and the relationships that exist between these messages. In this section, we show how provenance is extracted from the protocols identified in the previous section, and stored as *p-assertions* (see Section 2.1.2.2). In more detail, the provenance of the act of sending and receiving a message is captured as an *interaction p-assertion*, while a relationship is stored as a *relationship p-assertion*.

In more detail, note that the actual messages are not recorded in the provenance store, but the relationship p-assertions that refer to these messages are. These relationship p-assertions contain references to the data that was contained in the corresponding messages. We assume that each message contains a unique identifier, which is used in relationship p-assertions to refer to messages. Thus, given a message identifier, it is possible to find all relationship p-assertions that contain this identifier, and to retrieve a reference to the data that the original message contained. For more details on these data references, see Section 2.1.2.2 and Section 3.5 where we explain that the PASOA model used in this thesis is a process-based system that records and stores references to the data and the data process dependencies that constitute provenance.

To illustrate the process of recording these assertions, we create a UML sequence diagram for each of the use cases in the Provenance-based Auditing Architecture we defined in Section 4.1. These sequence diagrams model the `Data Request`, `Task Request` and `Query Request` communication protocols. We extend these protocols, which originally were defined as the interaction between two actors (e.g. between `Data Subject` and `Data Controller`, and between `Data Controller` and `Data Processor`), with the recording of provenance by invoking the `Record` use case. Recall from Section 4.1.2, that this use case enables these three main actors to store p-assertions in the `Provenance Store`.

In this section, our main focus is on how provenance is created and stored; securing these protocols will be the subject of Chapter 6. Therefore, in what follows, the messages related to the implementation of an authentication protocol and its technical details will be omitted from the diagrams.

First, we introduce the notation used in these diagrams. Then, we present the sequence diagrams themselves, one for each of the aforementioned protocols.

4.3.1 Notation

Within the sequence diagrams, two types of elements are exchanged: messages and p-assertions. Both contain application data—the data exchanged by the `Data Request`, `Task Request` and `Query Request` protocols (see Section 4.1). The notation used for this data is shown in Table 4.6.

	Actor			
Data	DS	DC	DP	AU
Purpose	purpose	purpose	-	-
Collected Data	data	data	-	-
Used Data	-	processData	processData	-
Task	-	-	task1	-
Processing Result	-	result	result	-
Ack Message	ok	ok	ok	ok
pquery Item	-	-	-	item
pquery Scope	-	-	-	scope
pquery Result	-	-	-	qresult

TABLE 4.6: Application Data

A message is a tuple consisting of application data and a unique identifier id_i:

$$\mathsf{message}(id_i, d) \tag{4.1}$$

The p-assertions can be grouped into two categories: interaction p-assertions and relationship p-assertions (See Section 2.1.2.2). The former records the act of receiving or sending a message. Therefore, it is a tuple of the identifier of the message it refers to, and a so-called *view*, which is either *receiver* or *sender*:

$$\mathsf{ipa}(id_i, view) \tag{4.2}$$

The latter records the relationship between interaction p-assertions. For example, a relationship p-assertion can express the fact that a message was sent in response to another. To encode this fact, relationship p-assertion contain the message identifiers id_i and id_j, the relationship rel between these messages, and the application data d to which the relationship pertains.[3]

$$\mathsf{rpa}(id_i, d, rel, id_j) \tag{4.3}$$

[3]Depending on the approach used to manage application data, d can be a reference to or a copy of the plain data contained in the message to which this assertion is related to.

In what follows, we make the Data Request, Task Request and Query Request protocols provenance-aware by recording interaction p-assertions and relationship p-assertions defined in Equations (4.2) and (4.3).

4.3.2 Recording Provenance in the Data Request Protocol

Recall from Section 4.1.1 that the first use case, Data Request, represents the process in which the Data Controller requests personal information from the Data Subject. This process is modelled in the sequence diagram presented in Figure 4.5 showing three objects: DS, DC, and PS, which are instances of the Data Subject, Data Controller and Provenance Store actors defined in Section 4.1.1.

The protocol modelled in this sequence diagram is the following: DC requests some personal information from DS for a given purpose, DS checks the purpose and, if DS accepts it, then it responds with the requested information. Then, when DC receives this information, DC acknowledges its receipt. At the same time, both actors record the p-assertions related to this process by communicating with PS.

4.3.2.1 Messages

In Figure 4.5, the messages exchanged between DS and DC are labelled Mi. These messages were defined in Section 4.2.3, and therefore are labelled with the same numbers as in Figure 4.4. For clarity, we repeat the meaning of each message, and indicate which application data they carry.

Message M4 contains *purpose*. With this message DC requests personal information from DS to be used for the indicated purpose. When M4 is received, the purpose is verified by DS, who responds with message M5 if the stated purpose is accepted. Message M5 contains *data*, the personal data requested. Upon receipt of this message, DC stores the data in a local database, and acknowledges the receipt to DS by sending M6.

4.3.2.2 Interaction p-assertions

This protocol is made provenance-aware by recording interaction p-assertions constructed according to Equation (4.2). These assertions are represented in the sequence diagram as Ii. In this diagram, assertions I7, I10 and I11 are related to messages M4, M5 and M6 and sent by DC and stored by PS upon receipt. Turning to assertions I8, I9 and I12, these denote the act of recording interaction p-assertions related to the same messages from the point of view of DS.

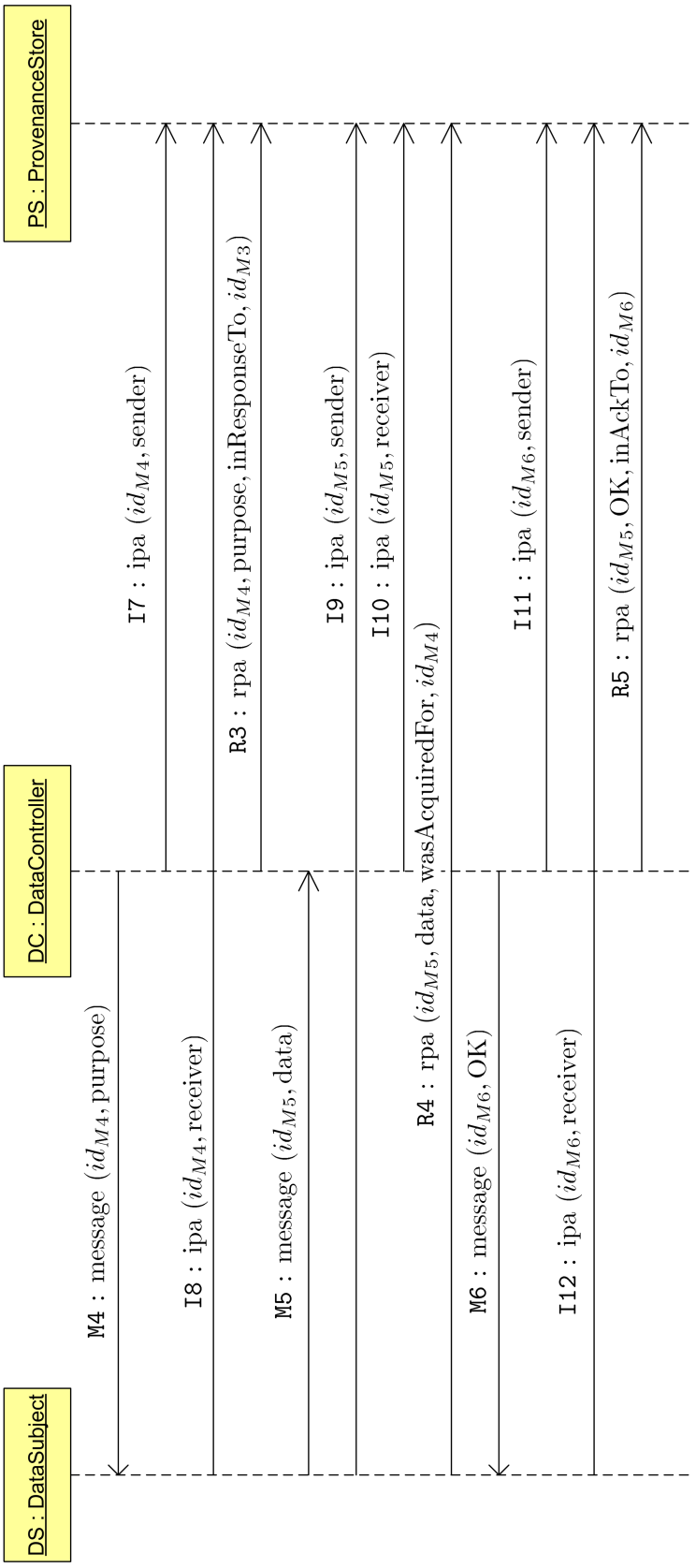


FIGURE 4.5: Data Request UML Sequence Diagram

4.3.2.3 Relationship p-assertions

The relationship p-assertions in this protocol are labelled by `Ri`. Using these assertions, the sender of a message records the relationship between two interaction assertions.

Messages `R3`, `R4` and `R5` denote the act of recording relationship p-assertion of previous messages in the Provenance Store.

Since an authentication protocol is executed before message `M4` is sent (not shown in this diagram), relation p-assertion `R3` creates a relationship indicating that `M4` was sent *in Response To* `M3`, which is the last message of the authentication protocol execution (again, this message is not shown in this diagram. See Figure 4.4). `R4` indicates that the `data` contained in `M5` *was Acquired For* the *purpose* contained in `M4`. Finally, `R5` indicates that `M6` was sent in acknowledgement to (*in Ack To*) `M5`.

Using the extensions made in this section to the Data Request protocol, we have made provenance available about the collection of personal information. In the next section, we will carry out a similar process for the processing of personal information.

4.3.3 Recording Provenance in the Task Request Protocol

The `Task Request` use case, discussed in Section 4.1.1, represents the protocol in which the `Data Controller` delegates the processing of information to the `Data Processor`. This process is modelled in the sequence diagram shown in Figure 4.6 containing the objects `DP`, `DC`, and `PS`, which are instances of `Data Processor`, `Data Controller` and `Provenance Store` actors.

This protocol is initiated by `DC`, who sends personal information that `DP` is requested to process. After receiving and processing this information, `DP` sends the result to `DC`. Finally, `DC` acknowledges the receipt of the result.

4.3.3.1 Messages

Again, the messages in Figure 4.6 are labelled `Mi`. Since we defined these messages in Section 4.2.3, they have the same labels as in Figure 4.4. For clarity, we repeat the meaning of each message, and indicate which application data is contained in them.

Message `M10` contains process data shown as *processData*, which is a data set of personal information to be processed by `DP`. After this is received, `DP` processes it using function task1 that outputs the *result*. The name of this function is made explicit, because the type of processing performed on the data is essential for determining whether the Auditing Requirements were satisfied. Using message `M11`, `DP` informs `DC` of the *result* of the processing. Upon receipt of this message, `DC` acknowledges its receipt using `M12`.

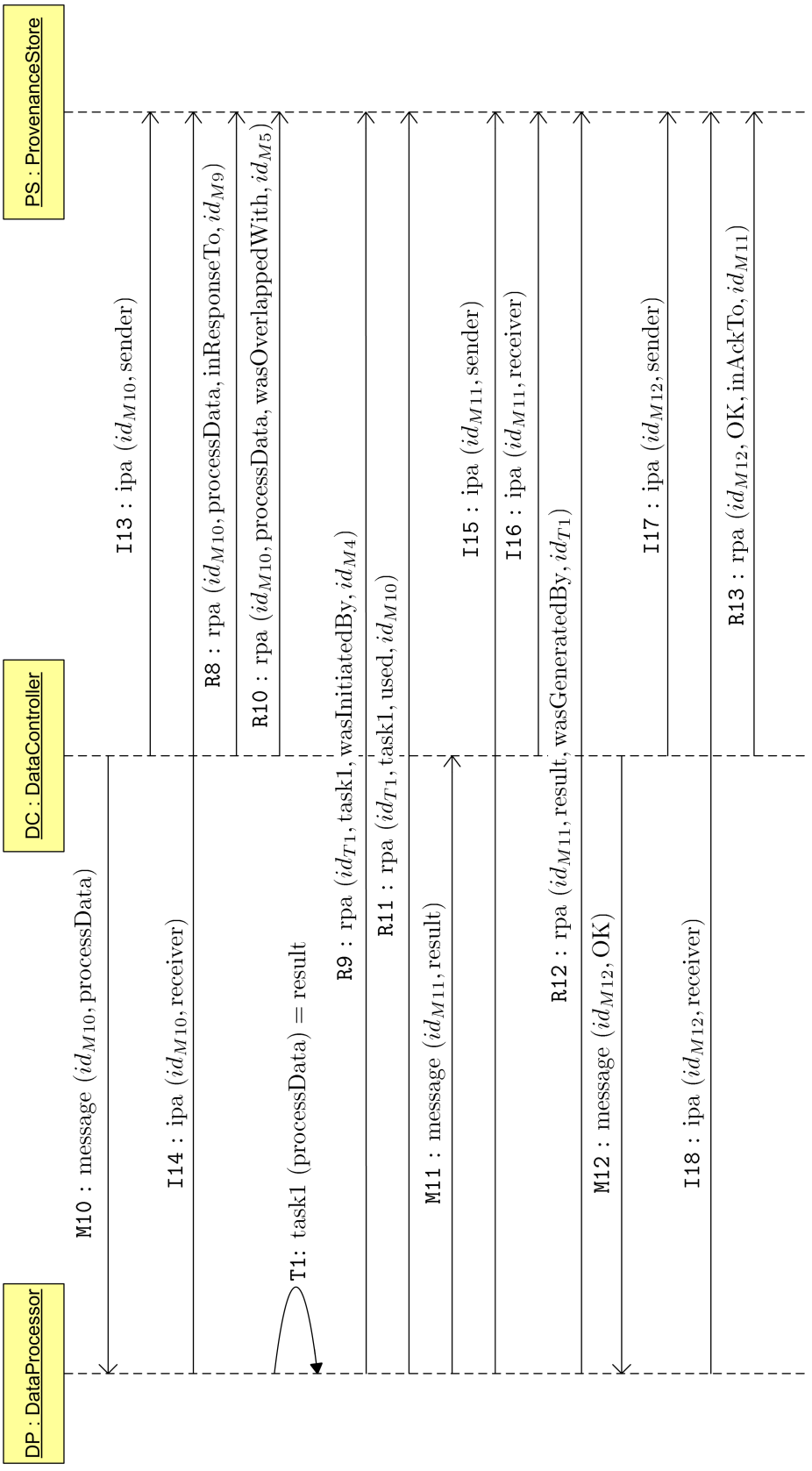


FIGURE 4.6: Task Request UML Sequence Diagram

4.3.3.2 Interaction p-assertions

As in the previous protocol, the protocol for this use case is made provenance-aware by recording interaction p-assertions. Here, assertions `I13`, `I16` and `I17` are related to the act of sending or receiving `M10`, `M11` and `M12` by `DC`. Similarly, assertions `I14`, `I15` and `I18` record the sending or receiving the same messages from the perspective of `DP`.

4.3.3.3 Relationship p-assertions

Relationship p-assertions `R8`, `R9`, `R10`, `R11`, `R12` and `R13` encode the relationships between the messages described above. We now briefly discuss each of them.

`R8` indicates that `M10` was sent *in Response To* `M9`, which is the last message of the authentication protocol execution (as in Section 4.3.2, messages exchanged during execution of this protocol). `R9` indicates that task1 *was Initiated By* the purpose contained in `M4`, which is part of the `Data Request` protocol. `R10` encodes that *processData was Overlapped With* the information contained in `M5`, which is the information sent by the DS in the `Data Request` protocol. `R11` indicates that task1 *used* data contained in `M10`. `R12` indicates that *result was Generated By* `T1`. Finally, `R13` encodes that `M12` was sent in acknowledgement to (*in Ack To*) `M11`.

By recording these assertions, we can capture the provenance of the *processing* of personal information. Thus, the modifications we made to `Data Request` in the previous section, combined with those made here to `Task Request`, reveal the provenance of the entire life cycle of personal data. To query this provenance, we need a new protocol, called `Query Request`. This protocol is developed in the next section.

4.3.4 Recording Provenance in the Query Request Protocol

Using the PrIMe methodology, we exposed the provenance in the two main communication protocols that deal with application data. These protocols were made provenance-aware in the previous two sections. Now, once the p-assertions are stored in the provenance store, an auditor should be able to access them in order to verify that data processing was in compliance with the Auditing Requirements. This process is formalised in the `Query Request` protocol. To keep track of the operations that the auditor performs, this protocol needs to be provenance-aware as well. Without this, the auditor is not accountable, which would constitute a major weakness of the system.

In this section, we present the sequence diagram related to this use case, which is shown in Figure 4.7. This sequence diagram contains the objects `AU` and `PS`, which are instances of the actors `Auditor` and `Provenance Store`. The process that is modelled by this sequence diagram is the following. `Auditor` requests the provenance of certain data item

(*item*) under certain scope (*scope*).The data item and scope of the relevant provenance questions were defined in Section 4.2.1. After receiving the query, `PS` executes it and returns the result (*qresult*) to the `Auditor`. Once the `Auditor` gets the query result, an acknowledgement is sent to `PS`. The *qresult* can subsequently be used to verify any of the Auditing Requirements.

4.3.4.1 Messages

In the sequence diagram presented in Figure 4.7 messages are labelled `Qi`. Message `Q1` contains a provenance query that includes the data *item* from which provenance is sought and the *scope* that defines the context of the provenance query result. Message `Q2` contains the provenance query result (*qresult*). Finally, message `Q3` contains an acknowledgement of the correct receipt of the provenance query result.

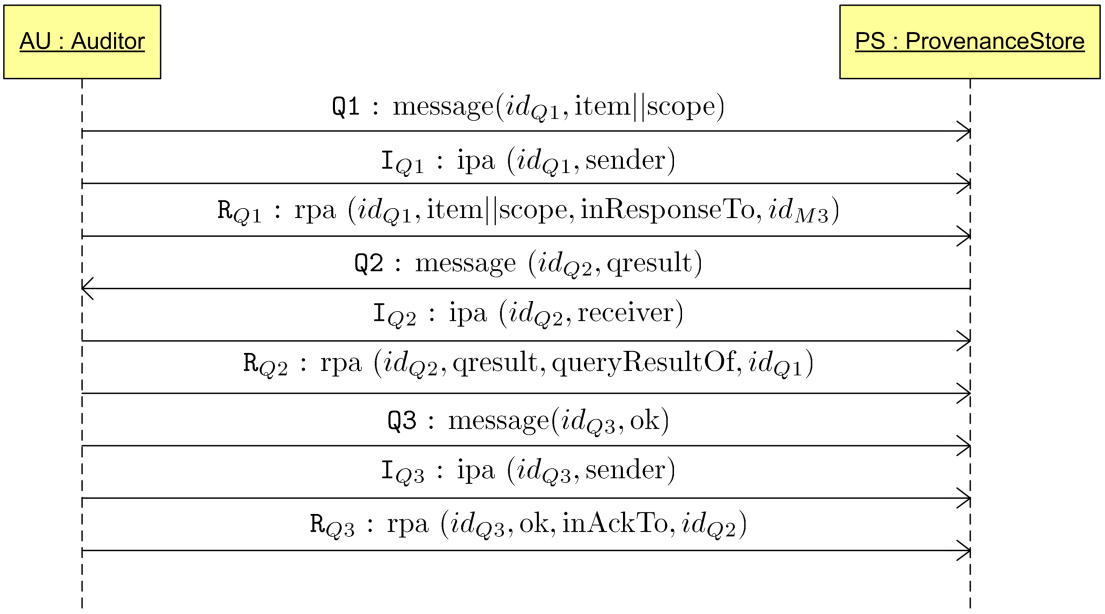


FIGURE 4.7: Query Request UML Sequence Diagram

4.3.4.2 Interaction p-assertions

The interaction p-assertions corresponding to the messages defined above are labelled I_{Qi}. Note that in the sequence diagram, interaction p-assertions generated by `PS` are not shown because they are internally created and recorded by the same entity.

4.3.4.3 Relationship p-assertions

In this protocol, the `Auditor` creates three relationship assertion, R_{Q1}, R_{Q2}, and R_{Q3}.

Assertion $\texttt{R}_{Q1}$ indicates that `Q1` was sent *in Response To* `M3`, which is the last message of the authentication protocol execution. $\texttt{R}_{Q2}$ indicates that *qresult* contained in `Q2` was *query Result Of* the *item* and *scope* contained in `Q1`. Finally, $\texttt{R}_{Q3}$ indicates that `Q3` was sent in acknowledgement to (*in Ack To*) `Q2`.

Now that we have a mechanism for capturing and accessing provenance, we are a step closer to answering the provenance questions stated in 4.2.1.

4.4 Answering Provenance Questions

In this section, we query the provenance captured in the protocols that we made provenance-aware in the previous section. Moreover, we demonstrate that the query results can be represented as a provenance DAG (see Section 2.1.4). Such a DAG indicates where and how the data was used. Thus, by following the relationships in this DAG, it is possible to ascertain how a data item was produced. Here, we perform a manual analysis of several provenance DAGs in an initial attempt to answer the provenance questions we defined in Section 4.2.1. In the next chapter, we automate this analysis to achieve one of the main goals of this thesis: the automatic compliance verification of the principles of DPA discussed in Chapter 3.

In the what follows, we present five examples of DAGs and the analysis that is carried out to answer the provenance questions in Tables 4.1, 4.2, 4.3, 4.4 and 4.5. These DAGs are based on our running example of Section 3.1.1 in which Alice buys medicine from an on-line pharmacy.

4.4.1 Requirement B, Purpose Compliance

Purpose compliance states that data processing was carried out in accordance with the purpose for which it was collected. The corresponding provenance question was presented in Table 4.1. To answer this question, it is necessary to identify the data used to obtain a processing result. In Figure 4.4 this data is called *processData*, and is contained in message `M10`, which is sent by `DC` to `DP`. The data in *processData* is a subset of the data collected by `DC`, i.e. not all the data that was collected is used. However, `M10` can contain additional information, such as information collected from a different entity than `DC` that is necessary for performing the processing. Note that this information could be personal and, therefore, it should also be audited.

4.4.1.1 Querying

To answer the Purpose Compliance question, the p-assertions related to security (among which those related to execution of the authentication protocol) are not necessary.

Therefore, a provenance query results in the form of a DAG of the *result* of a processing *without* security operations is extracted. This is achieved by performing a scoped provenance query [98], in which the scope is used to include only the relevant information in a query result. Thus, in this case, the security functions applied to the exchanged information are excluded. To reflect this, the provenance query presented in Table 4.1 contains "result" as data item and "requested and collected data without security operations" as scope.

To illustrate this process, we focus on the task *manage stock*, which creates a sales report in the On-line Sales Scenario. The result of this process is a report which lists the quantity sold for each product. To audit this process in the context of Alice's data, we construct a provenance query containing Alice's order (*Alice, clomid, 1*) as data item and "requested and collected data without security operations" as scope. The result of this query is shown in Figure 4.8, which is based on the provenance recorded in Figures 4.5 and 4.6. The piece of data whose provenance is sought is shown on the right, in this case *order*. Relationships through which this result was obtained are shown in bold. The data that was used is identified in the DAG with a yellow oval.

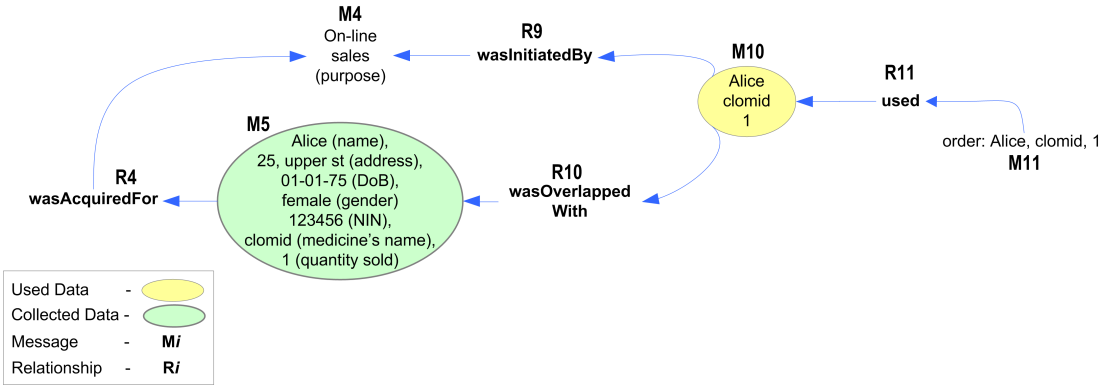


FIGURE 4.8: Provenance DAG of *register* **without** security operations

The provenance query result presented in Figure 4.8 can be encoded using the logical structure shown in Figure 4.9. In practice, this logical structure, which can be used to represent both provenance information and provenance query results, can be stored as an XML document [99]. For the sake of clarity, however, Figure 4.9 does not (strictly) adhere to the specific XML schema used in [99]. Rather, it depicts the most representative elements of the provenance DAG in Figure 4.8 and the relations that exist between them. Figure 4.9 shows the the source (sender) and sink (receiver) of each message and its contents, and the source (cause) and sink (effect) of each relationship and its description. For a more technical description of the logical structure used to represent provenance, we point the reader to [61], specifically Figure 3.6 in Section 3.6.3.

Note that tasks are not explicitly represented in the DAG in Figure 4.8. The reason for this is that the PASOA representation of provenance, which is used here, only models

data and the relationships between data. Processes are modelled as a relationship between the input data and the output data. By so doing, it essentially does not treat processes as first-class citizens. This is a drawback in the PASOA model, because in order to verify the auditing requirements, it is necessary to know which operations were performed internally on the data by a single actor. We come back to this issue in Chapter 5, where we use the OPM model instead.

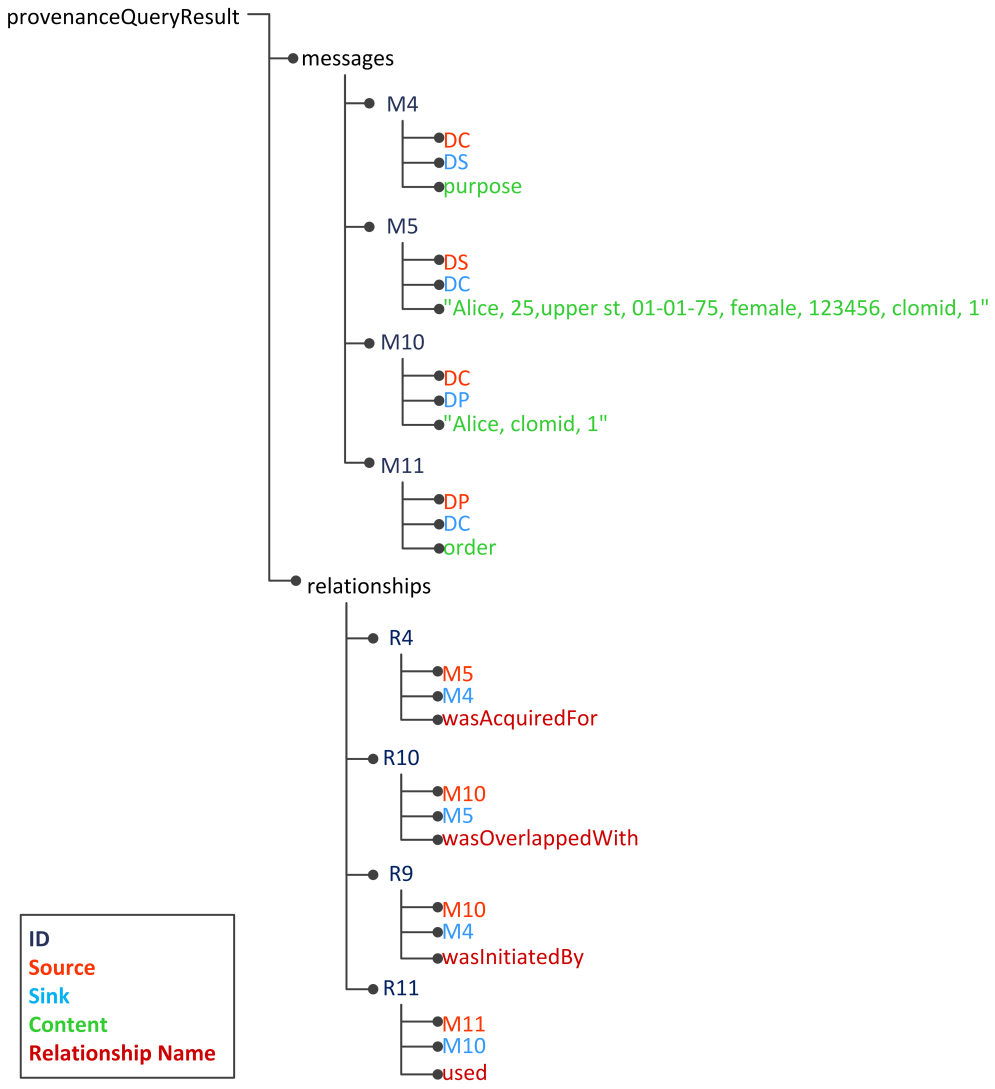


FIGURE 4.9: An alternative representation of the Provenance DAG in Figure 4.8

4.4.1.2 Analysis

To answer the provenance question in Table 4.1, we perform the *processing step* described in this table. To do so, we identify which data was used to process Alice's *order* in the DAG (yellow oval). This data includes the name, medicine's name and the quantity sold to the `Data Subject` who disclosed personal information to the on-line pharmacy.

In this particular case, this data is *Alice, clomid, 1.* We also identify the purpose of collection, which is "on-line sales".

This information needs to be compared with an established criterion—in this case the ICO's Register entry related to the On-line Sales Scenario (Figure 3.2). This register entry contains the purposes for which data is allowed to be collected and the types of data classes (See Section 3.2) that can be captured in relation to each purpose. Thus, we need to verify whether the on-line sales purpose is part of the declared *purposes* and whether the data that was used is part of the *list of Data Classes*. We can see that the *on-line sales purpose* is indeed present in the register entry. The list of data classes of this purpose includes the *personal details class* (containing the name), and the *goods or services provided class* (containing the medicines' names and the quantity sold). Based on this, we can affirm that the part of sales report that used Alice's data was obtained in accordance to the purpose for which was captured. Thus, the *order* was in compliance with Principle 2, i.e. it was processed following Requirement B.

4.4.2 Requirement C, Relevant Information Verification

To answer the provenance question in Table 4.2, it is necessary to identify two types of data: *Used Data*, which was identified in the previous section, and *Collected Data*. Collected data is the information sent by `DS` to `DC` in message `M5` as a result of a request made in message `M4`.

4.4.2.1 Querying

Again, to answer this question, the p-assertions related to security are not necessary. Thus, the provenance query presented in Table 4.2 contains "result" as data item and "requested and collected data without security operations" as scope.

To illustrate this process, we focus on the task that delivers the requested medicine to the buyer's home in the On-line Sales Scenario (*deliver medicine*). The result of this task could be *successful delivery* or *failed delivery*. To start the audit, we construct a provenance query containing this result as data item and "requested and collected data without security operations" as scope. The provenance query result of this provenance query is presented in Figure 4.10, in which collected data is represented by a green oval.

4.4.2.2 Analysis

To perform the processing step described in Table 4.2, we need to identify the data used to deliver Alice's medicine (used data), which is the set *Alice, 25 upper st, clomid, 1* (yellow oval), and the data collected from her (collected data), which is the set *Alice,*

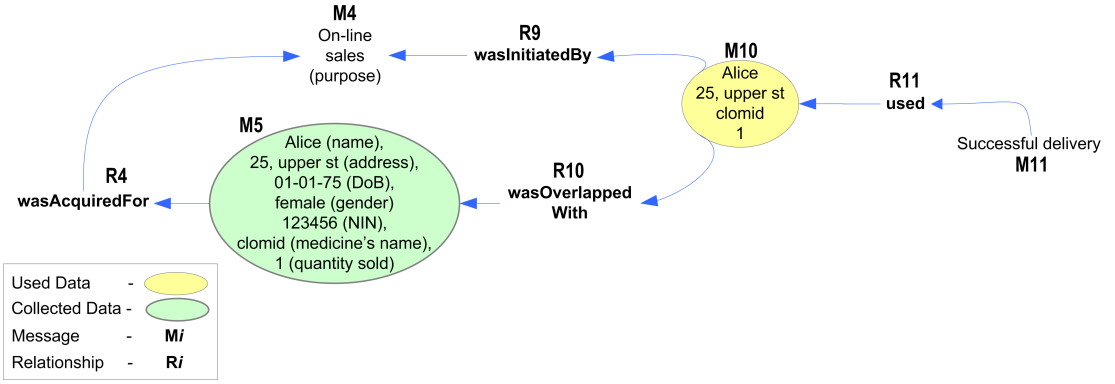


FIGURE 4.10: Provenance DAG of *Successful delivery* **without** security operations

25 upper st, 01-01-75, female, 123456, clomid, 1 (green oval). If we compare both sets, we can see that more data was collected than was necessary to perform the delivery of medicine. This means the delivery of medicine task is not in compliance with Principle 3, i.e. it does not follow Requirement C.

It is important to note that we use assumption 8 (Section 3.6) here, which states that auditing is started *after* the purpose for which personal data was collected has been fulfilled. If this is not the case, auditing might incorrectly conclude that data processing was not in compliance, since some data items were not yet used. Therefore, in order to obtain correct results, it is important that such analysis is postponed until after data processing has terminated.

4.4.3 Requirement F, Anonymity Preservation

To determine whether the Anonymity Preservation requirement was satisfied, we need to identify how the data requested from the `Data Subject` was used (which is contained in `M10`). Then, we can verify whether the results derived from this data were properly anonymised.

4.4.3.1 Querying

To answer the provenance question related to this requirement (presented in Table 4.3), we can reuse the provenance query result of Requirement B (see Figure 4.8). To do so, we identify where Alice's information was used. In this case, one of the uses of her information was to compile an sales report.

4.4.3.2 Analysis

As the processing step of Table 4.3 indicates, we need to determine whether information of a specific individual (in this case, Alice) is present in the result (the sales report).

Now, the ICO's Register entry does not contain a detailed list of data items that are allowed to appear in a result. However, to be able to automatically verify this requirement, we need rules that define which results tasks are allowed produce. In the next chapter, we develop a formalism for defining these rules. For simplicity, in this example we assume that there is a rule that simply states which data items are allowed to appear in the result. In this case, this rule states that the monthly sales report can only contain the name of the medicine and the quantity sold.

Now, in our specific scenario, Alice's name is appears in the sales report. As a result, if another entity (such as the Human Resources Department) has access to this register, it can deduce that Alice is taking a fertility drug. This information could be used against her, as we explained in Section 3.1.1. Thus, we can conclude that this data processing was not in compliance with Principle 5, i.e. it does not follow Requirement F.

4.4.4 Requirement G, Basic Security Characteristics Verification

The Basic Security Characteristics Verification requirement states that all information that is transported should exhibit the four basic security properties: the confidentiality, integrity, authentication and non-repudiation. The corresponding provenance question was presented in Table 4.4. Since this is a security requirement, to answer this question the p-assertions related to security are now required. Thus, the provenance query used here contains "result" as data item and "requested and collected data with security operations" as scope. Even though the assertions related to security are not shown in the corresponding sequence diagrams, we explain how the relationships depicted in Figure 4.4 can be used to verify Requirement G.

4.4.4.1 Querying

To illustrate this process, we again focus on the *manage stock* task from the On-line Sales Scenario. The result of the provenance query (see Table 4.4) in this example is presented in Figure 4.11. In this figure, encrypted data is represented by an orange rectangle and plain data by a blue rectangle. The encryption/signing relationship is labelled *encryptedSigned* and the verification/decryption relationship is labelled *verifiedDecrypted* (both shown in Figure 4.2.3). Therefore, if a provenance DAG contains the relationships *encryptedSigned* and *verifiedDecrypted*, data was transported using a digital signature scheme, and thus exhibits the required security characteristics.

4.4.4.2 Analysis

In Figure 4.11, the result is represented by *cryptoOrder*, which is the same as the one shown in Figure 4.8, but signed and encrypted. If we traverse the graph in the figure, we can see that *cryptoRegister* is the result of the encryption/signing of *plainOrder*, which, in turn, is the result of processing *plainName*, *plainMedName* and *plainQuantitySold*. These are results of the verification/decryption of *cryptoName*, *cryptoMedName* and *cryptoQuantitySold*. That means that the result (*order*) and the used data (*plainName, plainMedName, plainQuantitySold*) were encrypted and signed before being transported and verified and decrypted after being received. Then, we can confirm that *cryptoOrder* was processed in compliance to Principle 7, i.e. it follows Requirement G.

4.4.5 Requirement H, Information Transferred to a Secure Country

The Basic Security Characteristics Verification requirement states that information is only transmitted through the countries listed as secure by the DPA. Again, to answer the corresponding provenance question (Table 4.5), the p-assertions related to security are not necessary. Thus, the provenance query related to this requirement contains "result" as data item and "requested and collected data without security operations" as scope.

4.4.5.1 Querying

To illustrate this process, we need to focus on the task that transfers information between systems in different countries, which in the On-line Sales Scenario is called *transfer information*. The result of such a task could be *successful transfer to country X* or *failed transfer to country X*. Thus, we construct a provenance query with the result of data processing as data item.

4.4.5.2 Analysis

The result of this provenance query is presented in Figure 4.12. In this figure, we can see that two items of personal data related to Alice were transferred to an EU country. As can be seen in the corresponding ICO's Register entry (see Figure 3.2), the pharmacy is allowed to transfer data only to countries inside of the European Economic Area. Thus, in this case, Alice's personal information was transferred in compliance to Principle 8, i.e. it follows Requirement H.

This concludes our discussion of how the provenance captured in Section 4.3 can be used to answer the provenance questions we stated in Section 4.2. We demonstrated that provenance can be represented as a DAG, which indicates where and how the data

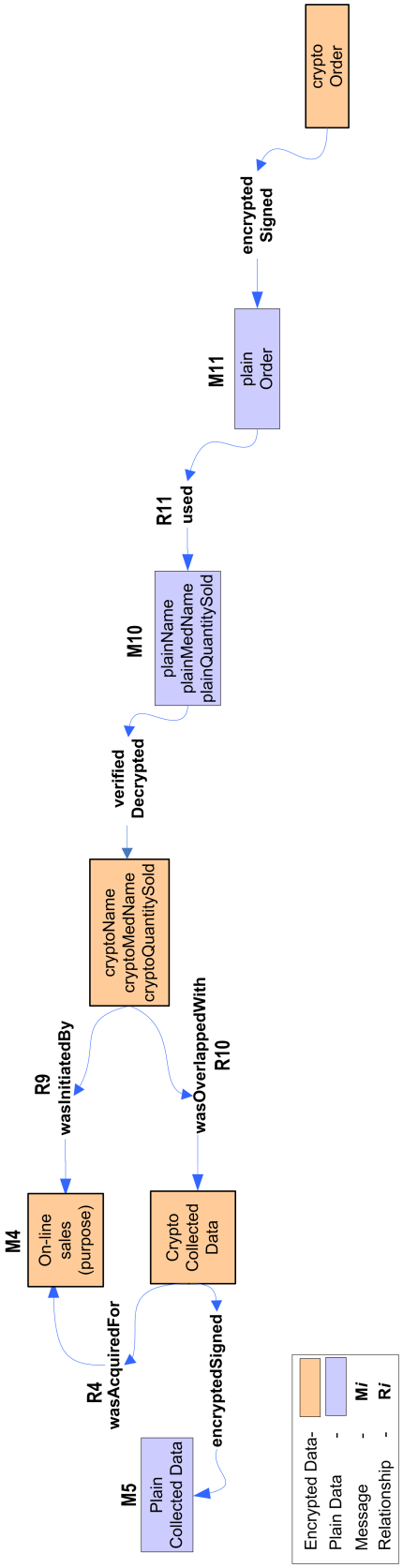


FIGURE 4.11: Provenance DAG of *cryptoaverageAge* **with** security operations

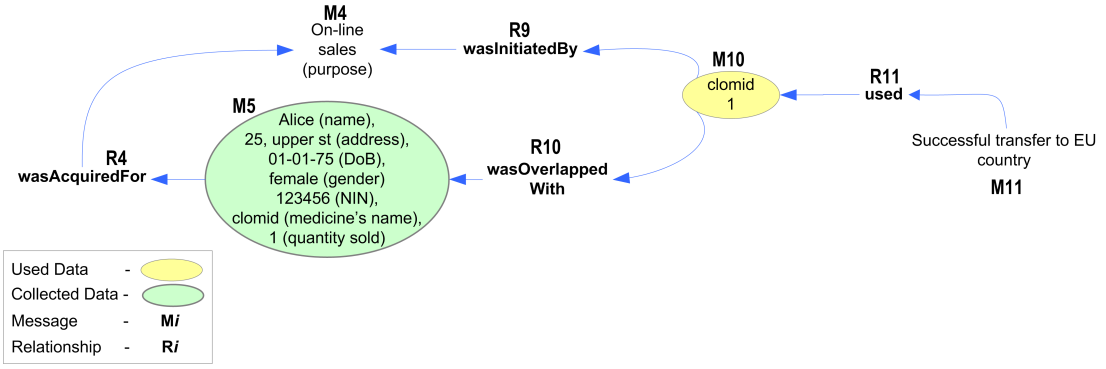


FIGURE 4.12: Provenance DAG of *Successful transfer* **without** security operations

was used. In an initial attempt to answer the provenance questions, we performed a manual analysis of these DAGs. This analysis will be automated in Chapter 5.

4.5 Discussion

Existing work has proposed the use of provenance to audit the quality of information as well as its use [114, 40, 82].

Philip *et al.* [114] and Chorley *et al.* [40] propose the idea of using provenance to create audit reports related to Evidence-Based Policy Assessment (EBPA). They suggest using provenance to evaluate the quality and reliability of data, the robustness of an analysis, and the validity of findings in an assessment. However, they do not present any analysis that shows how to use provenance in this specific field.

Kifor *et al.* [82] proposes the use of provenance in electronic healthcare record systems. They also investigate the privacy risk aspects of introducing provenance into healthcare systems. To decrease this risk, they propose not to store sensitive medical data in the provenance store, but only references. By doing so, unauthorized users are unable to link medical data to specific individuals. The described technique is already included in our work, not only for security reasons but also for scalability. To use provenance information for making audits is mentioned but not fully discussed. Neither do they present audit use cases or an analysis of security characteristics.

4.6 Conclusions

In this chapter, we have developed a Provenance-Based Auditing Architecture that supports audits of the processing of personal data. This architecture is based on the principles of the DPA and protocols we identified in Chapter 3. It can be implemented by

systems that process personal information and, as such, have to follow the rules of a legislative framework.

To determine which assertions need to be recorded in the Provenance Store to determine whether processing is in compliance with such a framework, we applied the PrIMe methodology. PrIMe identifies the provenance that needs to be captured in order to verify the Auditing Requirements. We extended the Data Request and Task Request protocols defined in Chapter 3 to record and secure this provenance, and introduced a third one, Query Request. This protocol can be used by an auditor to retrieve the provenance generated in the first two protocols.

Finally, we presented a preliminary set of provenance query results, in an initial attempt to determine whether data processing was in compliance with the Auditing Requirements. These query results are represented as DAGs. We also presented an intuitive manual analysis of these DAGs to determine whether information was processed according to these requirements. In so doing, we have shown that provenance, specifically in the form of a DAG, can facilitate audits on the processing of personal data and decide whether such processing was done in compliance with the DPA regulatory framework.

Now that we have an architecture that captures the necessary provenance, as well as a means to query this provenance, we can proceed to the second main contribution of this thesis: the automatic verification of the Auditing Requirements. This the subject of the next chapter.

Chapter 5

Compliance Framework

In the previous chapter, we developed the Provenance-based Auditing Architecture that captures and records provenance of the operations applied to personal information. In this chapter, we use this provenance to achieve the second main contribution of this thesis: a suite of algorithms that automatically verifies whether data processing was performed in compliance with the Auditing Requirements defined in Chapter 3.4.

To achieve this, we need to make two additional augmentations to the architecture. First, not only provenance should be available in a machine-readable format (i.e. a provenance DAG), but also the *rules* data processing needs to adhere to. These rules specify which data can be used and the legal operations that can be applied to it. As we discussed in Chapter 3.2, within the DPA framework these rules are described in the ICO's Register. Thus, to allow for automatic verification of these rules, our first step is to transform these into a machine-readable format as well. The result is a novel representation of processing rules using DAGs. While doing this, we will see that the ICO's Register is incomplete: we identify requirements that need to be satisfied, which are not explicitly stated in this register. We already encountered one example of this in Section 4.4 where we attempted to manually verify compliance of Requirement F, Anonymity Preservation. Specifically, we demonstrated that an explicit description of the data types and classes that are allowed to be present in a result were missing. Thus, in this chapter, we extend these rules to ensure that they fully capture the intent of the DPA.

The second augmentation needs to be made in the provenance DAGs. In the previous chapter, these DAGs expressed the data as nodes, and the processes as relations (relationship p-assertions to be more precise). However, by doing this, processes are treated as "second-class citizens", despite the fact that they are an essential aspect of a datum's provenance. Thus, to reflect their true importance, we adopt the OPM model [104], in which provenance is also represented as a DAG that contains specific elements to differentiate between artifacts (pieces of data) and processes (actions).

After making these two modifications, we develop the primary contribution of this chapter: the Compliance Framework. This framework combines the novel representation of processing rules mentioned above and a representation of provenance in the OPM model with a suite of algorithms for automatic verification of the Auditing Requirements.

Thus, in summary, the remainder of this chapter is organised as follows. First, in Section 5.1, we present the definitions and notation that are used throughout this chapter. Then, in Section 5.2, we present our novel representation of processing rules. Next, in Section 5.3, we present the provenance DAG representation of the operations applied to personal information within the Open Provenance Model. Given this rule representation and provenance DAGs, in Section 5.4, we develop a set of algorithms that perform the automatic verification of the Auditing Requirements. This verification is made by comparing the processing rules against the provenance DAG representation. We illustrate how this verification is performed in our running example of the On-line Pharmacy Scenario (Section 3.1). Finally, in Section 5.5 we discuss the contributions made in this chapter in the context of existing work, and conclude in Section 5.6.

5.1 Preliminaries

Before presenting the Compliance Framework, we first show some definitions and notation that are used throughout this chapter. Specifically, we introduce the *purposes* (p_i), which are the goals for which a set of data was collected, the *tasks* (t_i), which are the processes applied to data. Furthermore, we represent collected data by D_{Ai}, processed data by D_{Ui}, and *processing results* by R_i. Below, we describe these concepts in further detail.

Let $\mathcal{P}$ be the universal set of purposes. Then, there are two different types of purposes, *Collection Purposes* and *Processing Purposes*, which are defined as follows:

Definition 5.1 (Collection Purposes)**.** A set of purposes for which the data is to be collected. This set is a subset of $\mathcal{P}$.

Definition 5.2 (Processing Purposes)**.** A set of purposes for which the data is to be processed. This set is a subset of $\mathcal{P}$.

Going back to the On-line Sales Scenario, a collection purpose might be "on-line sales". An example of a processing purpose is "creating sales reports".

Furthermore, let $\mathcal{T}$ be the universal set of tasks. Then, we can define the concept of a *set of tasks* as:

Definition 5.3 (Set of Tasks)**.** A set of tasks is a set of operations that were applied to data. This set is defined as follows:

$$T = \{t_1, t_2, ..., t_n\}$$

where $n > 0$ and $T \subseteq \mathcal{T}$.

Then, a data *type* is defined as a formal description of a set of values and the basic operations that can be applied on them. Example of data types are "string", "integer" and "boolean". A data *class* is an attribute of an object with constraints for being a member of the class. Examples of classes are "name", "address" and "gender". Finally, a data instance is defined as an occurrence of an object with a data type which belongs to a certain class. For example, "Alice" is an instance of the data type "string" and belongs to the class "name".

Next, if we consider $\mathcal{Types}$ as a set containing all data types, $\mathcal{Instances}$ as a multiset containing all data instances and $\mathcal{Classes}$ as a set containing all data classes, we can express the following:

Definition 5.4 (Requested Data Classes)**.** Requested Data Classes are a set composed of ordered pairs (x, y), where x is a data class and y is the data type of such a class, representing the data classes that are requested from a user. This set is defined as follows:

$$C_Q = \{(class_1, type_1), (class_2, type_2), ..., (class_n, type_n)\}$$

where $n > 0$, $class_i \in \mathcal{Classes}$ and $type_i \in \mathcal{Types}$ for $1 \leq i \leq n$

Definition 5.5 (Used Data Classes)**.** Used Data Classes are a set containing ordered pairs (x, y), where x is a data class and y is the data type of such a class, representing the data classes that are used to perform a task. This set is defined as follows,

$$C_U = \{(class_1, type_1), (class_2, type_2), ..., (class_n, type_n)\}$$

where $n > 0$, $class_i \in \mathcal{Classes}$ and $type_i \in \mathcal{Types}$ for $1 \leq i \leq n$

Definition 5.6 (Generated Data Classes)**.** Generated Data Classes are a set composed of ordered pairs (x, y), where x is a data class and y is the data type of such a class, representing the data classes that are the result of a processing. This set is defined as follows,

$$C_G = \{(class_1, type_1), (class_2, type_2), ..., (class_n, type_n)\}$$

where $n > 0$, and for $1 \leq i \leq n$, $class_i \in \mathcal{Classes}$ and $type_i \in \mathcal{Types}$

In practice, the names of the tasks, the names of the data classes and the data types correspond to the names of the processes, the names of the data attributes and the data

types used at execution time, respectively. These names are created by system designers and should be made explicit when a system is made provenance aware. The entity who creates the processing rules should also be notified of these names, so it can use these in the corresponding processing rules.

The three previous definitions are used to define the processing rules. Since these rules are created at design time, they refer to data classes and data types, not actual data. In contrast, the three definitions that follow describe the actual data.

Definition 5.7 (Collected Data Instances)**.** Collected Data Instances are a multiset of data instances that are collected for Collection Purpose p (Definition 5.1). Collected Data Instances is a multiset, because instances can occur multiple times. For example the integer '25' can occur as an age as well as a quantity. This set is defined as follows,

$$D_A^p = \{d_{A1}, d_{A2}, ..., d_{An}\}$$

where $n > 0$, $D_C \subseteq \mathit{Instances}$ and $p \in \mathcal{P}$.

Definition 5.8 (Generated Data Instances)**.** Generated Data Instances are a multiset of data instances are the result of a processing, and also known as *processing results*. This set is defined as follows:

$$R = \{r_1, r_2, ..., r_n\}$$

where $n > 0$ and $R \subseteq \mathit{Instances}$.

Definition 5.9 (Used Data Instances)**.** Used Data Instances are a multiset of data instances that are used to perform a task, and are also known as *input data*. This multiset can contain sets of collected data instances (Definition 5.7) or sets of generated data instances (Definition 5.8), and is defined as follows:

$$D_U = \{d_{U1}, d_{U2}, ..., d_{Un}\}$$

where $n > 0$ and $D_U \subseteq \mathit{Instances}$.

The multisets of data instances defined above contain collected or used instances of data. These instances have a *class* and *type*. Thus, it is possible to associate the elements of $\mathit{Instances}$ with a class, which is an element of $\mathcal{Classes}$, and with a type, which is an element of $\mathcal{Types}$. As a result, an element e of a multiset of data instances is defined as $e \equiv (d, cl, ty)$, where $d \in \mathit{Instances}$, $cl \in \mathcal{Classes}$ and $ty \in \mathcal{Types}$. We define the accessors as $instance(e) = d$, $class(e) = cl$ and $type(e) = ty$.

Generated Data Instances contain results of processing which can be reused by different processes to produce a new result. Thus, Used Data Instances can contain elements

of Collected Data Instances and Generated Data Instances, or just Collected Data Instances. Later, in Section 5.4.1.3, we use this distinction to determine whether Generated Data Instances were correctly reused.

Finally, there exist relationships between the sets and multisets from Definitions 5.4, 5.5, 5.6, 5.7, 5.8 and 5.9. For example, such a relationship can represent that a multiset of Collected Data Instances was acquired for set of Collection Purposes. These relationships are identified by labels. If $\mathcal{Rel}$ is the universal set of relationship labels, then we derive the next definition.

Definition 5.10 (Relationship Labels)**.** Relationship labels identify relationships between elements of data processing. This set is defined as follows,

$$Rel = \{l_1, l_2, ..., l_n\}$$

where $n > 0$ and $Rel \subseteq \mathcal{Rel}$.

Using these definitions, we now present each component of the Compliance Framework in more detail. First, we focus on the Usage Rules Definition component, which represents the rules within our framework.

5.2 Usage Rules Definition

The *Usage Rules Definition* (URD) is a novel graph-based representation of a set of rules that can be verified within our Compliance Framework. These rules express which operations an application is allowed to perform on personal information, and are derived from the ICO's Register (see Section 3.2). In more detail, rules within the URD specify the conditions under which processing is valid over a set of data classes.

To represent the URD, we formally define the concept of a rule inspired by the ICO's Register, which contains the purposes for collection and the data classes to collect. Each rule represents the way in which a set of requested data classes (C_Q) can be used, i.e. which tasks can use a certain set of data classes to accomplish which purpose (p). To make this component and its corresponding analysis stage more fine-grained, we add some additional elements. These are: tasks to be performed (t) to accomplish purposes, data classes to be used (C_U) and data classes to be generated ($C_\mathcal{G}$) by such tasks. The rule definition is presented below:

Definition 5.11 (Processing Rule)**.**

$$Rule = \langle p, C'_Q, C_{Ui}, t_i, C_{\mathcal{G}i} \rangle$$

where $p \in \mathcal{P}$, $C'_Q \subseteq C_Q$, $C_{Ui} \subseteq C_U$,$t_i \subseteq \mathcal{T}$, $C_{\mathcal{G}i} \subseteq C_\mathcal{G}$

This rule expresses that, within the context of a collection purpose p, certain data classes and types can be requested. The requested set of data classes and types (C'_Q) is then divided into sets of used data classes (C_{Ui}) that are used by the corresponding tasks t_i to produce the corresponding set of generated data classes ($C_{\mathcal{G}i}$).

By including tasks, used data and generated data classes elements, these rules express in a more detailed way how information should be processed, compared to the rules in the ICO's Register. This enables the framework to verify the Auditing Requirements. For example, in the On-line Sales Scenario, we can define a processing rule for the "manage stock" task. This rule indicates that the set of personal information C_Q is collected based on the purpose "on-line sales". Then, C_U, which is a subset of C_Q, can be used by the task "manage stock" to obtain the set $C_{\mathcal{G}}$.

$$
\begin{aligned}
manageStock = \langle & \\
p &= \{\text{on-line sales}\}, \\
C_Q &= \{(name, string), (address, string), \\
&\quad (medicineName, string), (quantity, integer)\}, \\
C_U &= \{(medicineName, string), (quantity, integer)\}, \\
t &= \{\text{manage stock}\}, \\
C_{\mathcal{G}} &= \{(medicineName, string), (quantity, integer)\}\rangle
\end{aligned}
$$

By exposing the tasks' names, the data classes and the data types that each Data Controller is allowed to use in the processing of data, the analysis of provenance query results (such as the ones presented in Section 4.4) can be simplified. The URD combines multiple of these processing rules into a single formalism.

In more detail, the Usage Rules Definition is a set of processing rules represented by an edge-labelled directed acyclic graph (DAG) called G_Γ that is depicted in Figure 5.1. In this figure, the elements of G_Γ are represented by special symbols: purposes are represented by double-line rounded rectangles, tasks by squares, set of data classes by ovals and the set of generated data classes by bold-line ovals. These components are connected by labelled arrows indicating the relationships that exist among them. The labels of relationships are actions, which are stated in the present tense, since they express the fact that these actions are expected to always happen. More formally the *Usage Rules Definition Graph* G_Γ is defined as:

Definition 5.12 (Usage Rules Definition Graph)**.** Let us consider a set of purposes P_Γ, a set of data classes C_{Q_Γ}, a set of used data classes C_{U_Γ}, a set of tasks T_Γ, a set of generated

data classes $C_{\mathcal{G}_\Gamma}$, and a set of relationship's names $Rel_\Gamma = \{\mathsf{isAcquiredfor}, \mathsf{overlapsWith}, \mathsf{isInitiatedBy}, \mathsf{uses}, \mathsf{isGeneratedBy}\}$.

A Usage Rules Definition Graph $G_\Gamma = (V_\Gamma, E_\Gamma, Rel_\Gamma)$ is then a directed acyclic graph, where $V_\Gamma \subseteq P_\Gamma \cup C_{Q_\Gamma} \cup C_{U_\Gamma} \cup T_\Gamma \cup C_{\mathcal{G}_\Gamma}, E_\Gamma \subseteq (C_{Q_\Gamma} \times P_\Gamma \times \mathsf{isAcquiredfor}) \cup (C_{U_\Gamma} \times C_{Q_\Gamma} \times \mathsf{overlapsWith}) \cup (T_\Gamma \times C_{U_\Gamma} \times \mathsf{uses}) \cup (T_\Gamma \times P_\Gamma \times \mathsf{isInitiatedBy}) \cup (C_{\mathcal{G}_\Gamma} \times T_\Gamma \times \mathsf{isGeneratedBy})$.

G_Γ contains a set of purposes (P_Γ) which users' data (represented as sets of requested data classes C_Q) **is acquired for**. Sets of used data classes (C_U) **overlap with** a set of requested data, i.e. the requested data is divided in subsets of used data. Thus, a set of tasks (T_Γ) **uses** sets of used data classes to **generate** results, which we call sets of generated data classes. This way, each task can only use and produce the indicated set of data classes. Finally, to accomplish the initially stated purpose, tasks (T_Γ) **are initiated by** such a purpose.

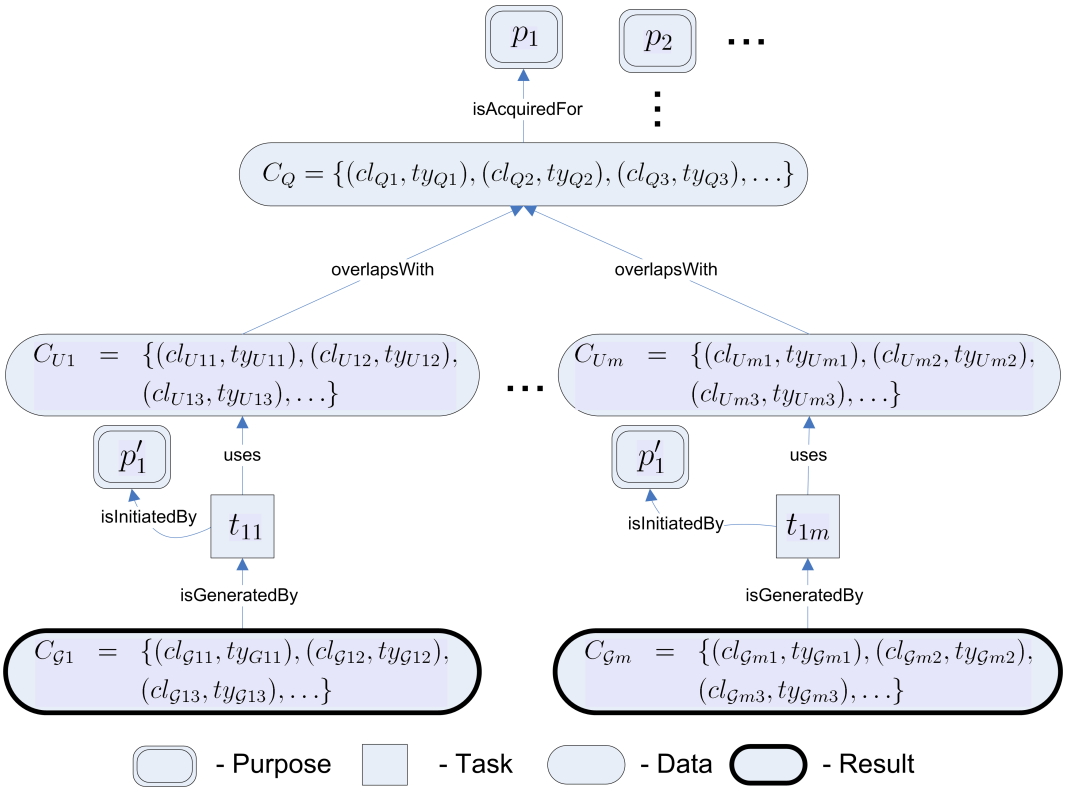


FIGURE 5.1: Usage Rules Definition Graph G_Γ

The URD can contain more than one purpose. Each purpose has a set of data associated with it that is requested in order to achieve it. The set of requested data is divided into multiple subsets, each of which is used by a task to produce a result. It is important to note that the union of all the used data sets should be equal to the requested data set, i.e. all the data that is requested should be used (Requirement C, see Section 3.4.3), although, more be requested than needed. Also note that an element of the set of requested data classes can be used in more than one set of used data classes. Below, the

Usage Rules Definition Graph edges (representing relations between data and tasks) are formally defined.

Let $G_\Gamma = (V_\Gamma, E_\Gamma, Rel_\Gamma)$ be a Usage Rules Definition Graph, $C_\mathcal{G}$ be a set of generated data classes, t be a task in T_Γ, C_{Ui} be a set of used data classes, C_Q be a set of requested data classes, and p be a purpose in P_Γ.

Definition 5.13 (Result is generated by Task)**.** A set of generated data classes $C_\mathcal{G}$ is generated by task $t \in T_\Gamma$ if there exists $(C_\mathcal{G}, t, \mathsf{isGeneratedBy}) \in E_\Gamma$.

Definition 5.14 (Task uses Used Data Classes)**.** Task t uses a set of used data classes $C_U \in C_{U_\Gamma}$ if there exists $(t, C_U, \mathsf{uses}) \in E_\Gamma$.

Definition 5.15 (Task is initiated by Purpose)**.** Task t is initiated by purpose $p \in P_\Gamma$ if there exists $(t, p, \mathsf{isInitiatedBy}) \in E_\Gamma$.

Definition 5.16 (Used Data overlaps with Requested Data)**.** A set of Used Data Classes C_U overlaps with a set of requested data classes $C_Q \in C_{Q_\Gamma}$ if there exists $(C_U, C_Q$ $\mathsf{overlapsWith}) \in E_\Gamma$.

Definition 5.17 (Requested Data is acquired for Purpose)**.** A set of requested data classes C_Q is acquired for Purpose $p \in P_\Gamma$ if there exists $(C_Q, p, \mathsf{isAcquiredFor}) \in E_\Gamma$.

In summary, by following the arrows in Figure 5.1, Definition 5.13 indicates that a set of generated data classes $C_\mathcal{G}$ **is generated by** a task t; Definition 5.15 indicates that a task t **is initiated by** a purpose p; Definition 5.14 indicates that a task t **uses** the set of data classes C_{Ui}; Definition 5.16 indicates that set of used data classes C_{Ui} **overlaps with** a set of requested data classes C_Q and Definition 5.17 indicates that the set of requested data classes C_Q **is acquired for** a set of purposes P. As the Usage Rules Definition Graph defines how the data should be used (which is done at design time), it only contains sets of data classes not actual data.

Now that we have defined the nodes (data classes and tasks) and the edges (the relationships defined in Definitions 5.13 to 5.17), we now define what constitutes a *valid* Usage Rules Definition Graph.

Property 1 (Well-Formed Usage Rules Definition Graph)**.** Let G_Γ be a Usage Rules Definition Graph. G_Γ is a Well-Formed Usage Rules Definition Graph if and only if

a) For any $C_\mathcal{G} \subseteq C_{\mathcal{G}_\Gamma}$ there exist one and only one $t \in T_\Gamma$, such that, $C_\mathcal{G}$ is generated by t.

b) For any $t \in T_\Gamma$ there exists one and only one $p \in P_\Gamma$ and one and only one $C_{Ui} \subseteq C_{U_\Gamma}$, such that t is initiated by purpose p and t uses a set of data classes C_{Ui}.

c) For any $C_{Ui} \subseteq C_{U_\Gamma}$, there exist one and only one $C_Q \subseteq C_{Q_\Gamma}$ such that C_{Ui} overlaps with a set of data classes C_Q and $\bigcup_i C_{Ui} = C_{U_\Gamma}$.

d) For any $C_Q \subseteq C_{Q_\Gamma}$ there exist one and only one $p \in P_\Gamma$ such that C_Q is acquired for a purposes p.

Property 1 indicates that a well-formed Usage Rule Definition Graph can contain more than one set of generated data classes $C_\mathcal{G}$, which are generated by one and only one task t. Such a task is initiated by one and only one purpose p and uses one and only one set of data classes C_{Ui}. Each C_{Ui} is overlapped by one and only one C_Q. At the same time, all the elements that are collected should be used, therefore, the union of all the C_{Ui} should be equal to C_Q (Property 1.c). Finally, C_Q is acquired for one and only one purpose p.

It is important to note that the algorithms we develop in Section 5.4 are only guaranteed to produce correct results on well-formed Usage Rule Definition Graphs.

To illustrate the URD, we present an example. For continuity, this example is based on our running example of the On-line Sales Scenario, which was presented in Section 3.1.1. Throughout this chapter, we use the Compliance Framework to verify whether Alice's personal information was used by the pharmacy in accordance to the usage rules. In this analysis, we focus on two tasks `deliver medicine` and `manage stock` that are part of the main purpose: `on-line sales`. Figure 5.2 presents the Usage Rules Definition related to the "on-line sales" purpose. As previously described, this component contains data classes and their corresponding data types. Figure 5.2 shows that to accomplish the `on-line sales` purpose the pharmacy collects the name of a person, address, medicine's name, and its quantity. To conduct the `deliver medicine` task the pharmacy needs the name of a person, address, the medicine's name to be delivered and its quantity, and for the `manage stock` task it just needs the medicine's name and its quantity. Both tasks are initiated by the same initial purpose, `on-line sales`, and produce a delivery state and a sales report, respectively.

Next, we present the second component of the Compliance Framework, the Processing View. In contrast to the Usage Rule Definition, the Processing View represents the actual operations performed on personal data.

5.3 Processing View

In the previous section, we created the Usage Rules Definition, a novel representation of rules that specifies which operations are allowed to be performed on personal data. To verify whether data processing was in compliance with the Auditing Requirements, we also need to know the operations were *actually* performed, and on which data. In the previous chapter, we represented the latter as nodes, and the former as edges in provenance DAGs. However, as mentioned in the introduction to this chapter, this does not do justice to the importance of the operations, since these form an important part

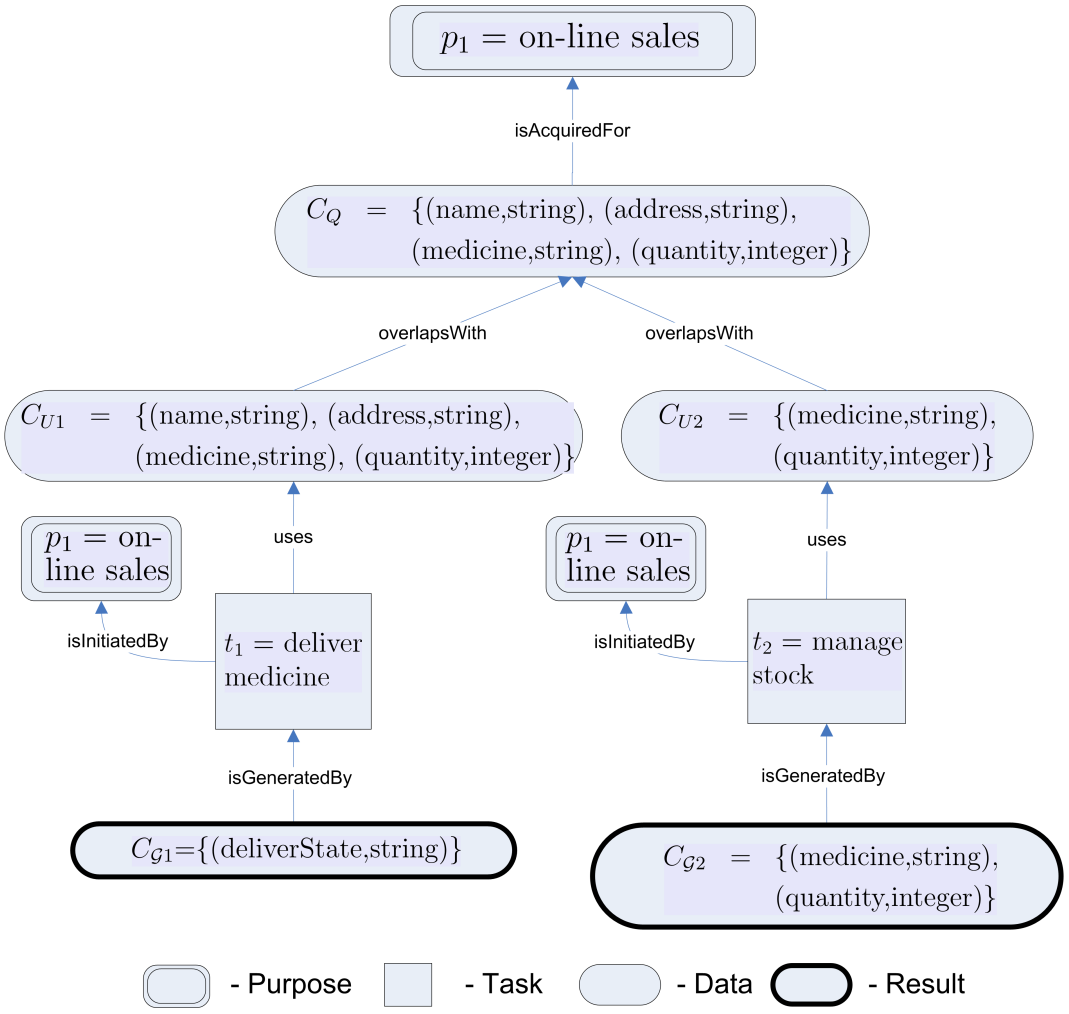


FIGURE 5.2: On-line Sales Example - Usage Rules Definition

of the provenance of data. Furthermore, upgrading operations from edges to nodes simplifies the verification process we perform in the next section. In light of this, we choose to represent provenance using the Open Provenance Model [104]. By doing so, we develop the second component of the Compliance Framework, the Processing View.

In more detail, the *Processing View* represents a provenance graph captured at execution time that contains information concerning past processing of the Data Subject's data. The Processing View is an edge-labelled directed acyclic graph called G_W that is very similar to the provenance graphs presented in Section 4.4. However, in contrast to these provenance graphs, the Processing View explicitly represents the task that was executed to obtain a result (as a node in the graph instead of an edge, i.e. relationship).

The Processing View presented in Figure 5.3 represents the life cycle of general data processing. In this view, nodes represent sets of data instances and the operations performed on them, and edges represent relations that exist between sets of data instances and operations (usage or result), or between two sets data instances (e.g. subset of). In Figure 5.3, the labels of the relationships are presented in past tense expressing the

fact that these actions happened in the past. Accordingly, the life cycle began when an application requested a multiset of data from a user declaring the purpose for which such a set **was acquired** (*collection purpose*). After checking the application purpose, the user sent the requested multiset of data instances (*collected data instances*). The goal of the application was to achieve the collection purpose. As a consequence, a *task* **was initiated by** the related *processing purpose*. Such a task **used** a multiset of data instances that **overlapped with** a multiset of *used data instances*. Note that the task could have used the previous collected multiset of data instances or a subset of it. Later, the task was executed with the used data instances as input and **generated** *results*. These results could have been reused in the execution of a new task to appear as collected data instances.

For the sake of clarity, Figure 5.3 shows a single multiset of data instances collected from a single entity (D_A). However, in real applications more than one multiset of data instances can be collected from different entities generating a graph that contains all these sets of data instances. This issue is discussed in detail in Section 5.4.1.

In Figure 5.3, the nodes of the Processing View are represented by special symbols: purposes are represented by double-line ovals, tasks by squares, multisets of collected data instances by ovals and multisets of processed data instances by bold-line ovals. These elements are connected by labelled arrows with the name of the relationships that exist between them.

Using these definitions, the *Processing View Graph* G_W can be defined as:

Definition 5.18 (Processing View Graph)**.** Let us consider a set of purposes P_W, a set of tasks T_W, a multiset of collected data instances D_{A_W}, a multiset of used data instances D_{U_W}, a multiset of results R_W and a set of relationship's names $Rel_W = \{\mathsf{wasInitiatedBy}, \mathsf{used}, \mathsf{overlappedWith}, \mathsf{wasGeneratedBy}, \mathsf{wasAcquiredFor}, \mathsf{contained}\}$.

A Processing View Graph $G_W = (V_W, E_W, Rel_W)$ is a directed acyclic graph, where $V_W \subseteq P_W \cup T_W \cup D_{A_W} \cup D_{U_W} \cup R_W$, $E_W \subseteq (R_W \times T_W \times \mathsf{wasGeneratedBy}) \cup (T_W \times P_W \times \mathsf{wasInitiatedBy}) \cup (T_W \times D_{U_W} \times \mathsf{used}) \cup (D_{A_W} \times D_{C_W} \times \mathsf{overlappedWith}) \cup (D_{U_W} \times R_W \times \mathsf{overlappedWith}) \cup (D_{A_W} \times P_W \times \mathsf{wasAcquiredFor})$.

Using all the concepts developed above, the edges (representing relations between data instances and processes, and among data instances themselves) of a Processing View Graph can now be formally defined. These edges mirror those defined in Section 5.2.

Let G_W be a Processing View Graph, R be a multiset of results in R_W, t be a task in T_W, D_{Ui} be a multiset of used data instances, and D_A be a multiset of collected data instances.

Definition 5.19 (Result was generated by Task)**.** R was generated by task $t \in T_W$ if there exists $(R, t, \mathsf{wasGeneratedBy}) \in E_W$.

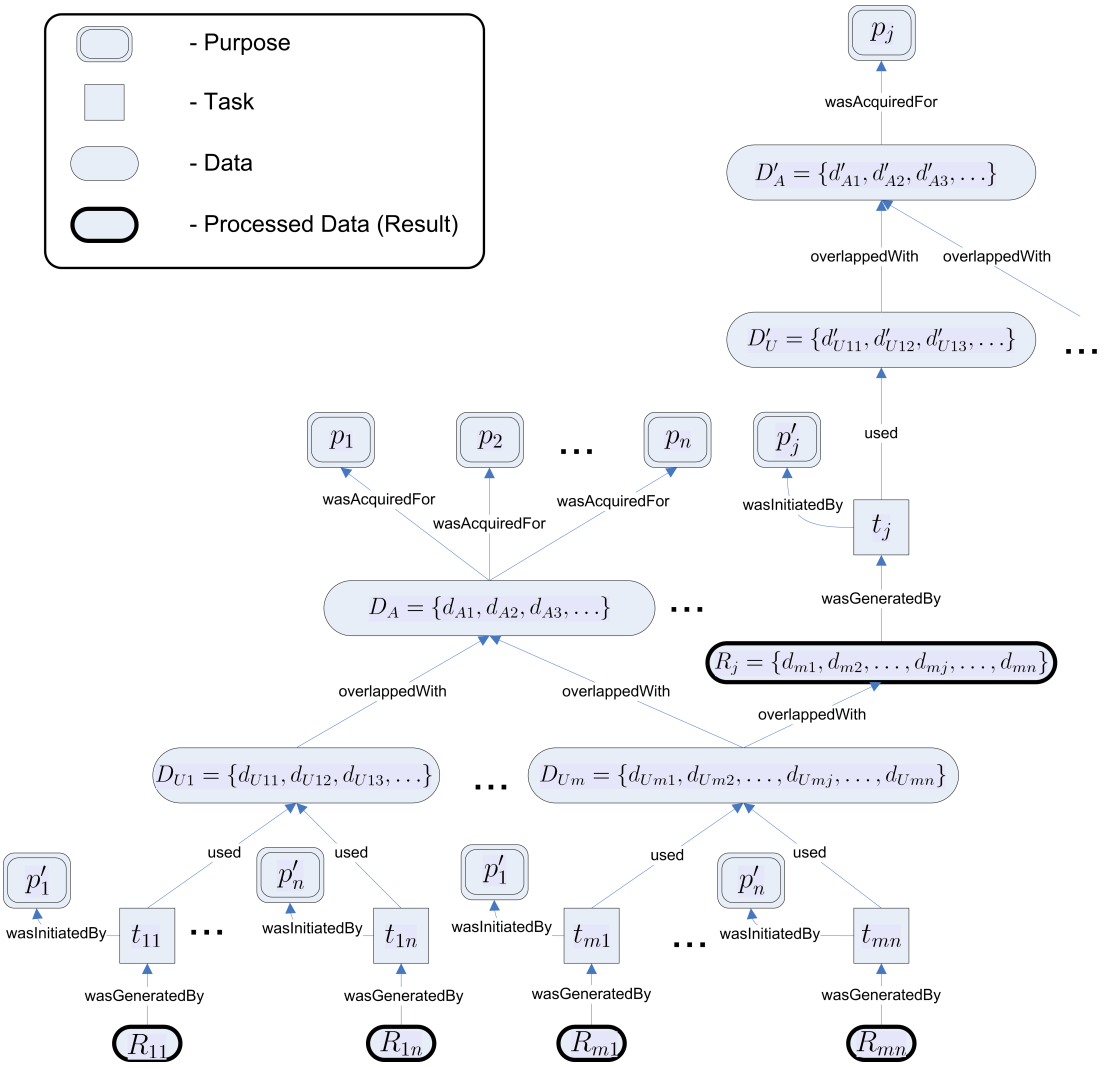


FIGURE 5.3: Processing View Graph G_W

Definition 5.20 (Task was initiated by Purpose)**.** Task t was initiated by purpose $p \in P_W$ if there exists $(t, p, \mathsf{wasInitiatedBy}) \in E_W$.

Definition 5.21 (Task used Used Data)**.** Task t used a multiset of used data instances $D_{Ui} \in D_{U_W}$ if there exists $(t, D_U, \mathsf{used}) \in E_W$.

Definition 5.22 (Used Data overlapped with Collected Data)**.** A multiset of used data instances D_{Ui} overlapped with a subset of collected data instances $D_A \in D_{A_W}$ if there exists $(D_{Ui}, D_A, \mathsf{overlappedWith}) \in E_W$.

Definition 5.23 (Used Data overlapped with a subset of Results)**.** A multiset of used data instances D_{Ui} overlapped with a subset of generated data instances $R' \in R_W$ if there exists $(D_{Ui}, R', \mathsf{overlappedWith}) \in E_W$ [1].

[1]The relationship overlappedWith is used in Definition 5.23 to indicate that one or more elements of D_{Ui} can appear one or more times in R'

Definition 5.24 (Collected Data was acquired for Purpose)**.** A multiset of collected data instances D_A was acquired for a set of purposes $P \in P_W$ if there exists $(D_A, P,$ wasAcquiredFor$) \in E_W$.

In summary, by following the arrows in Figure 5.3, Definition 5.19 indicates that a multiset of results R **was generated by** a task t; Definition 5.20 indicates that a task t **was initiated by** a purpose p; Definition 5.21 indicates that a task t **used** the multiset of data instances D_{Ui} in its execution; Definition 5.22 indicates that multiset of used data instances D_{Ui} **overlapped with** a subset of collected data instances D_A and Definition 5.23 indicates that the multiset of used data instances D_{Ui} **overlapped with** a subset of generated data instances R'. Finally, Definition 5.24 indicates that the multiset of data instances D_A **was acquired for** accomplishing a set of purposes P.

Similar to Section 5.2, we can use these definitions to define the property of well-formedness of a Processing View Graph:

Property 2 (Well-Formed Processing View Graph)**.** Let G_W be a Processing View Graph. G_W is a Well-Formed Processing View Graph if and only if,

a) For any $R \in R_W$ there exist one and only one $t_i \in T_W$, such that, R was generated by t.

b) For any $t \in T_W$ there exists one and only one $p \in P_W$ and one and only one $D_{Ui} \in D_{U_W}$, such that, t was initiated by purpose p and t used a multiset of data instances D_{Ui}.

c) For any $D_{Ui} \in D_{U_W}$ there exist one or more $D_A \in D_{A_W}$ and one or more $R' \in R_W$, such that, D_{Ui} overlapped with a subset of collected data instances D_A and D_{Ui} overlapped with a subset of generated data instances R'.

d) For any $D_A \in D_{A_W}$ there exist one or more $p \in P_W$, such that, D_A was acquired for a purpose p.

Property 2 indicates that a well-formed Processing View Graph can contain more than one collection purpose p_i for which a multiset of data instances is captured. The multiset of collected data (D_A) is divided in different subsets (D_{Ui}), each of which is used by one and only one task t to produce one and only one set of results R. Tasks are initiated by one purpose that is contained in the set of collection purposes. Note that elements of the multiset of collected data can be used in more than one multiset of used data. It is also important to mention that results can be reused and, therefore, be part of multisets of used data.

It is important to note that the algorithms we develop in Section 5.4 are only guaranteed to produce correct results on well-formed Processing View Graphs.

To illustrate the operation of this framework component, Figure 5.4 presents the Processing View related to the processing of Alice's personal information. There, we can see that Alice's personal information was collected for the `on-line sales` purpose. The set of data that was collected from her contains name, address, medicine's name and quantity. This data is subsequently used by two tasks: deliver medicine and manage stock, which use a subset of the collected information. As explained previously, the Processing View contains instances of data together with their class and type. This can be seen in Figure 5.4.

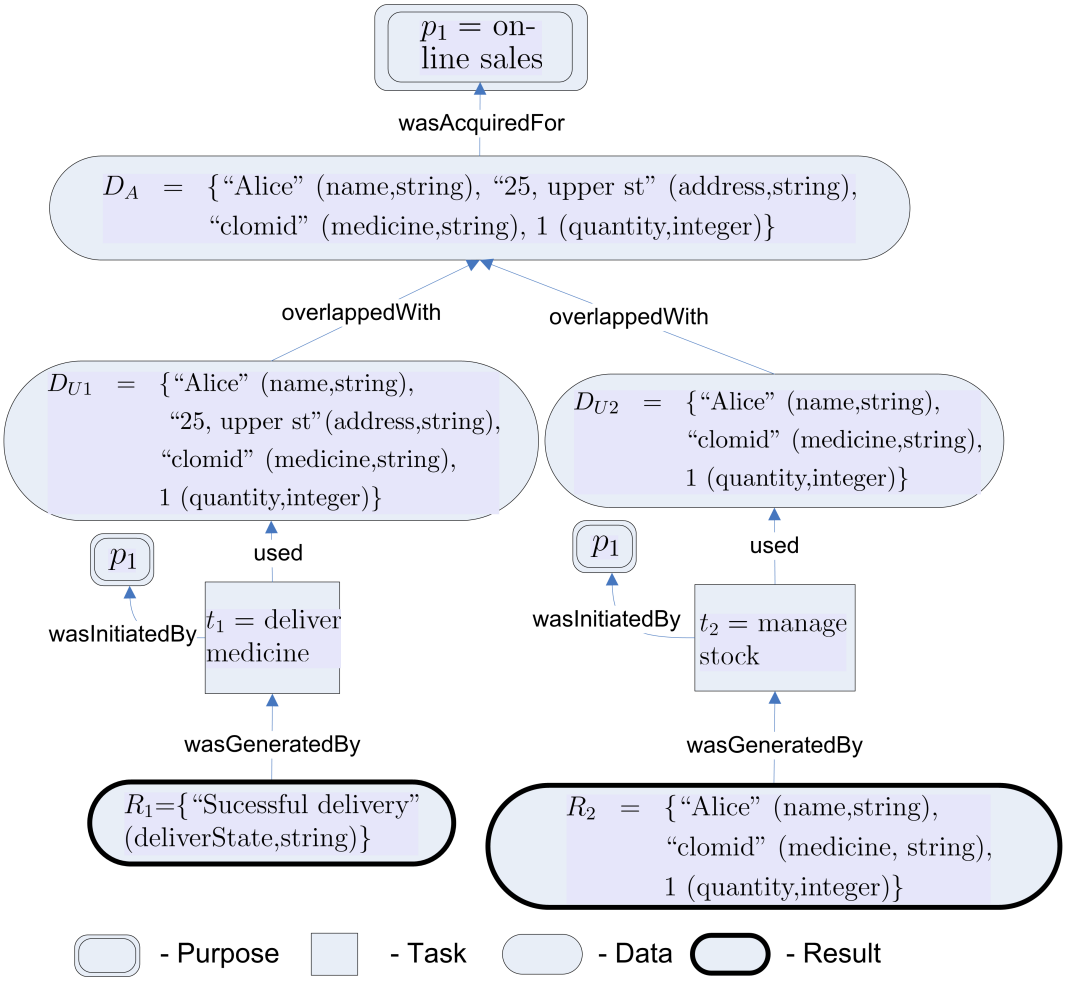


FIGURE 5.4: On-line Sales Example - Processing View

Next, we present the final component of the Compliance Framework, the Verification Algorithms. These algorithms analyse the Processing View to verify the compliance of the information processing with the Usage Rules Definition.

5.4 Verification Algorithms

Now that we have the Usage Rules Definition (a machine readable representation of processing rules) and the Processing View (a machine readable representation of how

data was used), we can develop algorithms to automatically verify that processing was performed in accordance with the Auditing Requirements.

These algorithms compare past processing, described in the Processing View, against the expected processing, represented by the Usage Rules Definition. To do this, provenance query results related to each audit requirement are extracted from the Processing View in the form of a subgraph. These subgraphs are obtained using the corresponding provenance queries explained in the tables presented in Section 4.2. Then, the data and the relationships of such these subgraphs are compared with the Usage Rules Definition. If these match we can conclude that the Auditing Requirements were satisfied.

In this section, we shall discuss each of these algorithms in turn. Specifically, for each Auditing Requirement described in Section 3.4, we develop one or more verification algorithms. Thus this Section is divided as follows:

- Section 5.4.1 discusses Requirement B: Purpose Compliance.
- Section 5.4.2 discusses Requirement C: Relevant Information Verification.
- Section 5.4.3 discusses Requirement F: Anonymity Preservation.
- Section 5.4.4 discusses Requirement G: Basic Security Characteristics Verification
- Section 5.4.5 discusses Requirement H: Information Transferred to a Secure Country.

5.4.1 Requirement B: Purpose Compliance

This requirement states that when performing a task, only the data that is strictly necessary to accomplish its initial stated purpose can be used. The verification of this requirement involves checking whether the correct class and type of data was used, whether data was used to accomplish the stated valid purposes and, if data is reused, whether processing was performed according to the stated purposes. From this, we derive three subrequirements which will be discussed in upcoming subsections: Used Data Compliance (B1), Purposes Validation (B2) and Reusing Results (B3). In what follows, we develop algorithms for verifying each subrequirement individually.

5.4.1.1 Subrequirement B1: *Used Data Compliance*

To verify the first subrequirement, Used Data Compliance, we need to check that the class and type of data used match those specified by the task, and that the purpose for which the task was initiated was correct. We also verify that the purpose and the task occur in the corresponding Usage Rules Definition. To do this, we focus on the

provenance of the result, which is a subgraph of the Processing View Graph, containing the ancestors of the node that represents the processing result. Specifically, we focus on the task that generated this result, the data this task used and the purpose for which the task was initiated. This subgraph, denoted with the symbol B_1 is called the *Used Data Compliance Subgraph*. This subgraph is defined below.

Definition 5.25 (Used Data Compliance Subgraph, $B_1(G_W, t_i)$)**.** A Used Data Compliance Subgraph is a subgraph of a well-formed Processing View Graph G_W for task $t_i \in G_W$, such that:

$$\begin{aligned} V_{B_1} &= \{p_i, D_{U_i}, t_i\} \\ E_{B_1} &= \{(t_i, p_i, \mathsf{wasInitiatedBy}), (t_i, D_{U_i}, \mathsf{used})\} \\ &\quad \text{where } p_i \in P_W, t_i \in T_W, D_{U_i} \in D_{U_W} \\ &\quad E_{B_1} \subseteq E_W, \text{ and } 0 < i \leq n. \end{aligned}$$

The general structure of a Used Data Compliance Subgraph $B_1(G_W, t_i)$ is presented in Figure 5.5. This structure encodes that a task t_i used a multiset of Used Data Instances D_{Ui}, and was initiated by the purpose p_i. Based on this, we can derive the following property of B_1:

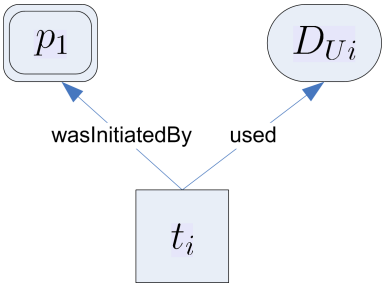


FIGURE 5.5: Subgraph B_1 of G_W

Property 3 (Well-Formed Set of Tasks)**.** Let T be set of tasks. T is well-formed in B_1 if, for all $t \in T$ Property 2.b holds.

At this point, we have extracted the required provenance information from the Processing View Graph in the form of the subgraph B_1. Now, we need to verify that the information processing was performed in accordance with the Usage Rules Definition Graph G_Γ, i.e. we need to compare the nodes and the relationships' names of B_1 against the ones stated in G_Γ.

By analysing B_1 against G_Γ, we can see that the set of nodes V_{B_1} contains *instances of data*, while the set of nodes V_Γ of G_Γ contains *classes* and *types* of data. Similarly, the set of edges, E_{B_1} contains relationships expressed in the past tense whilst E_Γ contains relationships expressed in the present tense (see Definitions 5.18 and 5.12).

The class and type of each instance created at execution time are collected at the same time as the provenance information and obtained using the accessors *class* and *type*. Thus, for each used data item present in V_{B_1} it is necessary to compare its class and type against the ones that appear in V_Γ. This comparison is straightforward in the case of purposes and tasks.

For the edges, we define the function $match(past, present)$ that given a relationship's name in past tense (contained in B_1) and another one in present tense (the one that should be present in G_Γ) returns *true* if both names are equivalent or $false$ if not.

All the characteristics that B_1 needs to exhibit to be in compliance with the *Used Data Compliance* subrequirement are summarised in the next property.

Property 4 (Data Processed According to a Valid Purpose)**.** Let B_1 be the subgraph defined in Definition 5.25, G_Γ be a Usage Rules Definition Graph, $C_{U_j} \in G_\Gamma$ a set of used data classes, $p_j \in G_\Gamma$ a purpose and $t_j \in G_\Gamma$ a task. $D_{U_i} \in B_1$ was processed by $t_i \in B_1$ according to a valid purpose $p_i \in B_1$ if

a) $p_i = p_j$,

b) $t_i = t_j$,

c) if for all $x \in D_{U_i}$, there exists one and only one pair $(cl, ty) \in C_{U_j}$ such that $class(x) = cl$ and $type(x) = ty$ and,

d) if for all $x \in Rel_{B_1}$, there exists one and only one $y \in Rel_\Gamma$ such that $match(x, y) =$ true.

Therefore, if Property 4 holds in subgraph B_1 of a Usage Rules Definition Graph G_Γ, we state that the processing result was obtained in compliance with the *Used Data Compliance* subrequirement. Algorithm 1 presents the verification process of Property 4.

The following example illustrates how the Used Data Compliance Subrequirement can be verified in the On-line Sales Scenario.

Example 5.1 (Used Data Compliance Subrequirement Example)**.** *First, we extract the subgraph $B_1(G_W, t_1)$ from the Processing View, which is related to the* `deliver medicine` *task, and is shown at the right of Figure 5.6. The left panel of the same figure shows the portion of the Usage Rules definition that contains the rules related to the* `deliver medicine` *task.*

Using Algorithm 1, we check whether data was processed according to a valid purpose. We can see that the purpose and the task contained in B_1 and G_Γ are the same. Therefore, the purpose and the task are valid. Then, we check the labels of the edges. We can see that all the past tense labels contained in B_1 have their present tense equivalent in G_Γ.

Algorithm 1 Data Processed According to a Valid Purpose

Input: $G_\Gamma = \{V_\Gamma, E_\Gamma, Rel_\Gamma\}, B_1(G_W, t_i) = \{V_{B_1}, E_{B_1}, Rel_{B_1}\}$
Output: 1 if graph B_1 is in compliance with the *Used Data Compliance* subrequirement or -1, -2, -3, -4 if it is not.

```
p_i ∈ V_B1, t_i ∈ V_B1, D_Ui ∈ V_B1, C_Uj ∈ V_Γ
if p_i ∈ V_Γ then
    if t_i ∈ V_Γ then
        for each x ∈ Rel_B1 and y ∈ Rel_Γ do
            if not(match(x, y)) then
                return -1                          ▷ Label not matched
            end if
        end for
        for each x ∈ D_Ui do
            if (class(x), type(x)) ∉ C_Uj then
                return -2                          ▷ Type not matched
            end if
        end for
    else
        return -3                                  ▷ Not registered task
    end if
else
    return -4                                      ▷ Not registered purpose
end if
return 1                                           ▷ Compliance
```

Finally, we check that the classes and types of instances of the used data set contained in B_1 correspond to the classes and types present in G_Γ.

G_Γ contains the set (`(name, string)`, `(address, string)`, `(medicine, string)`, `(quantity, integer)`), each element of which matches with the class and type of the instances contained in B_1. Therefore, we can conclude the data contained in D_{Ui} was processed by the `deliver medicine` task according to purpose p_1 and in compliance with the Used Data Compliance Subrequirement.

In the case of the `manage stock` task, we extract the subgraph $B_1(G_W, t_2)$, which is presented in Figure 5.7. In that subgraph, the purpose and the task are valid, as well as the labels of the edges. The Usage Rules Definition shows that the task `manage stock` should use the set `(medicine, string), (quantity, integer)`. However, the Processing View depicts that this task used an extra element: (name, string). Therefore, the processing presented in Figure 5.7 does not satisfy the Used Data Compliance Subrequirement since it is using more data than stated in the processing rules.

5.4.1.2 Subrequirement B2: *Purposes Validation*

The second subrequirement of the Purpose Compliance Requirement is Purposes Validation, which states that the processing purposes of a set of data were in accordance

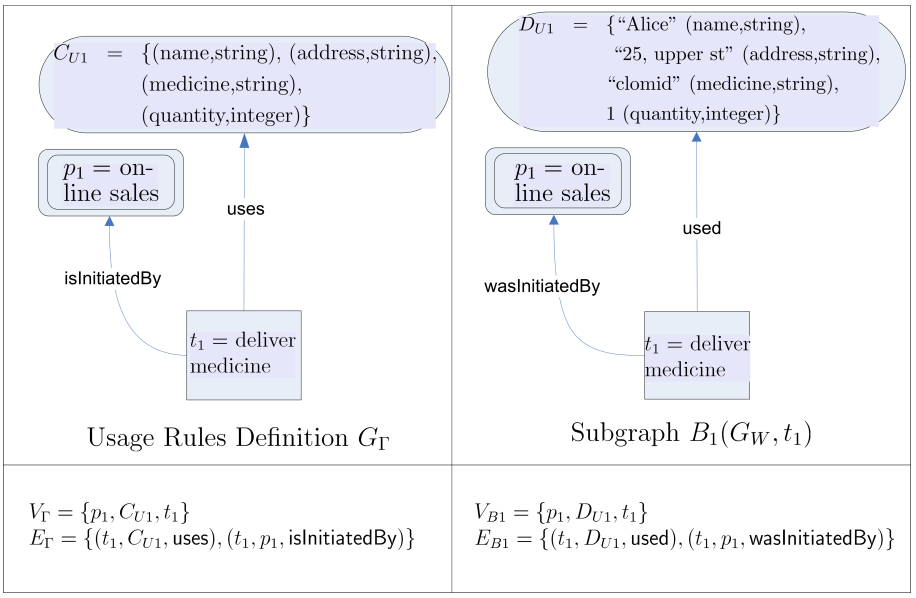


FIGURE 5.6: Used Data Compliance Verification Example: Task 1

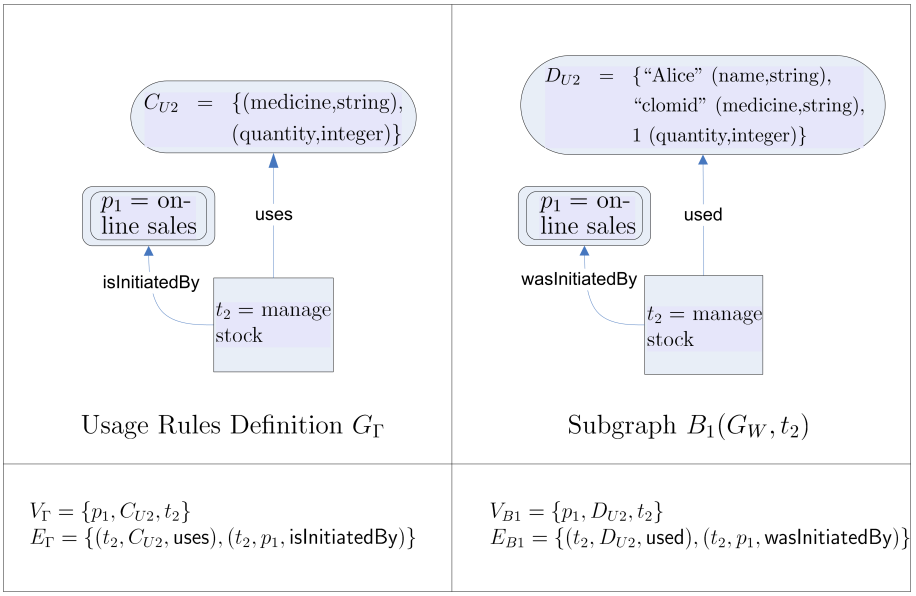


FIGURE 5.7: Used Data Compliance Verification Example: Task 2

with the collection purposes.

To verify this requirement, we start by extracting a subgraph from a Processing View Graph, which contains a set of tasks related to one set of collected data. Specifically, we focus on the purposes from which such tasks were initiated and the purposes from which the collected data was acquired. This is expressed as a subgraph B_2, which is called *Purposes Validation* subgraph, and is defined as:

Definition 5.26 (Purposes Validation Subgraph, $B_2(G_W, t_i)$)**.** Given a task t_i, a Purposes Validation Subgraph is a subgraph of a well-formed Processing View Graph G_W,

such that:

$$
\begin{aligned}
V_{B_2} &= \{p_i, D_A, D_{U_i}, t_i, p'_i\} \\
E_{B_2} &= \{(t_i, p'_i, \mathsf{wasInitiatedBy}), (t_i, D_{U_i}, \mathsf{used}), (D_{U_i}, D_A, \mathsf{overlappedWith}), \\
&\quad (D_A, p_i, \mathsf{wasAcquiredFor})\} \\
&\quad \text{where } p_i, p'_i \in P_W, t_i \in T_W, D_{U_i} \in D_W, D_A \in D_W, \\
&\quad E_{B_2} \subseteq E_W \text{ and } 0 < i \leq n
\end{aligned}
$$

The general structure of the Purposes Validation Subgraph $B_2(G_W, t_i)$ is presented in Figure 5.8, which shows the provenance graph of all the tasks performed over one set of collected data. In this graph, the execution of a task t_i was initiated by a purpose p'_i using the data D_U. At the same time, the set of Used Data D_U is a subset of the Collected Data D_A that was acquired for the purpose p_i.

Here, we are analysing the purposes related to the collection and processing of one set of data. However, more than one set of data can be collected from different entities. Therefore, in the verification of this subrequirement, all the sets of collected data should be checked. For clarity, in what follows we focus on a single set.

Before we introduce the verification algorithm, we first define the well-formedness property that needs to hold in order for the algorithm to produce meaningful results.

Property 5 (Well-formed Multiset of Collected Data)**.** Let D_A be a multiset of collected data. D_A is well-formed in B_2 if, for any D_A, Property 2.d holds.

In addition to Property 5, subgraph B_2 also exhibits Property 3, because B_2 contains a well-formed set of tasks T, since for any $t \in T$ was initiated by purpose p'. Such a p' is a *Processing Purpose*. Thus, as can be seen in Figure 5.8, the set P' containing all purposes p'_i is the set of Processing Purposes (Definition 5.2). According to Property 5, if B_2 contains a well-formed multiset of collected data D_A, this data was acquired for purposes $p_1, p_2, ..., p_n$, which are *Collection Purposes*. Hence, the set P containing all p_i is the set of Collection Purposes (Definition 5.1).

After both sets of purposes have been extracted, it is possible to check whether there exist some relations between them. Here, we define this relation as the subset operation, as can be seen in the next property.

Property 6 (Collection and processing justified by the same purposes)**.** Let P' be the set of Processing Purposes in B_2 and P be the set of Collection Purposes in B_2. Collection and processing are justified by the same purpose if $P' \subseteq P$

Finally, it is necessary to check that both sets contain valid purposes, i.e. they are contained in the Usage Rules Definition Graph. Therefore, we derive the next property:

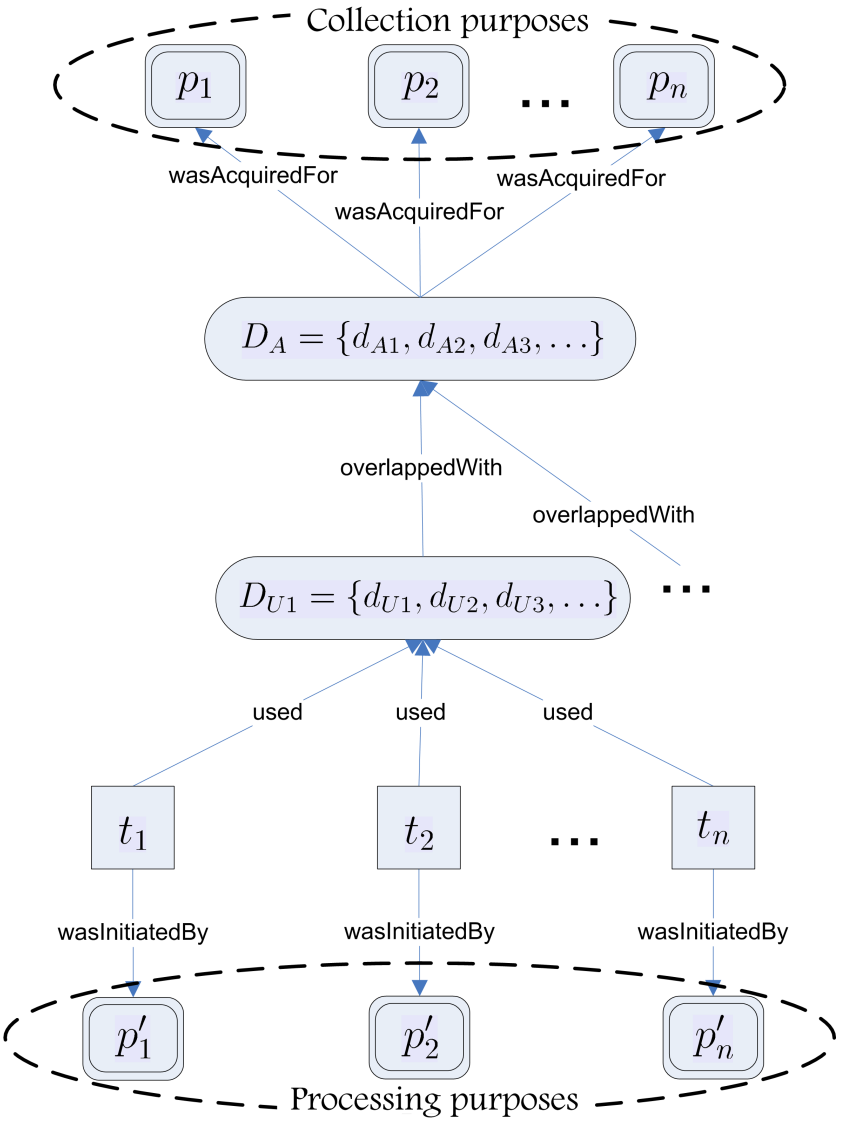


FIGURE 5.8: Subgraph B_2 of G_W

Property 7 (Validity of processing and collection purposes)**.** Let V_Γ be the nodes of the Usage Rules Definition Graph G_Γ, let P' be the set of processing purposes and P be the set of collection purposes extracted from B_2. The set of processing purposes and the set of collection purposes are valid if:

$$P' \subseteq V_\Gamma \text{ and } P \subseteq V_\Gamma$$

Therefore, when Properties 6 and 7 hold in subgraph B_2 of a Usage Rules Definition Graph G_Γ, we can say that subgraph B_2 is in compliance with the *Purposes Validation* subrequirement. The comparison process explained above is formalised in Algorithm 2.

The following example illustrates the process of verifying the Purposes Validation Subrequirement in the On-line Sales Scenario.

Algorithm 2 Purposes Validation

Input: $G_\Gamma = \{V_\Gamma, E_\Gamma, Rel_\Gamma\}, B_2(G_W, t_i) = \{V_{B_2}, E_{B_2}, Rel_{B_2}\}$
Output: 1 if B_2 is in compliance with the Purposes Validation Subrequirement, -2 or -1 otherwise.

```
P' contains all processing purposes p'_i in B_2
P contains all the collection purposes p_i in B_2
if P' ⊆ P then
    if P' ⊆ V_Γ ∧ P ⊆ V_Γ then
        return 1                                   ▷ Compliance
    else
        return -1                                  ▷ Not registered purpose
    end if
else
    return -2                                      ▷ Not compatible purpose
end if
```

Example 5.2 (Purposes Validation Subrequirement Example)**.** *First, we extract from the Processing View the subgraph $B_2(G_w, t_i)$ (where $1 \leq i \leq 2$), which is presented in Figure 5.9. According to Algorithm 2, we need to check that the Processing Purposes are contained in the Collection Purposes. In this example all the purposes are the same:* `on-line sales`*. Thus, we just need to check that this purpose is in the Usage Rules Definition. As the purpose is indeed contained in the Usage Rules Definition, we can say that the multiset of collected data D_A was used by the* `deliver medicine` *and the* `manage stock` *tasks in compliance with the Purposes Validation Subrequirement.*

Now, suppose that besides the tasks described previously, the on-line pharmacy also cross references customer information with personal data from job applicants. This search is conducted by the task `search job applicants` *that returns true if the name and date of birth of a customer match with the name and date of birth of a job applicant. As Figure 5.10 shows, this task is initiated by the purpose* `job application`*, which is not related to the collection purpose (*`on-line sales`*) and the purpose stated in the Usage Rules Definition (also,* `on-line sales`*). The reason is that Alice disclosed her information to accomplish the* `on-line sales` *purpose not the* `job application` *purpose. Thus, we can conclude that the* `search job applicants` *task is not in compliance with the Purposes Validation Subrequirement.*

5.4.1.3 Subrequirement B3: *Reusing Data*

The subrequirements in Sections 5.4.1.1 and 5.4.1.2 are used to verify the purposes related to data that was collected and processed by the same entity. However, an entity may use data generated by tasks executed by other entities. This data is referred to as *reused data*, which has its own processing purpose related to the task that produced it. To be in compliance with Requirement B, this processing purpose should be in

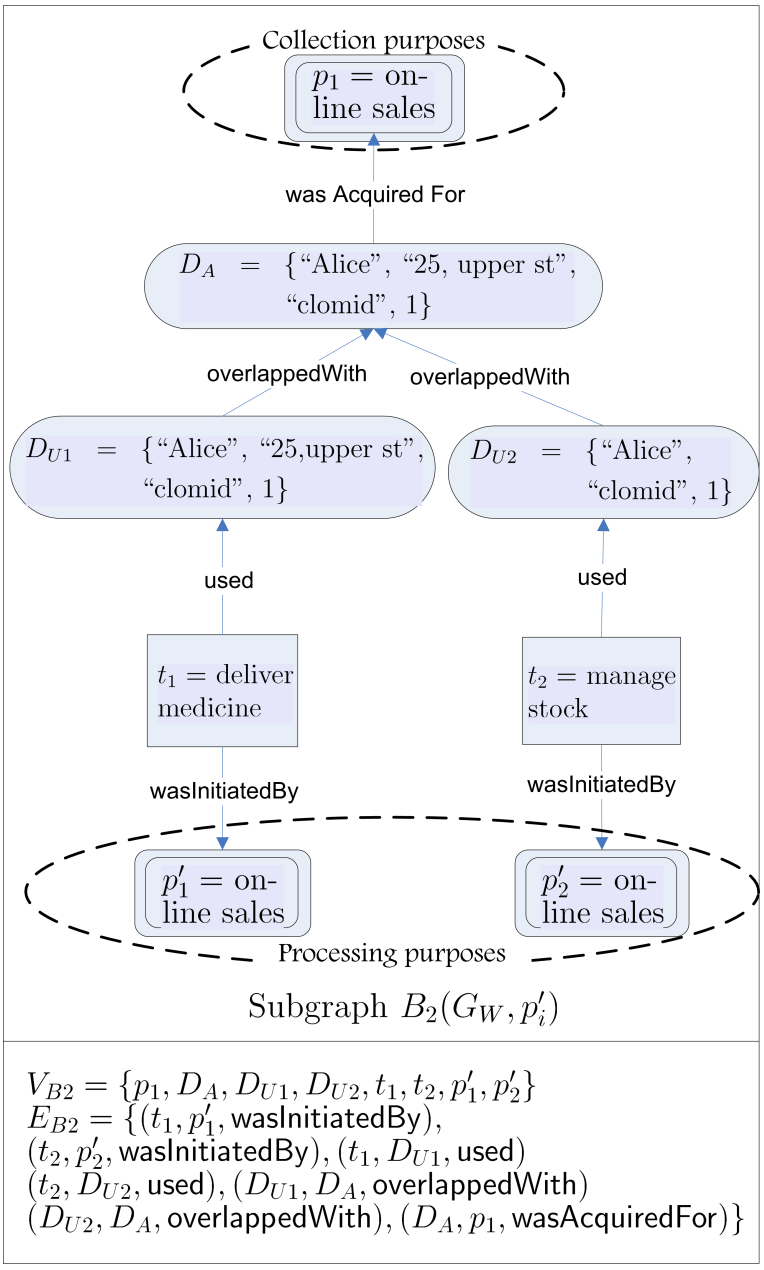


FIGURE 5.9: Purposes Validation Example 1

accordance to the purposes for which data was reused. This requirement is formalised in Subrequirement B3: Reusing Data.

To verify this subrequirement, we first need to identify reused data. This data resulted from a task executed by a different data controller than the one that uses it. Specifically, we focus on the task that generated reused data and the purpose from which such a task was initiated. For the sake of clarity, we focus on one reused result contained in one multiset of used data. Nevertheless, all the used data multisets related with one entity that contain reused data should be checked. The relations between Reused Data, the

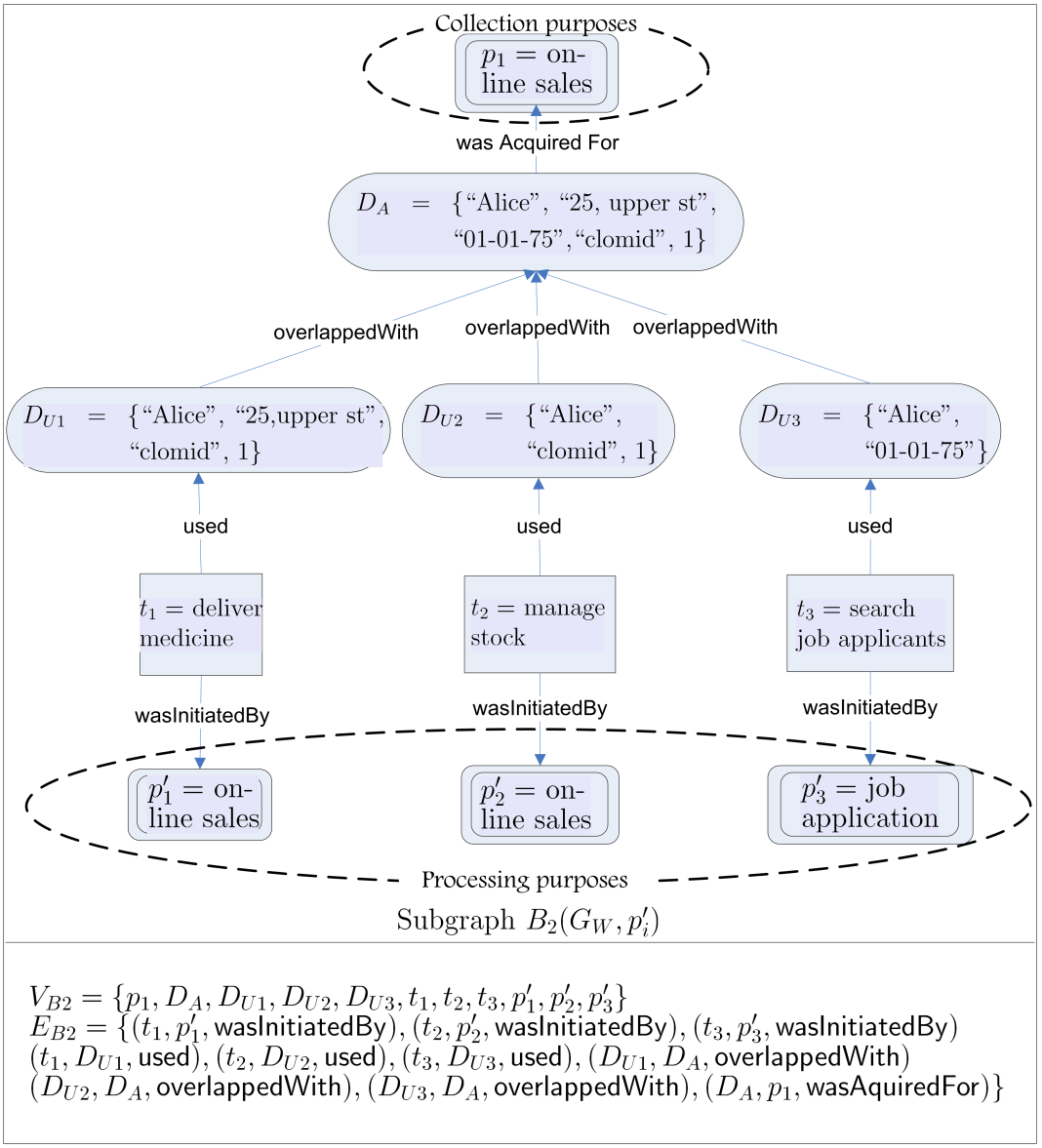


FIGURE 5.10: Purposes Validation Example 2

tasks that processed it and the purpose for which is was processed are expressed as a subgraph B_3, which is called *Reusing Data Subgraph*:

Definition 5.27 (Reusing Data Subgraph, $B_3(G_W, R_j)$)**.** A Reusing Data Subgraph B_3 is a subgraph of a well-formed Processing View Graph G_W consisting of results R_j, such that,

$$\begin{aligned} V_{B_3} &= \{R_j, t_j, p'_j\} \\ E_{B_3} &= \{(R_j, t_j, \mathsf{wasInitiatedBy}), (t_j, p'_j, \mathsf{wasGeneratedBy})\} \\ &\quad \text{where } p_j \in P_W, t_j \in T_W, R_j \in R_W, \\ &\quad E_{B_3} \subseteq E_W \text{ and } 0 < i \leq n \end{aligned}$$

The general structure of a Reusing Data Subgraph $B_3(G_W, R_j)$ is presented in Figure 5.11 (inside of the dotted-line rectangle) showing that a result R_j, which was reused in the multiset of used data D_{U_m} as d_{mj}, was generated by a task t_j that was initiated by a purpose p'_j. In this figure, $d_{m1} = d_{Um1}, d_{m2} = d_{Um2}, \ldots, d_{mn} = d_{Umn}$. Therefore, the extra data in D_{Um} is d_{mj}.

Similar to Section 5.4.1.2, we derive the following property of well-formedness.

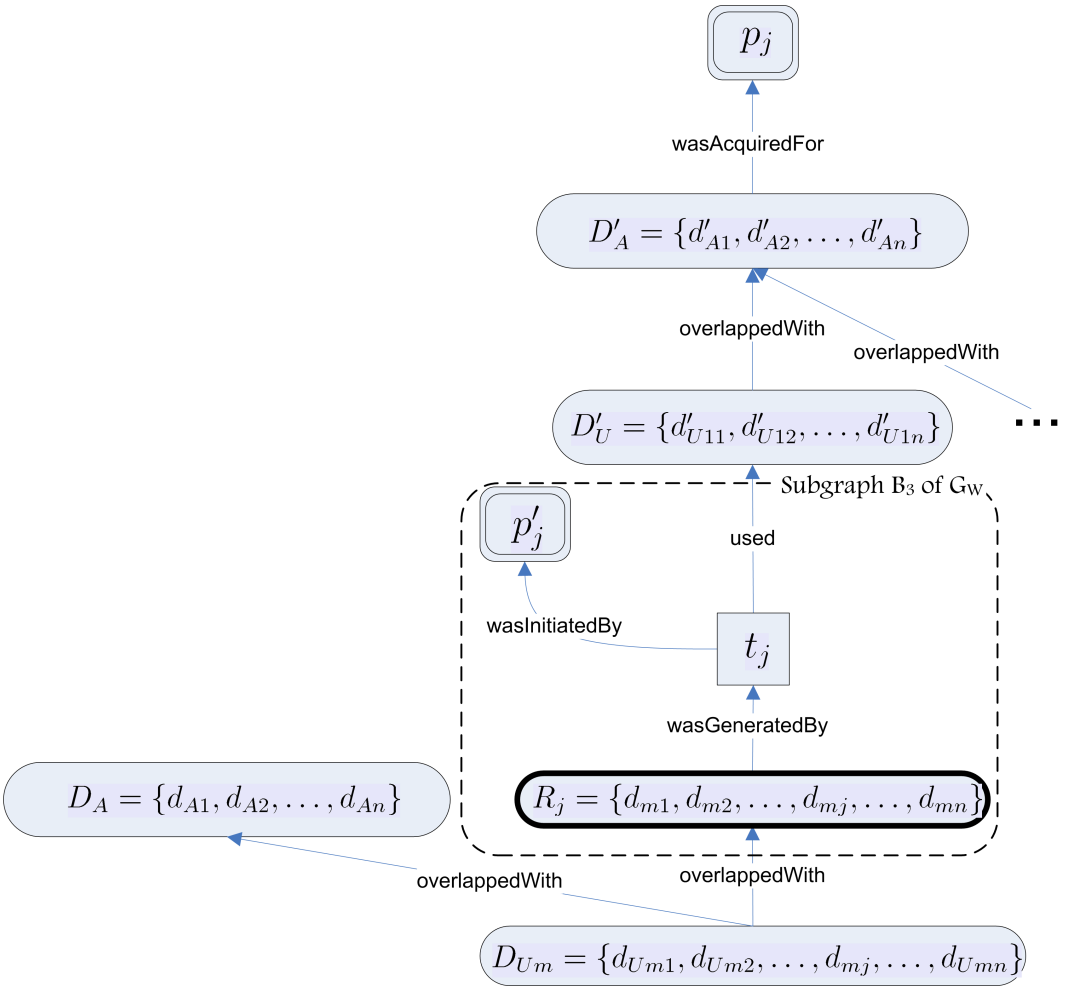


FIGURE 5.11: Subgraph B_3 of G_W

Property 8 (Well-formed Multiset of Results)**.** Let R be a multiset of results. R is well-formed in B_3 if for any $R \in B_3$ Property 2.a holds.

By using Property 8 and Property 3 of B_3, it is possible to obtain the task $t_j \in B_3$ that generated the result we are interested in and the purpose $p'_j \in B_3$ that initiated this task. After that, p'_j can be compared to the purposes for which the used data multiset D_U was processed. To obtain the processing purposes related to D_U we use Property 3 and 5. In that way, P' contains the set of processing purposes and P the set of collection purposes. Therefore, we just need to verify that both sets contain p'_j by using the next property.

Property 9 (Purpose Compatibility)**.** Let P' be a set of Processing Purposes and P be a set of Collection Purposes. A purpose $p' \in B_3$ is compatible with $P' \in V_{B_2}$ and $P \in V_{B_2}$ if

$$p' \in P' \text{ and } p' \in P$$

In this case, p'_j is not checked against the Usage Rules Definition Graph because P' and P were already checked in the subrequirement *Purpose Validation* ($B_2(G_W, t_i)$). Thus, if p'_j is contained in both sets, it is also contained in the Usage Rules Definition Graph, therefore, is a valid purpose.

If Property 9 holds in subgraph B_3 of G_W, we can say that such a subgraph is in compliance with the *Reusing Data* subrequirement. The verification process of this subrequirement is presented in Algorithm 3.

Algorithm 3 Reusing Data

Input: $G_\Gamma = \{V_\Gamma, E_\Gamma, Rel_\Gamma\}, B_3(G_W, R_j) = \{V_{B_3}, E_{B_3}, Rel_{B_3}\}$
Output: 1 if B_3 was in compliance with The Reusing Data Subrequirement, and -1 otherwise.

```
P' ⊆ G_Γ, P ⊆ G_Γ, p ∈ V_B3
if p ∈ P' and p ∈ P then
    return 1                                  ▷ Compatible purpose
else
    return -1                                 ▷ Not compatible purpose
end if
```

The following example illustrates the verification of the Reusing Data Subrequirement in the On-line Sales Scenario.

Example 5.3 (Reusing Data Subrequirement Example)**.** *To demonstrate how the Reusing Data Subrequirement can be verified, we extend the initially presented Processing View (Figure 5.3) by adding the reusing of results by the Human Resources Department. After Alice's information is processed, the pharmacy decided to share the results of the* `manage stock` *task (a sales report) with this department. When the Human Resources Department receives the sales report, it performs a search to find matches between job applicants' names and customers' names. The result of this search is used to infer information about job applicants' circumstances that can be costly to the company, such as chronic diseases or pregnancy. This information can be later used to take decisions regarding the hiring of applicants. In Alice's case, however, when she disclosed her information she did it with the purpose of buying her fertility drug on-line, not for it to be investigated as part of a job application.*

To verify this requirement, Figure 5.12 shows subgraph $B_3(G_W, R_2)$ expressing that some data items contained in the sales report (name and medicine's name) that were generated by the `manage stock` *task were reused by the* `job application` *task. However, this*

report used data that was collected for the purpose of p_1 = `on-line sales`*, as the corresponding* G_Γ *indicates. Therefore, reusing this result for the purpose of* p_2 = `job application` *is not in compliance with the Reusing Data Subrequirement, since* p_2 *is neither contained in the collection nor the processing purposes.*

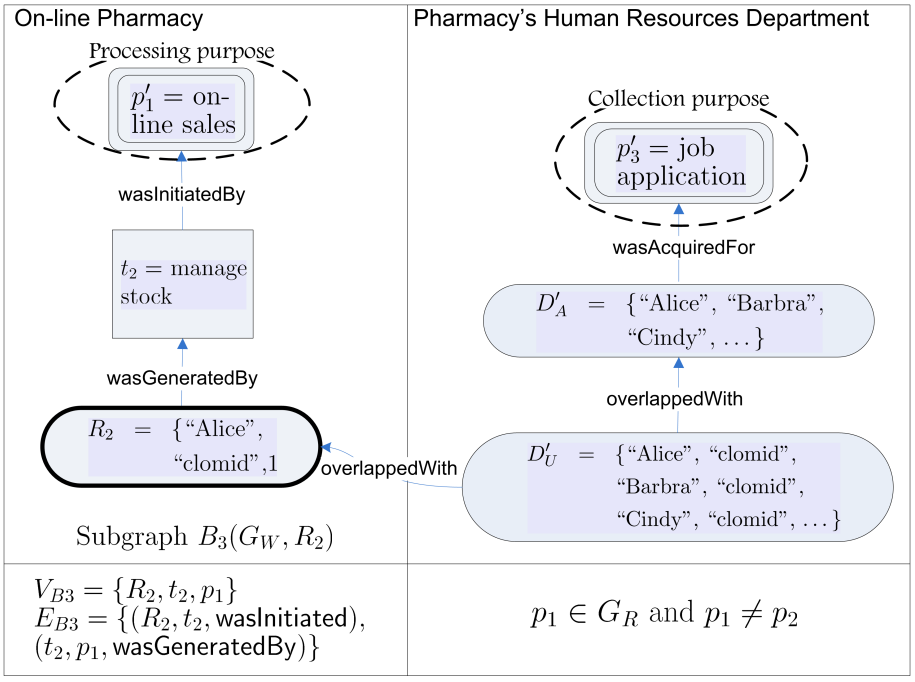


FIGURE 5.12: Reusing Data Example

This concludes the verification of Requirement B. Specifically, if all subrequirements discussed above are satisfied, we can say that the processing of personal information is in compliance with Requirement B. Next, we focus on the verification of Requirement C.

5.4.2 Requirement C: Relevant Information Verification

The Relevant Information Verification Requirement states that all the collected data should be used in processes initiated by the collection purpose. So, if some data was collected and not used, it means that more data than was strictly necessary was collected and, therefore, the process does not satisfy this requirement.

To verify compliance with this requirement, the multisets of used data instances as well as the multiset of collected data instances should be compared against the Usage Rules Definition. However, in subrequirement B1 (*Used Data Compliance*) the used data multisets are already compared against the Usage Rules Definition. Thus, here we only compare the multiset of collected data instances against the set of collected data classes.

To do this, we first extract from a Processing View Graph the provenance graph of all multisets of used data that are *overlapped with* one multiset of collected data, which was captured from one specific Data Subject. Then, we focus on the elements of collected data and the used data multisets. For the sake of clarity, we analyse one set of collected data, however, to verify compliance with this requirements, all sets of collected data should be checked. The relation between collected data and used data is expressed as a subgraph C that is called *Relevant Information* Subgraph and is defined below.

Definition 5.28 (Relevant Information Subgraph, $C(G_W, D_{U_i})$)**.** A Relevant Information Subgraph is a subgraph of a well-formed Processing View Graph G_W restricted to a set of used data D_{U_i} such that:

$$\begin{aligned} V_C &= \{D_A, D_{U_i}\} \\ E_C &= \{(D_{U_1}, D_A, \mathsf{overlappedWith}), (D_{U_2}, D_A, \mathsf{overlappedWith}), ..., \\ &\quad (D_{U_m}, D_A, \mathsf{overlappedWith})\} \\ &\quad \text{where } D_{U_i} \in D_W, D_A \in D_W, E_C \subseteq E_W \text{ and } 0 < i \leq m \end{aligned}$$

Hence, we derive the next property from this subgraph:

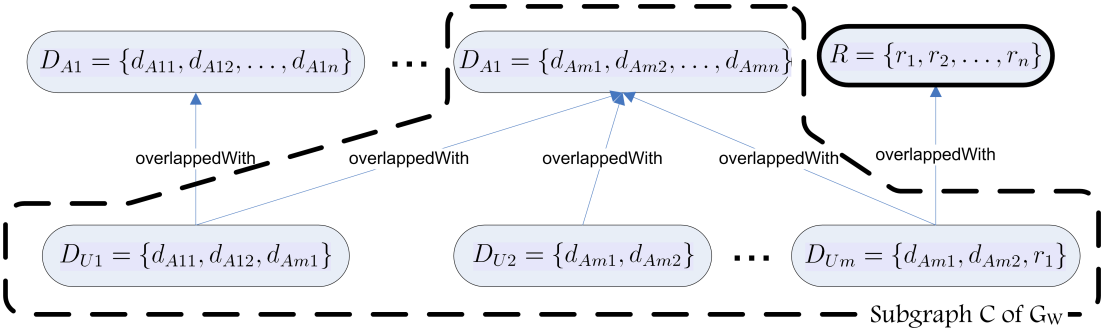


FIGURE 5.13: Subgraph C of G_W

Property 10 (Well-formed Multiset of Used Data)**.** Let D_U be a multiset of used data. D_U is well-formed in C if for any D_{U_i} Property 2.c holds.

Then, by using Property 10, we can extract the multisets of collected data instances D_A contained in subgraph C. After that, we compare the class and type of each element of D_A against the data classes that should be collected. These data classes are contained in G_Γ, specifically in C_Q. This is formalised in the following property.

Property 11 (All Collected Data was Used)**.** Let $D_A \in G_W$ be a multiset of collected data instances and $C_Q \in G_\Gamma$ a set of collected data classes. For each $x \in D_A$ there exists one and only one pair $(cl, ty) \in C_Q$ such that $class(x) = cl$ and $type(x) = ty$.

Therefore, if Property 11 holds in a subgraph C, we can conclude this subgraph is in compliance with the *Relevant Information Verification Requirement*. The verification process of this requirement is formalised in Algorithm 4.

Algorithm 4 Relevant Information Verification

Input: $G_\Gamma = \{V_\Gamma, E_\Gamma, Rel_\Gamma\}, C(G_W, D_{U_i}) = \{V_C, E_C, Rel_C\}$

Output: 1 if C is in compliance with the Relevant Information Verification Requirement, and -1 otherwise.

$C_Q \in V_\Gamma, D_A \in V_C$

if $\forall x \in D_A : (class(x), type(x)) \in C_Q$ **then**

  return 1 ▷ All collected data was used

else

  return -1 ▷ Not all collected data was used

end if

The following example illustrates the verification of the Relevant Information Verification Requirement in the On-line Sales Scenario.

Example 5.4 (Relevant Information Verification Requirement Example)**.** *To verify the Relevant Information Verification Requirement in the on-line sales scenario, we extract subgraph $C(G_W, D_{U_i})$ (where $1 \leq i \leq 2$) from the Processing View. This subgraph is presented in Figure 5.14. According to Algorithm 4, we need to check that the class and type of each of the elements of D_A are contained in the corresponding C_Q. Thus, we extract multisets D_A = { "Alice" (name, string), "25 upper st" (address, string), "clomid" (medicine, string), 1 (quantity, integer)} and C_Q = { (name, string), (address, string), (medicine, string), (quantity, integer)}, in which the class and type of each element of D_A is contained in C_Q. To clarity the information that we are analysing, Figure 5.14 does not show the classes and types of D_A. Instead, these can be seen in the corresponding Processing View (Figure 5.4). Therefore, assuming that all the tasks from which the pharmacy collected Alice's information were executed, we can conclude that the collection of information performed by the pharmacy is in compliance with the Relevant Information Verification requirement.*

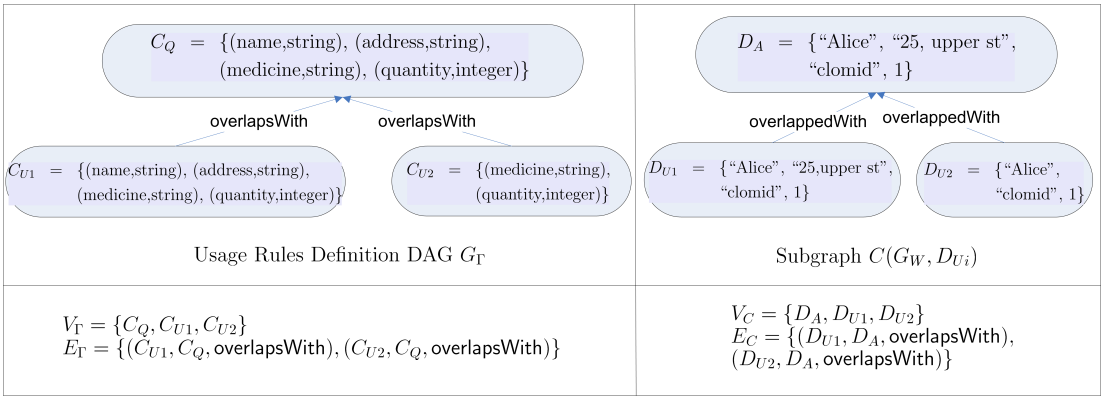


FIGURE 5.14: Relevant Information Verification Example 1

To illustrate a case of non-compliance, suppose that the on-line pharmacy also collected the nationality of Alice. Figure 5.15 shows the subgraph $C(G_W, D_{U_i})$ in which the item ***''British'' (nationality, string)*** *has been collected from Alice. Considering the purpose for which Alice's information is collected, there is no reason why the pharmacy*

asks for Alice's nationality. Thus, such an item does not appear in the corresponding Usage Rules Definition. This problem is exposed when the subgraph C is compared to this Usage Rules Definition. Therefore, in this specific case, we can conclude that the collection of information performed by the pharmacy is not in compliance with the Relevant Information Verification requirement.

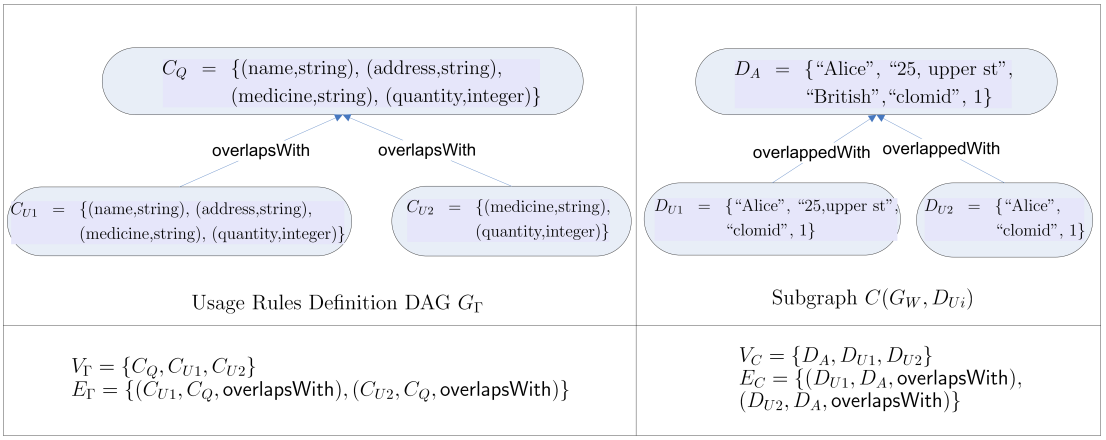


FIGURE 5.15: Relevant Information Verification Example 2

5.4.3 Requirement F: Anonymity Preservation

The Anonymity Preservation Requirement states that information that can be used to identify a specific individual should not be presented in a result of processing personal information. As previously discussed, there are different approaches to satisfy this property. Thus, after using an anonymisation technique, processing results contain two types of information: non-sensitive information and identifiers that reference to sensitive information.

To verify this requirement, we extract from a Processing View Graph a multiset of results related to one specific task. The class and type of the elements of this result are compared with the ones contained in the corresponding set of generated data classes from the Usage Rules Definition Graph. For clarity, we analyse one multiset of results, however, each multiset of results contained in the Processing View should be checked. The subgraph F of the Processing View that needs to be considered to verify this requirement is called the *Anonymity Preservation Subgraph* and is defined below.

Definition 5.29 (Anonymity Preservation Subgraph, $F(G_W, R_i)$)**.** A Anonymity Preservation Subgraph is a subgraph of a well-formed Processing View Graph G_W, which is focused on a set of results R_i, such that:

$$\begin{aligned} V_F &= \{R_i, t_i\} \\ E_F &= \{(R_i, t_i, \text{wasGeneratedBy})\} \\ &\quad \text{where } R_i \in R_W, t_i \in T_W, E_F \subseteq E_W \text{ and } 0 < i \leq n \end{aligned}$$

The general structure of an Anonymity Preservation Subgraph $F(G_W, R_i)$ is presented in Figure 5.16 showing that a result R_i was generated by a task t_i. By using Property 8, which holds in F, it is possible to obtain a multiset of results R_i, which is to be compared with the corresponding set of generated data classes from the Usage Rules Definition. This comparison is made using the following property:

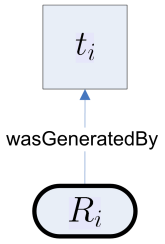


FIGURE 5.16: Subgraph F of G_W

Property 12 (Results' Anonymisation)**.** Let R_i be a multiset of generated data instances and $C_{\mathcal{G}_i}$ a set of generated data classes. For each $x \in R_i$ there exists one and only one pair $(cl, ty) \in C_{\mathcal{G}_i}$ such that $class(x) = cl$ and $type(x) = ty$.

If Property 12 holds in a subgraph F of G_W, we can conclude that this subgraph is in compliance with the *Anonymity Preservation* Requirement. The verification process of this requirement is formalised in Algorithm 5.

Algorithm 5 Anonymity Preservation of Results

Input: $G_\Gamma = \{V_\Gamma, E_\Gamma, Rel_\Gamma\}, F(G_w, R_i) = \{V_F, E_F, Rel_F\}$
Output: 1 if F is in compliance with the Anonymity Preservation Requirement or -1 otherwise.

```
C_{G_i} ∈ V_Γ, R_i ∈ V_C
if ∀x ∈ R_i : (class(x), type(x)) ∈ C_{G_i} then
    return 1                                  ▷ All data was anonymised
else
    return -1                                 ▷ Not all data was anonymised
end if
```

The following example illustrates the verification of the Anonymity Preservation Requirement in the On-line Sales Scenario.

Example 5.5 (Anonymity Preservation Requirement Example)**.** *To verify the Anonymity Preservation Requirement, it is necessary to extract two subgraphs, each of which is related to the two results that were produced in Figure 5.4. Then, to verify the* `delivery state` *result, we extract from the Processing View the subgraph $F(G_\Gamma, R_1)$, which is presented in Figure 5.17. In this figure, the task* `deliver medicine` *produces the result "Successful delivery". If we compare this subgraph to the Usage Rules Definition, we can see that the class and type of the elements of R_1 match with the ones that appear in $C_{\mathcal{G}_1}$. In that case, we can say that the processing of Alice's information was in compliance with the Anonymity Preservation Requirement, as the result produced by the* `deliver medicine` *task does not expose any information that can used to identify Alice.*

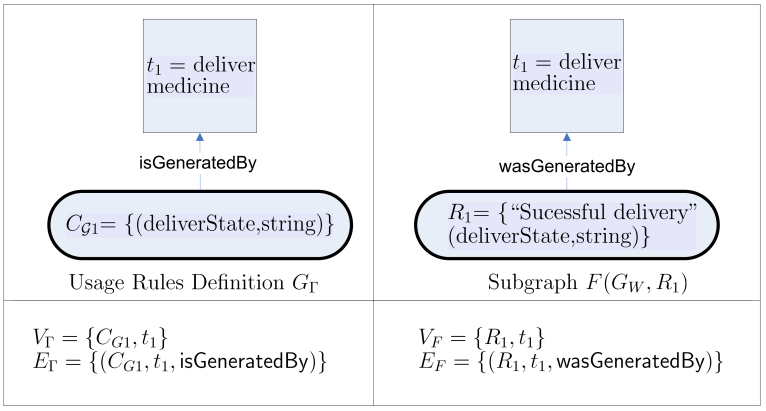


FIGURE 5.17: Anonymity Preservation Requirement Example: Result 1

Regarding the second result, we extract the subgraph $F(G_\Gamma, R_2)$ from the Processing View (see Figure 5.18). In this subgraph, the task `manage stock` *produces a sales report containing a name, a medicine name and a quantity. If we compare this subgraph to the Usage Rules Definition, we can see that the class and type of the elements of R_2 do not match with the ones contained in C_{G_2}: (name, string) should not have been used. The reason for this is that an sales report should not contain the name of the customers. Rather, it only needs the quantity sold of each item. Thus, in this case, we can say that the processing of Alice's information was not in compliance with the Anonymity Preservation Requirement, and therefore, such an inventory could be used against her interest, as shown in Section 5.3.*

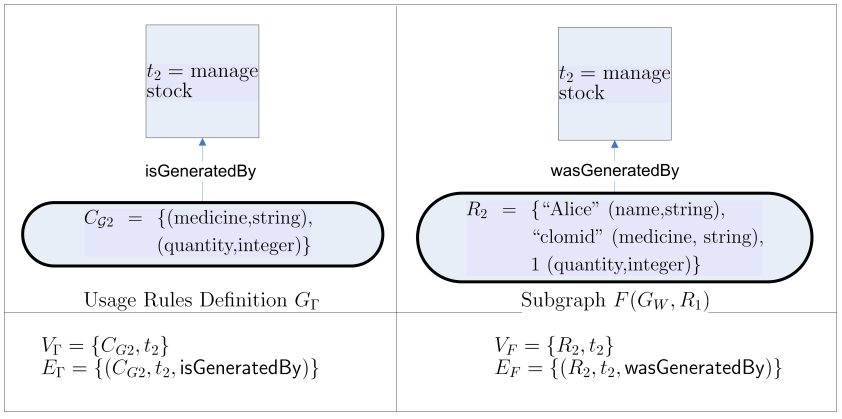


FIGURE 5.18: Anonymity Preservation Requirement Example: Result 2

5.4.4 Requirement G: Basic Security Characteristics Verification

The Basic Security Characteristics Verification Requirement states that all the data transferred between entities should be transmitted over secure communication channels. As discussed in Section 3.4.9, we focus on the implementation of the confidentiality, integrity and non-repudiation security properties (recall that one of the assumptions made in Section 3.6 assumes access control is implemented). It is important to note

that this requirement is not expressed in the Usage Rules Definition Graph, since it only specifies which operations related to the processing of application data can be performed.

As discussed in Section 4.4.4, the required security properties can be supported by encryption/decryption and digital signature schemes. The difference between the provenance DAG presented in Section 4.4.4 and the one we create here is that the applied security functions can be seen as tasks applied to data following a specific pattern instead of relationships. Since we are concerned about secure transportation, we focus on the security functions applied to data before it was sent and after it was received. To do this, we define a group of tasks indicating the implementation of such schemes.

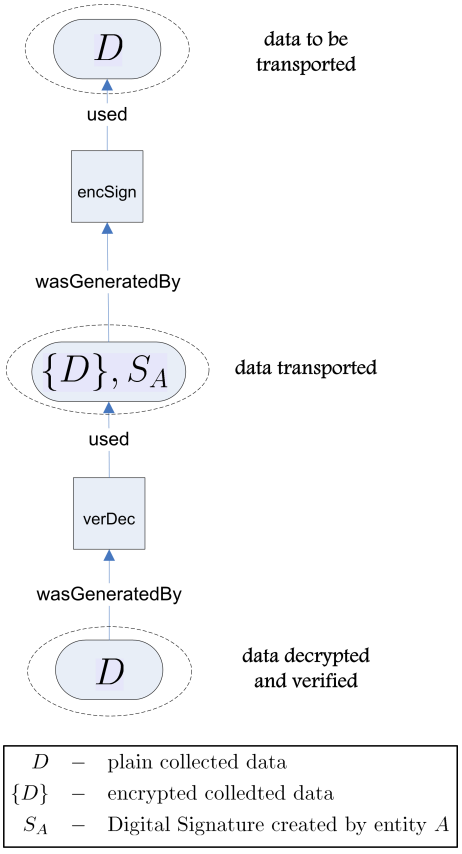


FIGURE 5.19: Subgraph G of G_W

As discussed in Section 4.1.3, data is encrypted and signed before being sent to an entity. Hence, a piece of encrypted data along with its corresponding signature is transported. When this piece of data is received, the data is decrypted and its signature is verified. The task that encrypts and creates the signature of a piece of data is denoted as $t_{ES} = encSign$, which is equivalent to the **encryptedSigned** relationship presented in Figure 4.11. Similarly, the task that decrypts a piece of data and checks its signature is denoted as $t_{VD} = verDec$, which is equivalent to the **verifiedDecrypted** relationship presented in Figure 4.11.

To verify this requirement, we extract from a Processing View DAG a subgraph containing a set of data that was transmitted between entities (collected data, used data or reused data). When in its plain representation, this set is represented as D, and as $\{D\}$ when it is encrypted. Finally, S_D expresses the signature related to the same piece of data. This extracted information is expressed as a subgraph G, which is called *Security Subgraph*. This subgraph is defined as:

Definition 5.30 (Security Subgraph, $G(G_W, D)$)**.** A Security Subgraph is a subgraph of a well-formed Processing View Graph G_V pertaining to a set of data D such that:

$$
\begin{aligned}
V_G &= \{D, (\{D\}, S_A), t_{ES}, t_{VD}\} \\
E_G &= \{(D, t_{VD}, \mathsf{wasGeneratedBy}), (t_{VD}, (\{D\}, S_A), \mathsf{used}), \\
&\quad ((\{D\}, S_A), t_{ES}, \mathsf{wasGeneratedBy}), (t_{ES}, D, \mathsf{used})\} \\
&\quad \text{where } D \text{ is any data set in } D_W, \{D\} \text{ is the encrypted version of } D, \\
&\quad S_A \text{ is the signature associated with } D, t_{ES} = \mathsf{encSign} \in T_W, \\
&\quad t_{VD} = \mathsf{verDec} \in T_W, V_G \subseteq V_W \text{ and } E_G \subseteq E_W.
\end{aligned}
$$

If the tasks t_{ES} and t_{VD} are present in the past processing of information, it means that encryption/decryption and digital signature schemes were used to protect the transported data. The following property verifies that data was protected:

Property 13 (Secured Data)**.** A multiset $D \in G$ was transported securely if,

$$
\begin{gathered}
\text{Exist } t_{ES}, t_{VD} \in G \text{ such that,} \\
(D, t_{VD}, \mathsf{wasGeneratedBy}) \in G,\ (t_{ES}, D, \mathsf{used}) \in G, t_{ES} = \mathsf{encSign} \text{ and } t_{VD} = \mathsf{verDec}
\end{gathered}
$$

The verification process of this requirement is formalised in Algorithm 6.

Algorithm 6 Basic Security Characteristics Verification

Input: $G(G_W, D) = \{V_G, E_G, Rel_G\}$
Output: 1 if G satisfies the Basic Security Characteristics Verification Requirement, -1 otherwise.

```
D ∈ G_W, t_ES = encSign, t_VD = verDec
if (D, t_VD, wasGeneratedBy) ⊆ E_G and (t_ES, D, used) ⊆ E_G then
    return 1                                   ▷ D was securely transported
else
    return -1                                  ▷ D was not securely transported
end if
```

The following example illustrates the verification of the Basic Security Characteristics Verification Requirement in the On-line Sales Scenario.

Example 5.6 (Basic Security Characteristics Verification Requirement Example)**.** *To verify the Basic Security Characteristics Verification Requirement, we extract from the*

Processing View the subgraph $G(G_W, R_2)$, which is presented in Figure 5.20. This graph shows that after the stock manager executed the `manage stock` *task to obtain sales report R_2, this report was encrypted, signed and sent to the on-line pharmacy. After being received by the pharmacy, the signature is verified, the data is decrypted, and obtained in its plain form. By analysing G, we can conclude that the set R_2 was transported in a secure way since the tasks t_{ES} = encSign and t_{VD} = verDec are present, which indicates the implementation of encryption/decryption and digital signature schemes. Thus, we can say that the R_2 was transported in compliance with the Basic Security Characteristics Verification Requirement.*

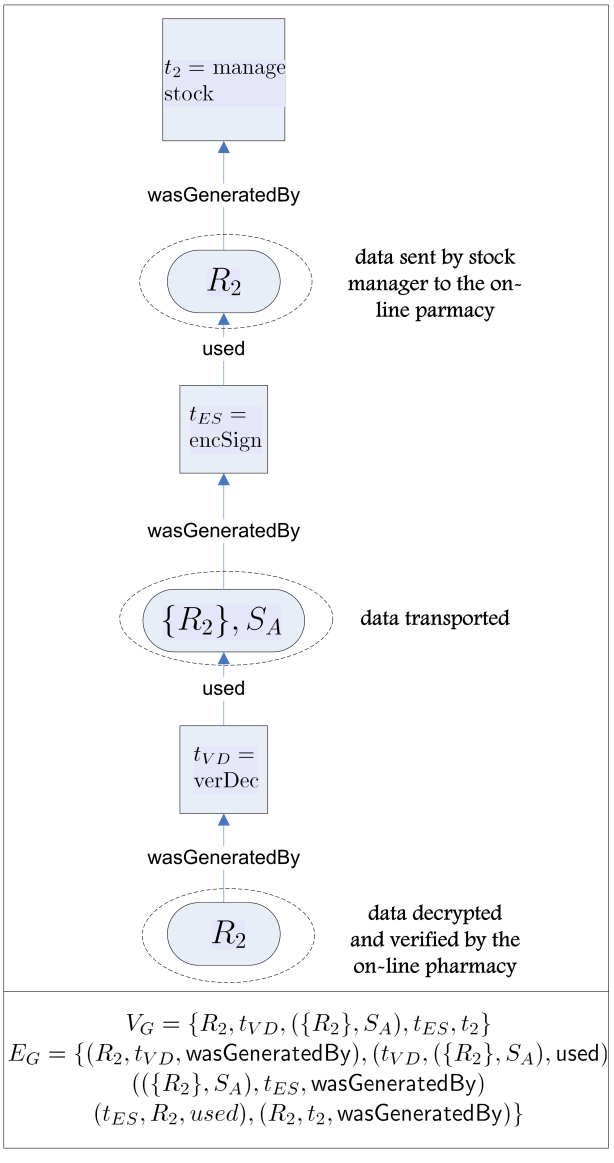


FIGURE 5.20: Basic Security Characteristics Verification Example: Result 2

5.4.5 Requirement H: Information Transferred to a Secure Country

The Information Transferred to a Secure Country states that personal information collected within the UK cannot be transferred to countries that do not implement the DPA principles explained in Chapter 3. In order to verify this requirement, it is necessary to include the task of transferring information in the Usage Rules Definition indicating to which countries personal information is transferred. Then, we make explicit the task "sentTo" and the country where this data is sent to. For example, if a set of data was sent to USA, the processing graph contains a task $t = \mathsf{sentToUSA}$. If the data is transferred within the European Union, the name of the task is $\mathsf{sentToEU}$. If the data is transferred within UK, the name of the task is simply sent. Note that the inclusion of this task does not change the structure of the Usage Rules Definition Graph, it just defines a specific task that should be notified and verified as the rest of the executed tasks.

To verify this requirement, we extract from the Processing View Graph the tasks related to the transferring of information to create the set T_{sent}. The elements of this set are compared with the ones contained in the Usage Rules Definition Graph. Since in the verification of this requirement just nodes are extracted, no subgraph is defined or depicted. The comparison process is explained in the next property.

Property 14 (Information Transferred to a Notified Country)**.** Let T_{sent} be a set containing all the tasks related to the transferring of information from G_W and $T_\Gamma \in G_\Gamma$ a set tasks. For each $x \in T_{sent}$ there exists one and only one $y \in T_\Gamma$ such that $x = y$.

If Property 14 holds in a subgraph T_{sent} of G_W, we can say that G_W is in compliance with the *Information Transferred to a Secure Country* Requirement. The verification process of this requirement is presented in Algorithm 7.

Algorithm 7 Data Transferred to a Secure Country

Input: $T_{sent} \in G_W, T_\Gamma \in G_\Gamma$
Output: 1 if the Data Transferred to a Secure Country is satisfied, -1 otherwise.

```
if ∀x ∈ T_sent : x ∈ T_Γ then
    return 1                          ▷ All the data was sent a secure country
else
    return -1                         ▷ Data sent to an insecure country
end if
```

The following example illustrates the verification of the Information Transferred to a Secure Country Requirement in the On-line Sales Scenario.

Example 5.7 (Information Transferred to a Secure Country Requirement Example)**.** *To illustrate how this requirement can be verified in the on-line sales scenario, we extract from the Processing View the set of all the tasks that involve the transfer of information. In this case this set of tasks is* $T_{sent} = \{sentToEU\}$*, which is presented in Figure 5.20.*

This graph shows that after the task **`manage stock`** *was executed, the result was sent to a country the EU. Since the task* sentToEU *is contained in the Usage Rules Definition, we can conclude that this task was in compliance with this requirement.*

Now, consider a different example, shown in Figure 5.22, where, instead of sending the sales report to the EU, it was sent to the United States. In this case, the destination country is not contained in the Usage Rules Definition, i.e. the task sentToUSA *is not contained in. Thus, we can conclude that this task was not in compliance with Requirement H.*

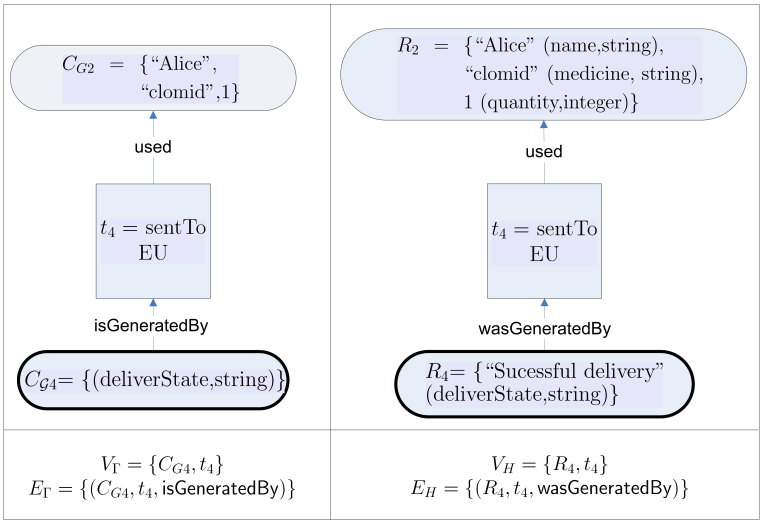


FIGURE 5.21: Information Transferred to a Secure Country Example 1

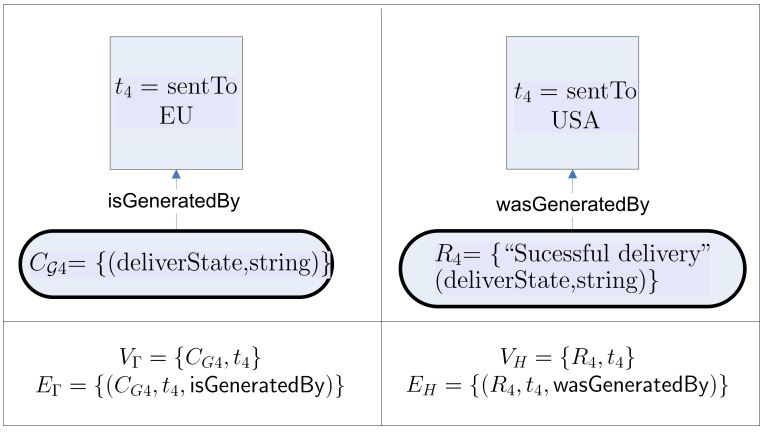


FIGURE 5.22: Information Transferred to a Secure Country Example 2

This concludes our discussion of a set of algorithms for verifying the Auditing Rules (B, C, H, G, F) using the Usage Definition Rules (specifying which operations can be performed on personal data), and the Processing View (which represents the operations that were actually performed, and the data instances on which they were performed). In combination, these algorithms constitute the third and final component of the Compliance Framework we developed in this chapter.

5.5 Discussion

Recently, a number of researchers have proposed provenance as a means of verifying compliance of different policies related to the use of personal information [66, 55, 119]. Others have proposed different technology to support compliance [20, 92, 81].

Hanson *et al.* [66] present a data-purpose algebra to annotate data with usage restrictions. With these restrictions, requirements similar to the ones presented here can be checked. In their approach, data is annotated after execution, in contrast to our approach, which creates provenance information at execution time. However, their approach does not contain any algorithms for performing the actual compliance verification.

Gil *et al.* [55] argue that computational workflow systems can be used to ensure and enforce the appropriate use of sensitive personal data. These systems provide compliance, transparency and accountability. To this end, they define a set of requirements that are similar to the ones we present here. Using a workflow system to support process transparency, the defined requirements can be verified. They also propose the use of this technology to enforce policies and negotiate these policies between the Data Subject and Data Controller. In our work, we argue that the openness of the Web makes enforcement of these policies infeasible. Instead, we propose compliance verification after the fact as a more viable solution. Moreover, Gil *et al.* do not offer a formalisation of their solution as we do here.

Ringelstein and Staab [119] present the PAPEL language (Provenance-aware Policy Definition and Execution Language) to define policies that control how application data and provenance information is accessed and processed. They use a modified version of OPM to express processing execution and an extended version of XACML to represent permission and restriction policies. By modelling execution conditions, proper use of data can be enforced. In contrast, as mentioned above, we provide accountability, not enforcement. Moreover, we do not modify OPM, but instead create a set of a graph-based rules, against which OPM-based provenance graphs can be easily checked without any transformations. They also create policies for controlling access and the processing of provenance information using XACML, while our policies (processing rules) apply to application data as well, and are represented as a DAG.

In the business processes context, Ly *et al.* [92] present similar work to verify the compliance of rules related to the quality of a data product (e.g. a database, a document, or a single record). They use process-aware information systems and process models to model the rules. Their requirements are different from ours, as they verify only the order in which processes are executed. Awad *et al.* [20] check compliance of control flow and data flow in business process models by using BPMN to express process models and BPMN-Q to represent policies. They design a set of queries and present how violations occur by comparing predefined patterns with the queries. These two approaches do not

support the verification of use of data items and are not based on an open model, such as OPM.

Kang *et al.* [81] present a similar approach to ours that helps users to conform with existing policies in a social networks context. This is achieved by making applications aware of data usage restrictions defined by the users. One drawback of this work is that they only define a small set of policies. In contrast, our framework defines a set of usage rules which can be easily extended and applied to a diversity of contexts.

Finally, there is also work on authorisation and enforcement of rules in databases [97, 141, 119]. However, given the openness of information on the Web and the possibility of inferring information using previously published information, authorisation and enforcement are very challenging to implement.

Miklau *et al.* [97] define accountability in databases as the ability to analyse past events to detect breaches, maintain data quality, and audit compliance with policies. In this way, their work is similar to ours. In order to make database systems accountable, databases should preserve a historical record of user operations such as creating, updating or selecting data records (accountability provenance). In contrast to our work, however, they store provenance with the data itself, which requires modifications to existing databases, and has possible negative repercussions to security. In terms of the provenance questions, their work only presents the idea of answering enquiries about which actions were taken when and by whom, but does not formally define these questions as we do.

Vaughan *et al.* [141] introduce the notion of evidence-based audits, which relies on a secure kernel that wraps critical primitives and logs their invocation with a so-called proof (which is similar to provenance), which acts as evidence that the operation was allowed. In contrast to our work, which records evidence of execution of application processes, instead of primitive operations. With respect to the rules that can be verified in their work, they use the proofs to provide information that enables a system to create access control policies. In our work, we do not consider access control. Moreover, our rules are created at design time depending on the necessities of each specific application.

5.6 Conclusions

In this chapter, we have presented the Compliance Framework for automatically verifying that past processing was carried out according to predefined processing rules. In this thesis, these rules are represented by the Auditing Requirements (see Chapter 3).

The first component of the Compliance Framework, the Usage Definition Rules (URD), represents these rules as a Directed Acyclic Graph (DAG). This DAG encodes restrictions on processing of personal data by specifying which classes and types of data can be used. The second component, the Processing View, represents provenance of data, i.e.

the actual operations that were performed on data. This Processing View (PV) is based on the Open Provenance Model (OPM), in which data and operations are represented as nodes, and edges represent the relations between them.

Using both the URD and the PV, we developed the third and final component of the Compliance Framework: a suite of algorithms for automatically verifying the Auditing Requirements. Since both the URD and PV are DAGs, these algorithms operate by matching subgraphs of the PV against the URD. Table 5.1 shows the extracted subgraphs and the requirements supported for each them. In summary, we have shown that by representing both past processing and processing rules as DAGs, the verification of the Auditing Requirements be easily performed and computationally implemented. Moreover, we believe that the URD can be used to not only represent the DPA requirements but also other processing rules that restrict the operations an application is allowed to perform. These rules can also be extended to verify proper use of any type of information as well.

Auditing Requirements	Supporting DAG
B – Purpose Compliance	$B_1(G_W, t_i)$ $B_2(G_W, t_i)$ $B_3(G_W, R_i)$
C – Relevant Information Verification	$C(G_W, D_{U_i})$
F – Anonymisation Preservation	$F(G_W, R_i)$
G – Basic Security Characteristics Verification	$G(G_W, D)$
H – Information Transferred to a Secure Country	$H(G_W, T)$

Table 5.1: Requirements supported by provenance DAGs

To ensure that the presented algorithms are practically viable, we developed a working proof of concept in the form of a library that provides audit functionality. This library allows for the manual definition of usage rules and can automatically analyse provenance to verify that data processing was performed in compliance with these rules.

The biggest practical challenge is that provenance and rules need to be expressed in a standard vocabulary when defining purposes, tasks, collected data and used data (classes and types). In that way, the auditing library can be used in an application independent manner. This issue is further discussed in Section 8.2.

Now, the Compliance Framework relies heavily on the correctness of the provenance DAGs (the Processing View Graphs). Thus, if these graphs are forged or altered, the result of the analysis can be compromised. Even though we secure the communication between the entities, the auditor is still able to alter provenance query results. Should this happen, the result of an audit processing will be based on incorrect provenance information. Consequently, it is necessary to secure provenance information, and ensure these types of attacks are effectively thwarted. Against this background, in the next chapter, we present a set of techniques for securing this provenance information.

Chapter 6

Securing the Provenance-based Auditing Architecture

At this point, we have modelled the complete provenance life cycle. In chapter 4 we defined the Provenance-based Auditing Architecture, an application architecture for recording, storing and querying provenance. Using the provenance that this architecture makes available, we also created the Compliance Framework, which contains a set of algorithms for analysing data processing and verifying that this processing was performed in accordance with the rules.

While the application data has been secured in Chapter 4, the provenance of this data that is captured by the Provenance-based Auditing Architecture and analysed by the Compliance Framework is still vulnerable. This is problematic, because if provenance is altered or forged by adversaries (including the auditor), audit results may be corrupted.

In more detail, these adversaries can perform attacks in an attempt to break one or more of the four basic security characteristics: confidentiality, integrity, authentication and non-repudiation [3]. Therefore, it is necessary to be able to detect any modification of provenance information during its complete life cycle of recording, storing, querying and analysis. In addition, if attacks were performed by insiders, it is also necessary to be able to identify this adversary.

To address this vulnerability, in this chapter, we secure the provenance generated by the Data Request, Task Request and Query Request protocols (see Chapter 4) by creating a secure communication channel between the actors within the architecture. These protocols guarantee that the provenance information within the system exhibits the four basic security characteristics.

Additionally, to secure the analysis stage developed as part of the Compliance Framework (see Chapter 5), we protect the provenance DAGs by applying cryptographic techniques. By doing so, we not only secure the graph as a whole, but also each part

independently. As a result, the properties of non-repudiation, end-point authentication and provenance information integrity are guaranteed to hold for the (subgraphs of) provenance DAGs. Moreover, we can guarantee that none of the actors, not even the auditor, can change p-assertions after these are recorded in the Provenance Store without being detected. We prove these guarantees using a model checker and verify that the mentioned security requirements hold during the complete provenance life cycle.

In summary, the contribution of this chapter is the *Secure Provenance-based Auditing Architecture*, a novel approach for securing provenance and application data during the entire provenance life cycle. This architecture contains a set of secure protocols for the exchange of provenance and a set of cryptographic components for securing provenance DAGs. Using this architecture, alteration of provenance information during its creation, storage and analysis can be detected, thereby improving the reliability of audits.

The remainder of this chapter is structured as follows. First, in Section 6.1, we introduce the terminology and concepts used throughout this chapter. Then, in Section 6.2, we secure communication protocols of the recording and storage phases of the provenance life cycle. Next, in Section 6.3, we secure the querying and analysis stages as well by developing the Secured Provenance Graph and an algorithm for checking its integrity. In Section 6.4, we combine these four stages to obtain the Secure Provenance-based Auditing Architecture, and formally prove that it exhibits the properties of confidentiality, integrity, authentication and non-repudiation. In Section 6.5 we discuss the contributions made in this chapter in the context of existing work. Finally, we conclude in Section 6.6.

6.1 Preliminaries

In this section, we define the notation that is used throughout the rest of the chapter. Specifically, we discuss how the assertions created by the actors of a provenance-aware system can be secured. To achieve this, we use a protocol for mutual authentication (a variant of TLS which we will refer to as the Optimised TLS Handshake Protocol [19]), and add cryptographic components (fields) to both the messages and the provenance assertions. These components are later used to verify the security properties during the storage stage, presented in Section 6.2.

Then, in Sections 6.2.1.2, 6.2.1.3 and 6.3.3, we secure the Data Request, Task Request and the Query Request communication protocols that were defined in Sections 4.3.2, 4.3.3 and 4.3.4. For this purpose, we first describe the Optimised TLS protocol and, later, introduce key concepts that are used to achieve this: key management, cryptographic functions and some useful auxiliary functions.

6.1.1 Optimised TLS Handshake Protocol

Recall from Section 2.3.1.3 that the OTLS protocol is a more efficient version of TLS which allows two actors to mutually authenticate each other in a single execution. Thus, OTLS ensures two important security properties required in our architecture: mutual authentication and confidentiality (see Section 4.1.3).

Specifically, as discussed in Section 2.3.1.3, OTLS is a protocol that enables client-server applications to negotiate a secure connection using a handshaking procedure, in which the participants agree on various parameters (such as cipher suites and session keys) to establish a secure connection. In doing so, entities are mutually authenticated through certificates and are able to create the session key that is used to encrypt the (sensitive) data that is contained in messages exchanged between the actors.

Now, in Section 6.2.1.1, we augment the OTLS protocol to capture provenance of its execution. This provenance is important to verify that the proper security measures were taken in the processing and transfer of sensitive application data (Requirement G in Section 3.4).

6.1.2 Key Management

In our architecture, cryptographic keys are used to sign and encrypt the messages that are exchanged between actors. In what follows, we assume that actors have already created private keys and exchanged public keys. We also assume that these keys do not change and are stored in a safe place so they cannot be compromised [94].

The keys related to each actor are presented in Tables 6.1 and 6.2. These tables contain all the actors participating in the Provenance-based Auditing Architecture (see Section 5). Hence, `DS`, `DC`, `DP`, `AU`, `PS` and `CA` are instances of the actors `Data Subject`, `Data Controller`, `Data Processor`, `Auditor`, `Provenance Store` and `Certified Authority`, respectively. Table 6.1 contains the private and public keys of each actor. Thus, given an actor A, k_A represents its public key and k_A^{-1} its private key. Table 6.2 presents the session keys created after executing the OTLS protocol between two communicating actors. Then, a session key k'_x is created, where x denotes the entities that use this specific key. For example, k'_{DS-DC} is used in the communication established between the entities DS and DC.

Note that the Certificate Authority is not part of the Architecture. However, we assume it exists and that it creates valid certificates for all actors inside the architecture.

	Entities					
Data	DS	DC	DP	AU	PS	CA
Public Key	k_{DS}	k_{DC}	k_{DP}	k_{AU}	k_{PS}	k_{CA}
Private Key	k_{DS}^{-1}	k_{DC}^{-1}	k_{DP}^{-1}	k_{AU}^{-1}	k_{PS}^{-1}	k_{CA}^{-1}

TABLE 6.1: Public, Private Keys used in Sequence Diagrams

Entities				
DS	DC	DP	AU	PS
k'_{DS-DC}	k'_{DS-DC}	k'_{DP-DC}	k'_{AU-PS}	k'_{DS-PS}
k'_{DS-PS}	k'_{DP-DC}	k'_{DP-PS}		k'_{DC-PS}
	k'_{DC-PS}			k'_{DP-PS}
				k'_{AU-PS}

TABLE 6.2: Session keys used in Sequence Diagrams

6.1.3 Cryptographic Notation

Before presenting the messages that are used in the sequence diagrams of this section, we first introduce the cryptographic notation we use on them.

In the communications protocols that are discussed in Sections 6.2.1.1, 6.2.1.2, 6.2.1.3 and 6.3.3, a hash-value h of a piece of data d is computed by a hash function $\mathtt{h}$, and represented as $h = \mathtt{h}(d)$. The concatenation operation[1] is represented by $\|$. The public and private keys of an actor A used for signing messages are represented by k_A and k_A^{-1} respectively. A signature of a message m is computed by $s = Sign_{k_A^{-1}}(m)$ and the extraction of m by $Ext_{k_A}(s) = m$. Note that, as explained in Section 2.3.2.2 the $Sign$ function only signs data, and does not hash data first. Thus, depending on whether it is used to sign data item m or the hash-value of m, the Ext function can be used to obtain m (in the clear) or the hash-value of m, respectively. The encryption of a piece of data d is computed by $e = \{d\}_{k'}$ and the decryption by $Dec_{k'}(e) = d$, where k' is a symmetric key. This symmetric key is a session key, which is agreed upon during the execution of a secure protocol and is used to encrypt private information.

6.1.4 Securing Messages

In this section, we secure the messages that are exchanged between the actors in the communication protocols. The first set of messages is used in the OTLS protocol [19, 79, 71], which provides mutual entity authentication and allows for the creation of a session key. The OTLS protocol, which was discussed in Section 6.1.1, will be formalised in Section 6.2.1.1. These messages are presented by a 5-tuple, a 4-tuple and a 3-tuple in (6.1), (6.2) and (6.3), respectively:

[1]Note that this concatenation operation is equivalent to the cons function used in programming languages as explained in Section 2.3.2.2

$$\mathsf{clientHello}(id_i, N_i, k_C, s_i, h_i) \tag{6.1}$$

$$\mathsf{serverHello}(id_i, e_i, s_i, h_i) \tag{6.2}$$

$$\mathsf{keyExchange}(id_i, e_i, h_i) \tag{6.3}$$

As can be seen in Table 6.3, and in messages (6.1), (6.2) and (6.3), the `client`'s key pair is represented by k_C and k_C^{-1} and its name is represented by C. Similarly, for the `server`, these are k_S, k_S^{-1} and S. Messages (6.1), (6.2) and (6.3) also contain a unique identifier id_i, which is used in the creation of provenance information. Also, message (6.3) contains the session key k'_{C-S} indicating this key is later used to encrypt data exchanged between the `client` (C) and the `server` (S).

Message Format	Accessors	Convenience Functions
Let Mi be clientHello$(id_i, N_i, k_C, s_i, h_i)$ where $s_i = Sign_{k_{CA}^{-1}}(C\|\|k_C)$, $h_i = \mathtt{h}(id_i\|\|C\|\|k_C)$	getID(Mi) = id_i, getRandomN(Mi) = N_i, getPubKey(Mi) = k_C, getSign(Mi) = s_i, getHash(Mi) = h_i	getName(Mi) = $head(Ext_{k_{CA}}$(getSign(Mi))), getDataItem(Mi) = getRandomN(Mi)||getName(Mi)||getPubKey(Mi)
Let Mi be serverHello(id_i, e_i, s_i, h_i) where $e_i = \{Sign_{k_S^{-1}}(k'\|\|N_j\|\|k_C)\}_{k_C}$, $s_i = Sign_{k_{CA}^{-1}}(S\|\|k_S)$, $h_i = \mathtt{h}(id_i\|\|S\|\|k_S)$, $N_j = N_i$, $k' = kgen(k_C)$.	getID(Mi) = id_i, getEncData(Mi) = e_i, getSign(Mi) = s_i, getHash(Mi) = h_i	getName(Mi) = $head(Ext_{k_{CA}}$(getSign(Mi))), getPubKey(Mi) = $snd(Ext_{k_{CA}}$(getSign(Mi))), getDataItem(Mi) = getRandomN(Mj)||getName(Mi)||getPubKey(Mi) where Mj is clientHello.
Let Mi be keyExchange(id_i, e_i, h_i) where $e_i = \{S\}_{k'_{C-S}}$, $h_i = \mathtt{h}(id_i\|\|C\|\|k_C)$.	getID(Mi) = id_i, getEncData(Mi) = e_i, getHash(Mi) = h_i	getDataItem(Mi) = getRandomN(Mj)||getName(Mj)||getPubKey(Mj) where Mj is serverHello.
Let Mi be secureMsg(id_i, e_i, s_i, h_i) where $e_i = \{id_i\|\|d\}_{k'}$, $s_i = Sign_{k_A^{-1}}(\mathtt{h}(id_i\|\|d\|\|h_i))$, $h_i = \mathtt{h}(id_i\|\|d)$. Where A is an actor who can be DS, DC or DP.	getID(Mi) = id_i, getEncData(Mi) = e_i, getSign(Mi) = s_i, getHash(Mi) = h_i	getAllData(Mi) = $Dec_{k'}$(getEncData(Mi)), getID(Mi) = $head$(getAllData(Mi)), getDataItem(Mi) = snd(getAllData(Mi))
Let Ii be ipa$(id_i, view, s_i)$ where $s_i = Sign_{k_A^{-1}}(\mathtt{h}(id_i\|\|view))$. Where view = $\{sender, receiver\}$ and A is an actor who can be DS, DC or DP.	getID(Ii) = id_i, getView(Ii) = $view$, getSign(Ii) = s_i	getAllData(Ii) = getID(Ii)||getView(Ii)
Let Ri be simpleRpa(e_i, s_i, h_i) where $e_i = \{id_i\|\|d\}_{k'}$, $s_i = Sign_{k_A^{-1}}(\mathtt{h}(id_i\|\|d\|\|h_i))$, $h_i = \mathtt{h}(id_i\|\|d)$. Where A is an actor who can be DS, DC or DP.	getEncData(Ri) = e_i, getSign(Ri) = s_i, getHash(Ri) = h_i	getAllData(Ri) = $Dec_{k'}$(getEncData(Ri)), getID(Ri) = $head$(getAllData(Ri)), getDataItem(Ri) = snd(getAllData(Ri))
Let Ri be rpa(e_i, s_i, h_i) where $e_i = \{id_i\|\|d\|\|rel\|\|id_j\}_{k'}$, $s_i = Sign_{k_A^{-1}}(\mathtt{h}(id_i\|\|d\|\|rel\|\|id_j\|\|h_i))$, $h_i = \mathtt{h}(id_i\|\|d\|\|rel\|\|h_j)$. Where h_j is the hash-value contained in message Mj identified by id_j and A is an actor who can be DS, DC or DP.	getEncData(Ri) = e_i, getSign(Ri) = s_i, getHash(Ri) = h_i	getAllData(Ri) = $Dec_{k'}$(getEncData(Ri)), getIDi(Ri) = $head$(getAllData(Ri)), getDataItem(Ri) = snd(getAllData(Ri)), getRel(Ri) = thd(getAllData(Ri)), getIDj(Ri) = fth(getAllData(Ri))

TABLE 6.3: Message Components and Convenience Functions

Note that in these messages, s_i represents a signature, e_i encrypted data and h_i a hash-value. Then, to access to the individual elements of the messages, we define a set of accessors and convenience functions in Table 6.3. It is important to note that the *Sign* operation used to construct one or more of the individual elements of these three messages ((6.1),(6.2) and (6.3)) does not hash data before signing it.

After the last message of the protocol is sent, the actors that executed the protocol have a symmetric key that can be used to protect the confidentiality of the exchanged

information. These actors have also created and exchanged their public keys, which are later used to provide non-repudiation and integrity.

In order to ensure all these security properties hold for sensitive application data, the *Sign* operation that is used to construct some of the individual elements in the rest of the messages is applied in conjuction with the hash operation (h), therefore, data is hashed before being signed.

Using these symmetric, public and private keys, we define secureMsg in (6.4). This message contains secured information that guarantees the confidentiality, non-repudiation and integrity properties on the original data that is exchanged in our protocols:

$$\mathsf{secureMsg}(e_i, s_i, h_i) \tag{6.4}$$

These messages are symmetrically encrypted and signed. Therefore, they contain encrypted data (e_i), a signature (s_i) and a hash-value (h_i) related to their corresponding assertions. The hash-value along with a unique identifier id_i, which is encrypted on e_i, (See Table 6.3) are used to create the relationship and the hash-value of the next assertions. We also define a set of accessors and convenience functions that are described in Table 6.3.

The ipa message in (6.5) contains the same elements of the message defined in (4.2) and a signature. This signature is used to guarantee the integrity and non-repudiation of interaction p-assertions related to the messages exchanged between the actors:

$$\mathsf{ipa}(id_i, view, s_i) \tag{6.5}$$

As previously defined in message (4.2), interaction p-assertions contain the unique identifier id_i of the message that the interaction refers to. They also contain a view indicating whether the entity that creates the interaction was the receiver or sender of the corresponding message. To provide non-repudiation and integrity of the information recorded in the Provenance Store, the interaction p-assertions' content is now signed using the sender's key (s_i). Moreover, a set of accessors and convenience functions related to message (6.5) is defined in Table 6.3. Note that the information contained in the ipa message is not encrypted. The reason for this is that this information ($id_i, view$) is not sensitive. Therefore, it cannot be used to break the security of the protocols.

Finally, messages (6.6) and (6.7) contain secure information that guarantees the relationship p-assertions that encode the relation between two interaction p-assertions is secured:

$$\mathsf{simpleRpa}(e_i, s_i, h_i) \tag{6.6}$$

$$\mathsf{rpa}(e_i, s_i, h_i) \tag{6.7}$$

Recall from message (4.3) that these p-assertions contain a message identifier id_i that is a unique identifier of the first interaction p-assertion (called the *cause*) and id_j of second (called the *effect*) of a relationships rel in the Provenance Store. To guarantee the four basic security characteristics, messages (6.6) and (6.7) make the following alterations to message 4.3:

- Integrity of the information asserted by the actors is guaranteed by introducing a hash-value h_i. This hash-value is created by the sender to protect the content of the assertion and its relationship with the previous message. For this reason, this hash-value includes the hash-value of the previous relation p-assertion, which is identified by its corresponding id.
- To provide non-repudiation, assertions also contain a signature s_i, which is computed by the sender.
- To provide confidentiality, the information transported in this message is encrypted in e_i.

Contrary to the information contained in the ipa messages, rpa messages contain sensitive information. Even though relationship p-assertions could contain pointers only, these pointers refer to existing pieces of data, and the fact that a relationship between these pieces of data exist is sensitive information. Knowledge of the name of the relationship and the pointers can be used to generate an (anonymous) graph from which information about the processing of the information might be inferred.

Finally, if an assertion related to the first message of a protocol is created, a special relationship p-assertion is created, which does not contain a relationship but a hash-value (see message (6.6)). As in previous secure messages, a set of accessors and convenience functions related to messages (6.6) and (6.7) is defined in Table 6.3.

Using the secure messages we created in this section, we can now secure the first two stages of the provenance life cycle: recording and storage.

6.2 Securing the Recording and Storage Stage

In this section, we secure the Data Request and Task Request communication protocols which are part of the Provenance-based Auditing Architecture presented in Chapter 4. Additionally, we modify the OTLS protocol to record provenance of its execution. By doing so, evidence is recorded of the fact that proper security measures have been taken, which is needed to verify compliance of the Security Requirements defined in Chapter 3.

In what follows, we formalise the modified protocols using an extension of UML called UMLSec, which offers cryptographic notation for secure systems development [79]. This formalisation includes one UML sequence diagram for each of the following protocols: OTLS, Data Request and Task Request. In the remainder of this section, we first show how these protocols are secured (Section 6.2.1). Then, in Section 6.2.2, we develop algorithms for verifying that the protocols were executed correctly by detecting message tampering or forgery.

6.2.1 Securing Protocols

6.2.1.1 OTLS Protocol

In this section, we modify the OTLS protocol discussed in 6.1.1 to capture and secure provenance of its execution. To do this, we present a UMLSec sequence diagram used in the formalisation of the OTLS protocol, which establishes a secure communication channel between the actors that are part of the Provenance-based Auditing Architecture.

The OTLS protocol allows actors to verify each others' identities and create a session key used to encrypt/decrypt exchanged messages, i.e. to authenticate themselves. For that reason, we use this protocol to establish communication between each of the entities (DS-DC, DP-DC) and between the entities and the Provenance Store[2] (DS-PS, DP-PS, DC-PS and AU-PS). This protocol is modelled in Figure 6.1, which shows the objects `client`, `server` and `PS`. Both `client` and `server` can be instances of any of the actors presented in the architecture, and `PS` is an instance of the `Provenance Store` actor. The model of this protocol is based on the examples presented in [79].

In order to achieve a clear distinction between protocol messages and provenance-related messages, we first explain the protocol messages that are part of the normal (i.e. "provenance unaware") execution of the protocol. Later, all the provenance-related messages (Interaction and Relationship p-assertions) are explained. Note that the same applies to the rest of the protocols (Data Request, Task Request and Query Request).

Messages In the OTLS sequence diagram, the messages exchanged between `client` and `server` are denoted as `Mi`. These messages are defined according to (6.1), (6.2) and (6.3), which represent the messages described in Section 6.1.1, and are explained in more detail below.

clientHello The `client` initiates the protocol by sending the clientHello message (`M1`).

This message contains a random number N_i and the `client`'s public key k_C. It

[2]Note that in order to protect the Provenance Store only p-assertions from a previously authenticated entity can be stored. Therefore, when the OTLS protocol is executed between an entity and the Provence Store no p-assertions are recorded.

also contains a message identifier id_{M1} and a hash-value, which are part of the provenance integrity support that is explained later. The `client` also includes its own certificate to authenticate to the `server`. In practice, this message also contains a lists of the cryptographic capabilities of the `client`, such as the version of the protocol, the cipher suites supported by the `client`, and the data compression methods supported by the `client`. In our model, these parameters are omitted as they do not alter the security of the protocol, which is the main goal of this formalisation. Below, message `M1` is formally described presenting its content during the execution of the protocol in Figure 6.1.

$$\begin{aligned} \text{Let certificate}_{M1} &= Sign_{k_{CA}^{-1}}(C||k_C) \\ \text{Let integrityHash}_{M1} &= \mathtt{h}(id_{M1}||C||k_C) \end{aligned}$$

`M1`=

clientHello(

id_{M1},	▷ `M1` OTLS message identifier
N_i,	▷ random number
k_C,	▷ `client`'s public key
certificate$_{M1}$,	▷ `client`'s certificate
integrityHash$_{M1}$	▷ provenance integrity hash-value

)

serverHello The serverHello message (`M2`) is sent by the `server` as a response to a clientHello message. Upon receipt of the clientHello message, the `server` generates the pre-master secret k'_{C-S} using a key generation function ($kgen$) [3] Then, the pre-master secret is signed along with the client random number[4] (N_j) and the client public key (k_C) using the server private key (k_S^{-1}), and encrypts it using the public key of the client (k_C). By signing the pre-master secret, the client random number and the client public key together, the server binds the pre-master secret with the current session and client. Then, the signed and encrypted pre-master secret is sent to the client along with its certificate ($Sign_{k_{CA}^{-1}}(S||k_S)$). This message is presented in (6.2). Again, this message also contains a message identifier id_{M2}, the cryptographic cipher suite and the data compression method selected by the `server`. As before, the last two parameters are omitted for the same reasons as mentioned above. Below, message `M2` content is formally described.

[3] $kgen$ denotes a key generation function used to create strong keys. To prevent related-key attacks [96], keys need to be generated truly randomly and contain sufficient entropy. To support these properties, keys can be randomly generated using a pseudo-random number generator (PRNG) [28] or a cryptographic hash function along with a pass-phrase [117].

[4] The random data in the clientHello and serverHello messages is created using a carefully designed pseudo-random function to be used in the creation of the key. In that way, it is possible to guarantee that the generated symmetric key will be unique in the scope of a single actor.

$$\begin{aligned} \text{Let signedEncryptedSecret}_{M2} &= \{Sign_{k_S^{-1}}(k'_{C-S}||N_j||k_C)\}_{k_C} \\ \text{Let certificate}_{M2} &= Sign_{k_{CA}^{-1}}(S||k_S) \\ \text{Let integrityHash}_{M2} &= \mathtt{h}(id_{M2}||S||k_S) \end{aligned}$$

M2=

serverHello(

id_{M2},	▷ M2 OTLS message identifier
signedEncryptedSecret$_{M2}$,	▷ signed and encrypted pre-master secret
certificate$_{M2}$,	▷server's certificate
integrityHash$_{M2}$	▷ provenance integrity hash-value

)

keyExchange When the client receives the serverhello message, it decrypts the pre-master secret with its private key and extracts the public key from the certificate. By extracting the signature on the pre-master secret as well as the server certificate, the client authenticates the server. Once the client extracts the pre-master secret and verifies the signatures, it sends the keyExchange message (M3). This message uses the created pre-master secret as symmetric encryption key[5] (k'_{C-S}) to encrypt the name of the server (S). This message is used to verify that both actors know the agreed upon key and is presented in (6.3). After that, the created symmetric key can be used to exchange application data in a secure way. Below, message M3 content is formally described.

$$\begin{aligned} \text{Let encryptedServerName}_{M3} &= \{S\}_{k'_{C-S}} \\ \text{Let integrityHash}_{M3} &= \mathtt{h}(id_{M3}||C||k_C) \end{aligned}$$

M3=

keyExchange(

id_{M3},	▷M3 OTLS message identifier
encryptedServerName$_{M3}$,	▷encrypted server's name
integrityHash$_{M3}$	▷provenance integrity hash-value

)

By exchanging these messages, the client and the server verify each others' identities and create a symmetric session key k'_{C-S}. When the protocol terminates, the exchange of encrypted application data can commence using the created session key . The role of client and server can be represented by any of the entities that communicate in our model. In this way, entities are mutually authenticated and there is a symmetric key used by two entities to encrypt the information they want to exchange.

[5] In practice, the pre-master secret is used to generate a master secret and then the session key used to encrypt data. As we assume that keys are created using well-designed methods (See Section 7.4), we do not present this process, which is performed by the server and the client using a public algorithm defined in the TLS standard. Instead, we use the pre-master secret as the session key.

In the sequence diagrams that follow, we assume that the OTLS protocol has been previously executed between the main entities of the protocol and between each entity and the Provenance Store. Thus, the corresponding entities have already been mutually authenticated and have the created symmetric key, as shown in Table 6.2.

Interaction p-assertions To make the OTLS protocol provenance-aware, actors should record the interaction p-assertions related to the messages they send, which are labelled `Ii` in Figure 6.1 and use the message format from (6.5).

Interaction p-assertions `I1`, `I4` and `I5` are recorded by the `client` in the `PS`. These assertions refer to the messages `M1` (clientHello), `M2` (serverHello) and `M3` (keyExchange) described above. Therefore, they contain the OTLS message identifiers of the corresponding messages, i.e. they contain id_{M1}, id_{M2} and id_{M3} respectively. Similarly, interaction p-assertions `I2`, `I3` and `I6` refer to the same messages, but now from the perspective of the `server`. Below, interaction p-assertions `I1, I2, I3, I4, I5` and `I6` are formally described.

Message `I1`

$$\text{Let signatureWithHash}_{I1} = Sign_{k_C^{-1}}(\mathtt{h}(id_{M1}||\text{sender}))$$

`I1`=
ipa(
id_{M1}, ▷ `M1` OTLS message identifier
sender, ▷view
signatureWithHash$_{I1}$ ▷message signature with hash
)

Message `I2`

$$\text{Let signatureWithHash}_{I2} = Sign_{k_S^{-1}}(\mathtt{h}(id_{M1}||\text{receiver}))$$

`I2`=
ipa(
id_{M1}, ▷`M1` OTLS message identifier
receiver, ▷view
signatureWithHash$_{I2}$ ▷message signature with hash
)

Message `I3`

$$\text{Let signatureWithHash}_{I3} = Sign_{k_S^{-1}}(\mathtt{h}(id_{M2}||\text{sender}))$$

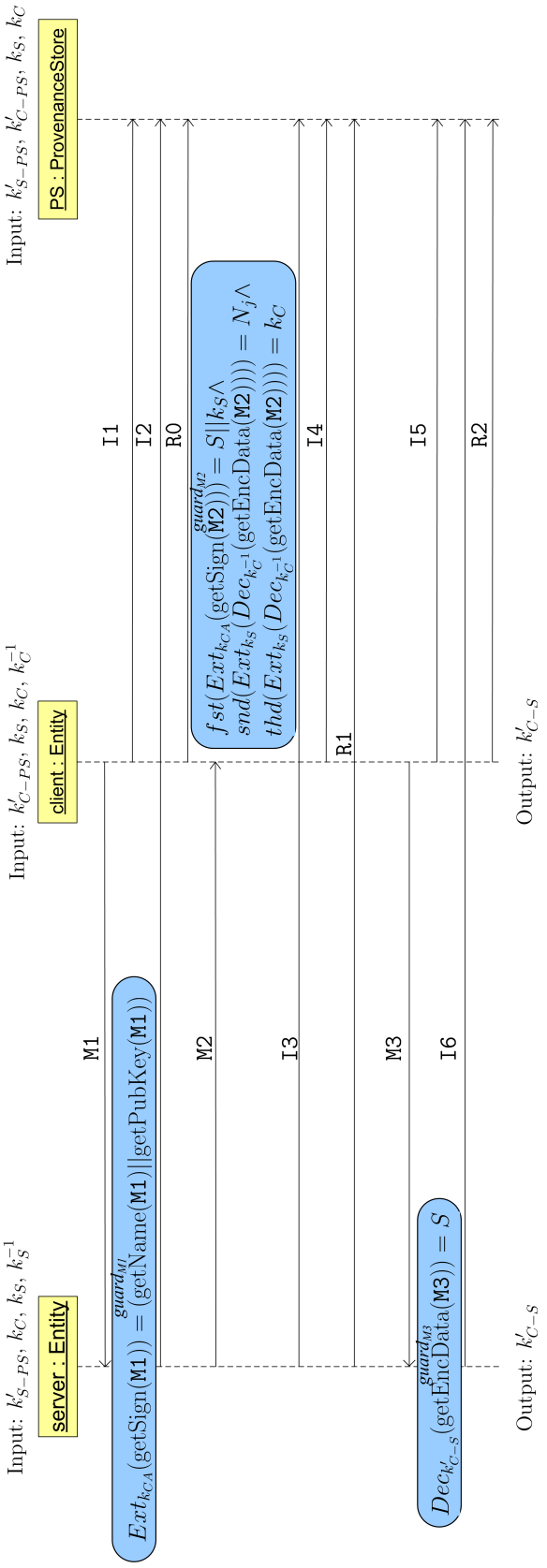


FIGURE 6.1: OTLS UMLSec Sequence Diagram

I3=
ipa(
id_{M2}, ▷M2 OTLS message identifier
sender, ▷view
signatureWithHash$_{I3}$ ▷message signature with hash
)

Message I4

Let signatureWithHash$_{I4} = Sign_{k_C^{-1}}(\mathtt{h}(id_{M2}||\text{receiver}))$

I4=
ipa(
id_{M2}, ▷M2 OTLS message identifier
receiver, ▷view
signatureWithHash$_{I4}$ ▷message signature with hash
)

Message I5

Let signatureWithHash$_{I5} = Sign_{k_C^{-1}}(\mathtt{h}(id_{M3}||\text{sender}))$

I5=
ipa(
id_{M3}, ▷M3 OTLS message identifier
sender, ▷view
signatureWithHash$_{I5}$ ▷message signature with hash
)

Message I6

Let signatureWithHash$_{I6} = Sign_{k_S^{-1}}(\mathtt{h}(id_{M3}||\text{receiver}))$

I6=
ipa(
id_{M3}, ▷M3 OTLS message identifier
receiver, ▷view
signatureWithHash$_{I6}$ ▷message signature with hash
)

Relationship p-assertions As part of a provenance-aware communication protocol, the actors also need to record the relationship p-assertions related to the interaction p-assertions they create. These assertions encode the relation between application data and the message that contains it. In Figure 6.1, these assertions are labelled Ri and use the message format from (6.6) and (6.7). Next, the relationship p-assertions created in this protocol are described.

- Relationship p-assertion R0 is not related to another message. Thus, since this message is the first message of the protocol, the ancestor hash of message M1 is created using (6.6). Below, relationship p-assertion R0 is formally described.

$$
\begin{aligned}
\text{Let encryptedData}_{R0} &= \{id_{M1}||\text{getDataItem(M1)}\}_{k_{C-PS}} \\
\text{Let integrityHash}_{R0} &= \text{h}(id_{M1}||\text{getDataItem(M1)}) \\
\text{Let signatureWithHash}_{R0} &= Sign_{k_C^{-1}}(\text{h}(id_{M1}||\text{getDataItem(M1)}||\text{integrityHash}_{R0}))
\end{aligned}
$$

R0=
simpleRpa(
encryptedData$_{R0}$, ▷encrypted data
signatureWithHash$_{R0}$, ▷message signature with hash
integrityHash$_{R0}$ ▷provenance integrity hash-value
)

- Message R1 is a relationship p-assertion denoting that the message M2 was sent *in Response To* the receipt of the message M1. Relationship R1 is created using (6.7) and below it is formally described.

$$
\begin{aligned}
\text{Let encryptedData}_{R1} &= \{id_{M1}||\text{getDataItem(M2)}||\text{inResponseTo}||id_{M2}\}_{k_{S-PS}} \\
\text{Let integrityHash}_{R1} &= \text{h}(id_{M1}||\text{getDataItem(M2)}||\text{inResponseTo}||\text{getHash(R0)}) \\
\text{Let signatureWithHash}_{R1} &= Sign_{k_S^{-1}}(\text{h}(id_{M1}||\text{getDataItem(M2)}||\text{inResponseTo}|| \\
&\quad id_{M2}||\text{integrityHash}_{R1}))
\end{aligned}
$$

R1=
rpa(
encryptedData$_{R1}$, ▷ encrypted data
signatureWithHash$_{R1}$, ▷ message signature with hash
integrityHash$_{R1}$ ▷ provenance integrity hash-value
)

- Finally, message R2 is a relationship indicating that the message M3 was sent *in Response To* the receipt of message M2. Below, relationship p-assertion R2 is formally described.

$$
\begin{aligned}
\text{Let encryptedData}_{R2} &= \{id_{M2}||\text{getDataItem(M3)}||\text{inResponseTo}||id_{M3}\}_{k_{C-PS}} \\
\text{Let integrityHash}_{R2} &= \text{h}(id_{M2}||\text{getDataItem(M3)}||\text{inResponseTo}||\text{getHash(R1)}) \\
\text{Let signatureWithHash}_{R2} &= Sign_{k_C^{-1}}(\text{h}(id_{M2}||\text{getDataItem(M3)}||\text{inResponseTo}|| \\
&\quad id_{M3}||\text{integrityhash}_{R2}))
\end{aligned}
$$

R2=
rpa(
encryptedData$_{R2}$, ▷ encrypted data
signatureWithHash$_{R2}$, ▷ message signature with hash
integrityHash$_{R2}$ ▷ provenance integrity hash-value
)

It is important to mention that, as opposed to other relationship p-assertions created in this formalisation, the ones related to the OTLS protocol just contain the digital certificates and public keys (instead of application data) of the corresponding messages. By doing so, we avoid the exposure of private information that could compromise the security of the protocol, for example by recording the symmetric key in the corresponding relationship p-assertion.

6.2.1.2 Securing Data Request Protocol

The second protocol we secure is the Data Request Protocol. Recall from Section 4.3.2 that this protocol is used to request personal information by the Data Controller from a Data Subject. To secure this protocol, we extend the sequence diagram presented in Figure 4.5. The result is shown in Figure 6.2. Note that the modified protocol uses the messages defined above. The difference with Figure 4.5 is that these messages are now secured (see Section 6.1.4). Furthermore, the modified protocol contains several *guards*, which are logical conditions that specify whether the execution of the protocol should be continued. These guards guarantee that the information and provenance exchanged between actors is secure. In Section 6.2.2, these guards are explained in further detail.

Messages In the Data Request sequence diagram, the messages exchanged between DS and DC are denoted as Mi. These messages are created according to (6.4) to provide confidentiality, non-repudiation and integrity. In what follows, we explain each message.

Message M4 contains purpose, which is symmetrically encrypted using the session key and signed using the private key of DC. With this message DC requests to DS personal information to be used with the indicated purpose. When M4 is received, it is verified and decrypted (shown by guard$_{M4}$). Below, message M4 is formally described.

$$\begin{aligned} \text{Let encryptedData}_{M4} &= \{id_{M4}||\text{purpose}\}_{k'_{DS-DC}} \\ \text{Let integrityHash}_{M4} &= \mathtt{h}(id_{M4}||\text{purpose}) \\ \text{Let signatureWithHash}_{M4} &= Sign_{k_{DC}^{-1}}(\mathtt{h}(id_{M4}||\text{purpose}||\text{integrityHash}_{M4})) \end{aligned}$$

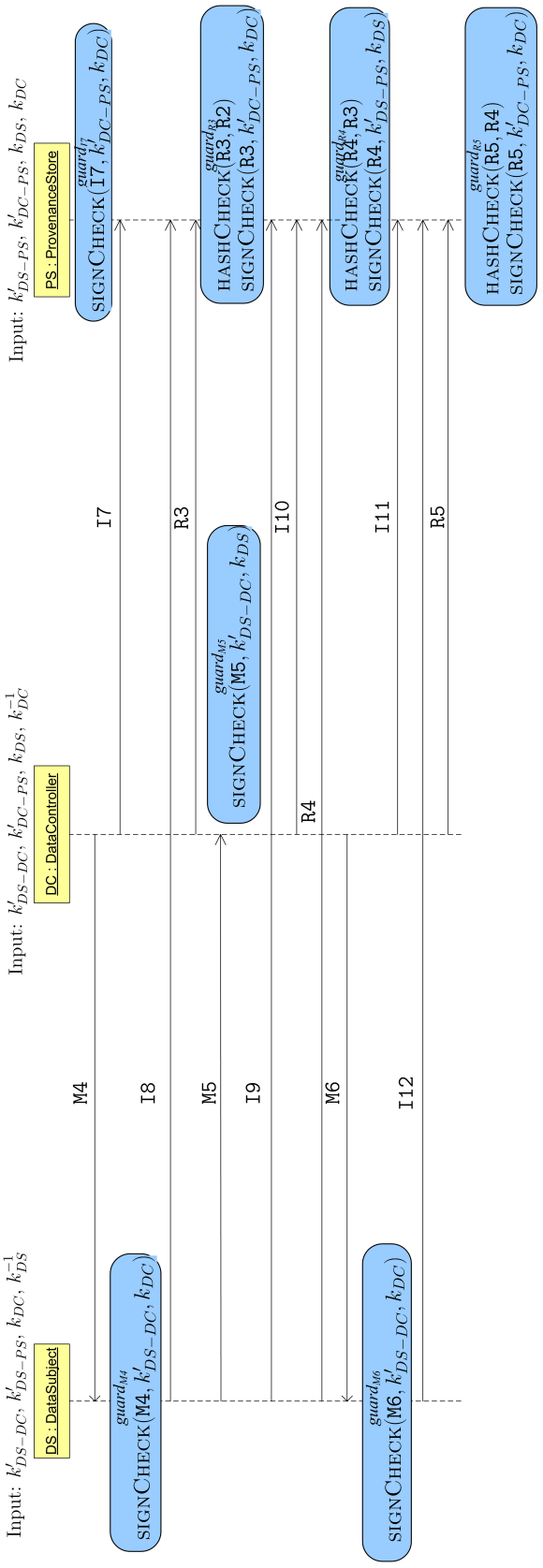


FIGURE 6.2: Data Request UMLSec Sequence Diagram Formalisation

M4=

secureMsg(

encryptedData$_{M4}$, ▷ encrypted data

signatureWithHash, ▷ message signature with hash

integrityHash$_{M4}$ ▷ provenance integrity hash-value

)

In response, DS sends the encrypted personal data requested (data) in Message M5, which is also signed by DS. When DC receives M5, the signature is verified and the data is decrypted, (see guard$_{M5}$), to later be stored in a local database, which is maintained by DC. Below, message M5 is formally described.

$$\begin{aligned} \text{Let encryptedData}_{M5} &= \{id_{M5}||\text{data}\}_{k'_{DS-DC}} \\ \text{Let integrityHash}_{M5} &= \mathtt{h}(id_{M5}||\text{data}) \\ \text{Let signatureWithHash}_{M5} &= Sign_{k_{DS}^{-1}}(\mathtt{h}(id_{M5}||\text{data}||\text{integrityHash}_{M5})) \end{aligned}$$

M5=

secureMsg(

encryptedData$_{M5}$, ▷ encrypted data

signatureWithHash$_{M5}$, ▷ message signature with hash

integrityHash$_{M5}$ ▷ provenance integrity hash-value

)

Then, DC acknowledges the receipt of the data to DS in M6, which is also verified and decrypted in guard$_{M6}$. Below, message M6 is formally described.

$$\begin{aligned} \text{Let encryptedData}_{M6} &= \{id_{M6}||\text{OK}\}_{k'_{DS-DC}} \\ \text{Let integrityHash}_{M6} &= \mathtt{h}(id_{M6}||\text{OK}) \\ \text{Let signatureWithHash}_{M6} &= Sign_{k_{DC}^{-1}}(\mathtt{h}(id_{M6}||\text{OK}||\text{integrityHash}_{M6})) \end{aligned}$$

M6=

secureMsg(

encryptedData$_{M6}$, ▷ encrypted data

signatureWithHash$_{M6}$, ▷ message signature with hash

integrityHash$_{M6}$ ▷ provenance integrity hash-value

)

Note that the message numbers continue from the OTLS sequence diagram showing that the OTLS protocol protocol was successfully completed before this protocol was executed. The same applies for the messages and protocols that follow.

Interaction p-assertions Interaction p-assertions (Ii) are built according to (6.5) providing non-repudiation and integrity. Similarly to the sequence diagram of Figure 4.5, in this sequence diagram assertions I7, I10 and I11 denote the act of receiving or sending messages M4, M5 and M6 by DC in the Provenance Store. Similarly, assertions I8, I9 and I12 denote the act of sending or receiving the same messages from the perspective of DS. The difference with Figure 4.5 is that in this sequence diagram the interaction

p-assertion are signed. Moreover, when interaction p-assertions are received by the PS, their signatures are checked at its reception. Below, interaction p-assertions `I7, I8, I9, I10, I11` and `I12` are formally described.

Message `I7`

$$\text{Let signatureWithHash}_{I7} = Sign_{k_{DC}^{-1}}(\mathtt{h}(id_{M4}||\text{sender}))$$

`I7`=
ipa(
id_{M4}, ▷`M4` Data Request message identifier
sender, ▷view
signatureWithHash$_{I7}$ ▷message signature with hash
)

Message `I8`

$$\text{Let signatureWithHash}_{I8} = Sign_{k_{DS}^{-1}}(\mathtt{h}(id_{M4}||\text{receiver}))$$

`I8`=
ipa(
id_{M4}, ▷`M4` Data Request message identifier
receiver, ▷view
signatureWithHash$_{I8}$ ▷message signature with hash
)

Message `I9`

$$\text{Let signatureWithHash}_{I9} = Sign_{k_{DS}^{-1}}(\mathtt{h}(id_{M5}||\text{sender}))$$

`I9`=
ipa(
id_{M5}, ▷`M5` Data Request message identifier
sender, ▷view
signatureWithHash$_{I9}$ ▷message signature with hash
)

Message `I10`

$$\text{Let signatureWithHash}_{I10} = Sign_{k_{DC}^{-1}}(\mathtt{h}(id_{M5}||\text{receiver}))$$

`I10`=
ipa(
id_{M5}, ▷`M5` Data Request message identifier
receiver, ▷view
signatureWithHash$_{I10}$ ▷message signature with hash
)

Message `I11`

$$\text{Let signatureWithHash}_{I11} = Sign_{k_{DC}^{-1}}(\mathtt{h}(id_{M6}||\text{sender}))$$

`I11`=
ipa(
id_{M6}, ▷`M6` Data Request message identifier
sender, ▷view
$\text{signatureWithHash}_{I11}$ ▷message signature with hash
)

Message `I12`

$$\text{Let signatureWithHash}_{I12} = Sign_{k_{DS}^{-1}}(\mathtt{h}(id_{M6}||\text{receiver}))$$

`I12`=
ipa(
id_{M6}, ▷`M6` Data Request message identifier
receiver, ▷view
$\text{signatureWithHash}_{I12}$ ▷message signature with hash
)

Relationship p-assertions Relationship p-assertions (`Ri`) are built according to (6.6) and (6.7) providing confidentiality, non-repudiation and integrity. The relationships created in this sequence diagram are the same as the ones created in the sequence diagram of Figure 4.5. The differences are the inclusion of the corresponding hash-values, the encryption of data application and the signing of messages. In this diagram, when a relationship p-assertion is received by the PS, its hash-value and signature are checked according to the corresponding guard. These guards are explained in detail in Section 6.2.2. Below, relationship p-assertions `R3, R4` and `R5` are formally described.

Message `R3`

$$\begin{aligned}
\text{Let encryptedData}_{R3} &= \{id_{M3}||\text{getDataItem}(\mathtt{M4})||\text{inResponseTo}||id_{M4}\}_{k'_{DC-PS}} \\
\text{Let integrityHash}_{R3} &= \mathtt{h}(id_{M3}||\text{getDataItem}(\mathtt{M4})||\text{inResponseTo}||\text{getHash}(\mathtt{R2})) \\
\text{Let signatureWithHash}_{R3} &= Sign_{k_{DC}^{-1}}(\mathtt{h}(id_{M3}||\text{getDataItem}(\mathtt{M4})||\text{inResponseTo}|| \\
&\quad id_{M4}||\text{integrityHash}_{R3}))
\end{aligned}$$

`R3`=
rpa(
encryptedData$_{R3}$, ▷encrypted data
signatureWithHash$_{R3}$, ▷message signature with hash
integrityHash$_{R3}$ ▷provenance integrity hash-value
)

Message `R4`

$$\begin{aligned}
\text{Let encryptedData}_{R4} &= \{id_{M4}||\text{getDataItem}(\texttt{M5})||\text{wasAcquiredFor}||id_{M5}\}_{k'_{DS-PS}} \\
\text{Let integrityHash}_{R4} &= \texttt{h}(id_{M4}||\text{getDataItem}(\texttt{M5})||\text{wasAcquiredFor}||\text{getHash}(\texttt{R3})) \\
\text{Let signatureWithHash}_{R4} &= Sign_{k_{DS}^{-1}}(\texttt{h}(id_{M4}||\text{getDataItem}(\texttt{M5})||\text{wasAcquiredFor}|| \\
&\quad id_{M5}||\text{integrityHash}_{R4}))
\end{aligned}$$

`R4`=

rpa(

encryptedData$_{R4}$, ▷encrypted data

signatureWithHash$_{R4}$, ▷message signature with hash

integrityHash$_{R4}$ ▷provenance integrity hash-value

)

Message `R5`

$$\begin{aligned}
\text{Let encryptedData}_{R5} &= \{id_{M5}||\text{getDataItem}(\texttt{M6})||\text{inAckTo}||id_{M6}\}_{k'_{DC-PS}} \\
\text{Let integrityHash}_{R5} &= \texttt{h}(id_{M5}||\text{getDataItem}(\texttt{M6})||\text{inResponseTo}||\text{getHash}(\texttt{R4})) \\
\text{Let signatureWithHash}_{R5} &= Sign_{k_{DC}^{-1}}(\texttt{h}(id_{M5}||\text{getDataItem}(\texttt{M6})||\text{inAckTo}|| \\
&\quad id_{M6}||integrityHash_{R5}))
\end{aligned}$$

`R5`=

rpa(

encryptedData$_{R5}$, ▷encrypted data

signatureWithHash$_{R5}$, ▷message signature with hash

integrityHash$_{R5}$ ▷provenance integrity hash-value

)

6.2.1.3 Securing Task Request Protocol

In this section, we secure the Task Request Protocol and record its provenance. Recall from Section 4.3.3 where this protocol is used to delegate the processing of personal data by `DS` to `DP`. This section presents the sequence diagram of the modified protocol, which is presented in Figure 6.3. This diagram models the interaction as shown in Figure 4.6, but instead uses secure message presented in Section 6.1.4.

Messages Similarly to the `Data Request` sequence diagram, messages are denoted as `Mi` and use (6.4) to provide confidentiality, non-repudiation and integrity.

Message `M10` contains `processdata`, which is symmetrically encrypted using the session key and signed using `DC`'s private key. This data is a set of personal information to be processed by `DP`. Thus, `DP` needs to verify the signature and decrypt such set of data

to process it, as $\texttt{guard}_{M10}$ shows. After the set of data is decrypted, `DP` computes it using the internal function `task1` that returns `result`. Below, message `M10` is formally described.

$$\begin{aligned}
\text{Let encryptedData}_{M10} &= \{id_{M10}||\text{processData}\}_{k'_{DP-DC}} \\
\text{Let integrityHash}_{M10} &= \mathtt{h}(id_{M10}||\text{processData}) \\
\text{Let signatureWithHash}_{M10} &= Sign_{k_{DC}^{-1}}(\mathtt{h}(id_{M10}||\text{processData}||\text{integrityHash}_{M10}))
\end{aligned}$$

`M10`=

secureMsg(
encryptedData$_{M10}$, ▷ encrypted data
signatureWithHash$_{M10}$ ▷ message signature with hash
integrityHash$_{M10}$ ▷ provenance integrity hash-value
)

Message `M11` is signed by `DP` and contains the encrypted result of the previous processing (`result`). Thus, when `DC` receives `M11`, the signature is verified and the result is decrypted in $\texttt{guard}_{M11}$. Below, message `M11` is formally described.

$$\begin{aligned}
\text{Let encryptedData}_{M11} &= \{id_{M11}||\text{result}\}_{k'_{DP-DC}} \\
\text{Let integrityHash}_{M11} &= \mathtt{h}(id_{M11}||\text{result}) \\
\text{Let signatureWithHash}_{M11} &= Sign_{k_{DP}^{-1}}(\mathtt{h}(id_{M11}||\text{result}||\text{integrityHash}_{M11}))
\end{aligned}$$

`M11`=

secureMsg(
encryptedData$_{M11}$, ▷ encrypted data
signatureWithHash$_{M11}$, ▷ message signature with hash
integrityHash$_{M11}$ ▷ provenance integrity hash-value
)

Finally, `DC` sends an acknowledgement to the reception of result to `DP` in `M12`, which is verified and decrypted in $\texttt{guard}_{M12}$. Below, message `M12` is formally described.

$$\begin{aligned}
\text{Let encryptedData}_{M12} &= \{id_{M12}||\text{OK}\}_{k'_{DP-DC}} \\
\text{Let integrityHash}_{M12} &= \mathtt{h}(id_{M12}||\text{OK}) \\
\text{Let signatureWithHash}_{M12} &= Sign_{k_{DC}^{-1}}(\mathtt{h}(id_{M12}||\text{OK}||\text{integrityHash}_{M12}))
\end{aligned}$$

`M12`=

secureMsg(
encryptedData$_{M12}$, ▷ encrypted data
signatureWithHash$_{M12}$, ▷ message signature with hash
integrityHash$_{M12}$ ▷ provenance integrity hash-value
)

Interaction p-assertions As in the previous protocol, interaction p-assertions (`Ii`) are built according to (6.5) providing non-repudiation and integrity. Similarly to the sequence diagram of Figure 4.6, in this sequence diagram interaction p-assertions `I13`,

`I14`, `I15`, `I16`, `I17` and `I18` denote the act of receiving and sending the messages described above by `DC` and `DP`. Again, interaction p-assertion are signed and their signatures are verified according to the corresponding guard. Below, interaction p-assertions `I13`, `I14`, `I15`, `I16` and `I17` are formally described.

Message `I13`

$$\text{Let signatureWithHash}_{I13} = Sign_{k_{DC}^{-1}}(\mathtt{h}(id_{M10}||\text{sender}))$$

`I13`=
ipa(
id_{M10}, ▷`M10` Task Request message identifier
sender, ▷view
signatureWithHash$_{I13}$ ▷message signature with hash
)

Message `I14`

$$\text{Let signatureWithHash}_{I14} = Sign_{k_{DP}^{-1}}(\mathtt{h}(id_{M10}||\text{receiver}))$$

`I14`=
ipa(
id_{M10}, ▷`M10` Task Request message identifier
receiver, ▷view
signatureWithHash$_{I14}$ ▷message signature with hash
)

Message `I15`

$$\text{Let signatureWithHash}_{I15} = Sign_{k_{DP}^{-1}}(\mathtt{h}(id_{M11}||\text{sender}))$$

`I15`=
ipa(
id_{M11}, ▷`M11` Task Request message identifier
sender, ▷view
signatureWithHash$_{I15}$ ▷message signature with hash
)

Message `I16`

$$\text{Let signatureWithHash}_{I16} = Sign_{k_{DC}^{-1}}(\mathtt{h}(id_{M11}||\text{receiver}))$$

`I16`=
ipa(
id_{M11}, ▷`M11` Task Request message identifier
receiver, ▷view
signatureWithHash$_{I16}$ ▷message signature with hash
)

Message `I17`

$$\text{Let signatureWithHash}_{I17} = Sign_{k_{DC}^{-1}}(\texttt{h}(id_{M12}||\text{sender}))$$

`I17`=
ipa(
id_{M12}, ▷`M12` Task Request message identifier
sender, ▷view
$\text{signatureWithHash}_{I17}$ ▷message signature with hash
)

Message `I18`

$$\text{Let signatureWithHash}_{I18} = Sign_{k_{DP}^{-1}}(\texttt{h}(id_{M12}||\text{receiver}))$$

`I18`=
ipa(
id_{M12}, ▷`M12` Task Request message identifier
receiver, ▷view
$\text{signatureWithHash}_{I18}$ ▷message signature with hash
)

Relationship p-assertions Here, relationship p-assertions (`Ri`) are also built according to (6.6) and (6.7) providing confidentiality, non-repudiation and integrity. The created relationships as the same as the ones created in the sequence diagram of Figure 4.6. However, this sequence diagram includes hash-values, encryption of data application and signing of messages. Then, when the relationships `R8`, `R9`, `R10`, `R11`, `R12` and `R13` are received by the PS, application data is decrypted, and their hash-values and signatures are verified, as $\texttt{guard}_{R8}$, $\texttt{guard}_{R9}$, $\texttt{guard}_{R10}$ $\texttt{guard}_{R11}$ $\texttt{guard}_{R12}$ and $\texttt{guard}_{R13}$ respectively show. Below, relationship p-assertions `R8, R9, R10, R11, R12` and `R13` are formally described.

Message `R8`

$$\begin{aligned}
\text{Let encryptedData}_{R8} &= \{id_{M10}||\text{getDataItem}(\texttt{M10})||\text{inResponseTo}||id_{M9}\}_{k'_{DC-PS}} \\
\text{Let integrityHash}_{R8} &= \texttt{h}(id_{M10}||\text{getDataItem}(\texttt{M10})||\text{inResponseTo}||\text{getHash}(\texttt{R7})) \\
\text{Let signatureWithHash}_{R8} &= Sign_{k_{DC}^{-1}}(\texttt{h}(id_{M10}||\text{getDataItem}(\texttt{M10})||\text{inResponseTo}|| \\
&\quad id_{M9}||\text{integrityHash}_{R8}))
\end{aligned}$$

`R8`=
rpa(
$\text{encryptedData}_{R8}$, ▷encrypted data
$\text{signatureWithHash}_{R8}$, ▷message signature with hash
$\text{integrityHash}_{R8}$, ▷provenance integrity hash-value
)

Message R9

$$
\begin{aligned}
\text{Let encryptedData}_{R9} &= \{id_{T1}||\text{getDataItem(T1)}||\text{wasInitiatedBy}||id_{M4}\}_{k'_{DP-PS}} \\
\text{Let integrityHash}_{R9} &= \text{h}(id_{T1}||\text{getDataItem(T1)}||\text{wasInitiatedBy}||\text{getHash(R8)}) \\
\text{Let signatureWithHash}_{R9} &= Sign_{k_{DP}^{-1}}(\text{h}(id_{T1}||\text{getDataItem(T1)}||\text{wasInitiatedBy}|| \\
&\quad id_{M4}||\text{integrityHash}_{R9}))
\end{aligned}
$$

R9=

rpa(
encryptedData$_{R9}$, ▷encrypted data
signatureWithHash$_{R9}$), ▷ message signature with hash
integrityHash$_{R9}$ ▷provenance integrity hash-value
)

Message R10

$$
\begin{aligned}
\text{Let encryptedData}_{R10} &= \{id_{M10}||\text{getDataItem(M10)}||\text{overlappedWith}||id_{M5}\}_{k'_{DC-PS}} \\
\text{Let integrityHash}_{R10} &= \text{h}(id_{M10}||\text{getDataItem(M10)}||\text{overlappedWith}||\text{getHash(R9)}) \\
\text{Let signatureWithHash}_{R10} &= Sign_{k_{DC}^{-1}}(\text{h}(id_{M10}||\text{getDataItem(M10)}||\text{overlappedWith}|| \\
&\quad id_{M5}||\text{integrityHash}_{R10}))
\end{aligned}
$$

R10=

rpa(
encryptedData$_{R10}$, ▷encrypted data
signatureWithHash$_{R10}$), ▷message signature with hash
integrityHash$_{R10}$ ▷provenance integrity hash-value
)

Message R11

$$
\begin{aligned}
\text{Let encryptedData}_{R11} &= \{id_{T1}||\text{getDataItem(T1)}||\text{used}||id_{M10}\}_{k'_{DP-PS}} \\
\text{Let integrityHash}_{R11} &= \text{h}(id_{T1}||\text{getDataItem(T1)}||\text{used}||\text{getHash(R10)}) \\
\text{Let signatureWithHash}_{R11} &= Sign_{k_{DP}^{-1}}(\text{h}(id_{T1}||\text{getDataItem(T1)}||\text{used}|| \\
&\quad id_{M10}||\text{integrityHash}_{R11}))
\end{aligned}
$$

R11=

rpa(
encryptedData$_{R11}$, ▷encrypted data
signatureWithHash$_{R11}$, ▷message signature with hash
integrityHash$_{R11}$ ▷provenance integrity hash-value
)

Message R12

$$
\begin{aligned}
\text{Let encryptedData}_{R12} &= \{id_{M11}||\text{getDataItem(M11)}||\text{wasGeneratedBy}||id_{T1}\}_{k'_{DP-PS}} \\
\text{Let integrityHash}_{R12} &= \mathtt{h}(id_{M11}||\text{getDataItem(M11)}||\text{wasGeneratedBy}||\text{getHash(R11)}) \\
\text{Let signatureWithHash}_{R12} &= Sign_{k^{-1}_{DP}}(\mathtt{h}(id_{M11}||\text{getDataItem(M11)}||\text{wasGeneratedBy}|| \\
&\quad id_{T1}||\text{integrityHash}_{R12}))
\end{aligned}
$$

R12=

rpa(

encryptedData$_{R12}$, ▷encrypted data

signatureWithHash$_{R12}$, ▷message signature with hash

integrityHash$_{R12}$ ▷provenance integrity hash-value

)

Message R13

$$
\begin{aligned}
\text{Let encryptedData}_{R13} &= \{id_{M12}||\text{getDataItem(M12)}||\text{inAckTo}||id_{M11}\}_{k'_{DC-PS}} \\
\text{Let integrityHash}_{R13} &= \mathtt{h}(id_{M12}||\text{getDataItem(M12)}||\text{inAckTo}||\text{getHash(R12)}) \\
\text{Let signatureWithHash}_{R13} &= Sign_{k^{-1}_{DC}}(\mathtt{h}(id_{M12}||\text{getDataItem(M12)}||\text{inAckTo}|| \\
&\quad id_{M11}||integrityHash_{R13}))
\end{aligned}
$$

R13=

rpa(

encryptedData$_{R13}$, ▷encrypted data

signatureWithHash$_{R13}$, ▷message signature with hash

integrityHash$_{R13}$ ▷provenance integrity hash-value

)

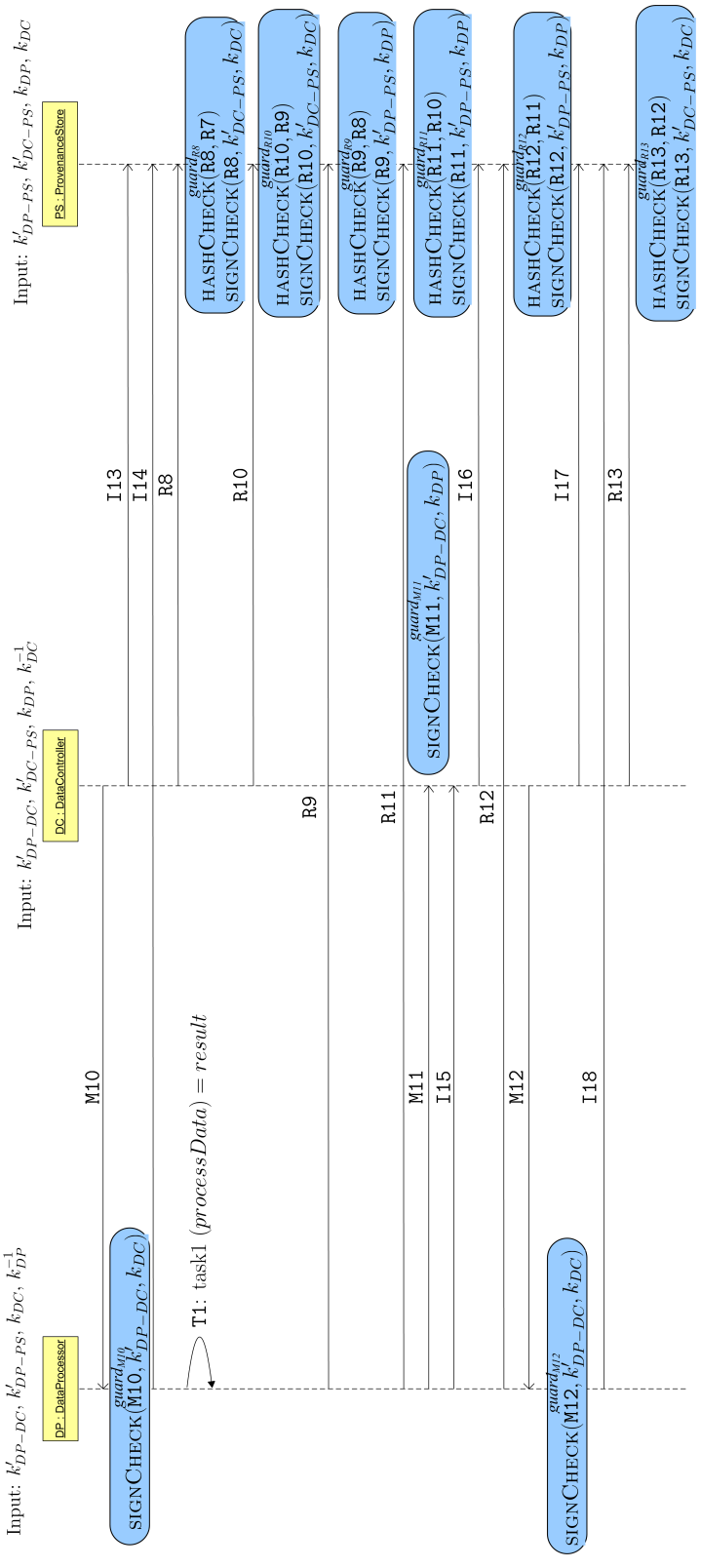


FIGURE 6.3: Task Request UMLSec Sequence Diagram Formalisation

6.2.2 Verifying the Execution of the Protocols

Now that the OTLS, Data Request and Task Request protocols have been secured, we can verify the integrity, confidentiality and non-repudiation of the messages and p-assertions exchanged between the actors. Thus, before assertions are recorded in the Provenance Store, they need to be checked. In the diagrams 6.1, 6.2 and 6.3, these checks are represented by guards, which are processes carried out by the actors after messages have been received.

Algorithm 8 Signature Message Checking

Input: `Mi` : *message*, k' : *sessionKey*, k_A : *publicKey*
Output: **true** if the signature of `Mi` is correct or **false** if it is not.

```
1: procedure SIGNCHECK(Mi : message, k' : sessionKey, k_A : publicKey)
2:     if Ext_{k_A}(getSign(Mi)) = h(Dec_{k'}(getEncData(Mi))) then
3:         return true                                  ▷ Signature is correct
4:     else
5:         return false                                 ▷ Signature is incorrect
6:     end if
7: end procedure
```

Guards are shown in the sequence diagrams in rounded blue rectangles and are identified by the names $\texttt{guard}_{Mi}$, $\texttt{guard}_{Ii}$ and $\texttt{guard}_{Ri}$, which are used to verify the content of messages `Mi`, interaction p-assertion `Ii` and relationship p-assertion `Ri`, respectively. These guards use Algorithm 8 to check the message signatures and Algorithm 9 to verify the hash-values that are included in the messages to guarantee their integrity. In the following sections, we discuss in further detail how the correct execution of the protocols described in Section 6.2 (OTLS, Data Request and Task Request) can be verified.

Algorithm 9 Hash-value Message Checking

Input: `Ri` : *message* and `Ri-1` : *message*
Output: **true** if the hash-value of `Ri` is correct or **false** if it is not.

```
1: procedure HASHCHECK(Ri : message, Ri-1 : message)
2:     if getHash(Ri) = h(getDataItem(Ri) || getRel(Ri) || getHash(Ri-1)) then
3:         return true                                  ▷ Hash-value is correct
4:     else
5:         return false                                 ▷ Hash-value is incorrect
6:     end if
7: end procedure
```

6.2.2.1 Verifying the Execution of OTLS

In this section, we only explain the guards related to the execution of the OTLS protocol; the explanation of the guards related to the assertions generated by these messages is very similar to the ones in the Data Request and Task Request protocols, and are therefore presented in the next section.

After receiving `M1`, $\texttt{guard}_{M1}$ is performed by the `server` to check the `client`'s certificate. In this guard, the signature is verified by extracting the `client`'s public key k'_C and the name C from the certificate (which is essentially the public key and the client name signed with the Certificate Authority private key) and comparing it to the plain copy of both. If this verification is successful, the `server` sends message `M2`. When the `client` receives `M2` the $\texttt{guard}_{M2}$ is verified. In this guard, the `server`'s certificate is verified in the same way as in the previous message. Then, the random number (N_j), the client's public key (k_C) and the pre-master secret (k') are extracted from the signature. The latter is used to create the next message. After that, message `M3` is sent with the `server`'s name encrypted with the session key. In $\texttt{guard}_{M3}$, this key is used to decrypt it and verify that both, the server and the client, posses the generated key. At this stage the session key is already agreed upon and it is renamed to k'_{C-S}.

6.2.2.2 Verifying the Execution of Data Request

The verification process for the Data Request protocol is divided in three parts: checking messages, checking interaction p-assertions and checking relationship p-assertions.

The guards related to the checking of messages `Mi` are very similar. When messages `M4`, `M5` and `M6` are received, the corresponding data is decrypted and the signature is verified using Algorithm 8. For example, $\texttt{guard}_{M4}$ represents the process that verifies the signature and decrypts the data contained in message `M4`, which in this case is the `purpose`. Thus, this guard first checks that the signature is valid and then decrypts `purpose`. The remaining guards ($\texttt{guard}_{M5}$, $\texttt{guard}_{M6}$) are interpreted similarly.

The second set of guards is related to the interaction p-assertions that are about to be stored in the Provenance Store. In this sequence diagram, we only present $\texttt{guard}_{I7}$, which represents the verification process of the interaction p-assertion `I7`. This p-assertion contains information about `M4` and it is recorded by `DC`. Then, this guard checks that the signature contained in `I7` is valid before recording its content in the Provenance Store using Algorithm 8. The guards related to the remaining interaction p-assertions are created and interpreted similarly.

Turning to relationship p-assertions, when `R3`, `R4`, `R5` are received by `PS`, their hash-values[6] and signatures are checked using Algorithm 8 and 9. These verifications are represented as guards $\texttt{guard}_{R3}$, $\texttt{guard}_{R4}$, $\texttt{guard}_{R5}$.

[6] As hash-values depend on hash-values contained in previous messages, the Provenance Store should wait until the indicated message arrive to be able to check the corresponding hash-value.

6.2.2.3 Verifying the Execution of Task Request

The interpretation of guards related to the checking of messages `Mi` in the `Task Request` protocol is very similar to the ones previously explained. Upon receipt of messages `M10`, `M11` and `M12`, their corresponding data is decrypted and the signature is verified using Algorithm 8.

The interpretation of interaction p-assertions' guards is the same as the ones explained in previous diagrams. As a consequence, these guards are not present in this protocol.

Turning to the guards that are related to the relationship p-assertions, the explanation of $\texttt{guard}_{R8}$, $\texttt{guard}_{R9}$, $\texttt{guard}_{R10}$, $\texttt{guard}_{R11}$, $\texttt{guard}_{R12}$ and $\texttt{guard}_{R13}$ is the same as the the previous guards related to relationship p-assertions: the signature and the hash-value are verified using Algorithm 8 and 9, respectively. After each guard is successfully checked, the corresponding assertion is stored in the Provenance Store. In that way, we guarantee that that the integrity of each assertion was not compromised.

6.2.2.4 Verifying Assertions during Storage Stage

After the successful execution of the protocols discussed above, the corresponding assertions are stored in the Provenance Store. It is important to note that assertions are decrypted and verified before being stored. Thus, in order to protect confidentiality during storage, assertions should be encrypted by the Provenance Store using a specially created key or can be protected by implementing access control techniques (See Section 2.4).

At any point it is possible to check the integrity of the entire content of the Provenance Store by verifying the hash-values and signatures of the assertions. This guarantees that the assertions were not modified during exchange or storage. This check can be performed periodically to verify the integrity of the Provenance Store, and to take the necessary measures if a problem is found. This mechanism can also be used to detect internal attacks, such as attacks from the Provenance Store administrator who attempts to maliciously modify the stored assertions. To be able to perform these checks, the secure storage of the entities' keys (public and private) is crucial. This needs to be done to avoid problems checking signatures created by entities whose public key have changed. In contrast, session keys do not need to be stored, since, as mentioned before, after decrypting the information it can be re-encrypted using a specific key for protecting confidentiality during storage.

Another important issue is the malicious insertion of assertions. This can occur in three different ways: (1) inserting messages in the communication channel between two actors, which leads to the creation of malicious assertions, (2) the insertion of a malicious assertions in the communication channel between an actor and the Provenance Store,

and (3) the direct insertion of malicious assertions into Provenance Store. The first attack is prevented by the OTLS protocol, which creates a secure communication channel. This prevents the insertion of malicious messages, and consequently, the creation of assertions related to them. To prevent the second attack, we assume that all actors that record assertions are properly authenticated to the Provenance Store, so we can trust the assertions created by them. To prevent the third one, we rely on the secure storage of the private keys of the actors, which are required to create properly signed assertions. Finally, as Assumption 1 explains (see Section 3.6), we assume that provenance information cannot be partially or totally deleted.

So far, we have secured individual assertions created by individual actors in our architecture. However, a provenance DAG returned in response to a query to the Provenance Store using only these secure assertions is still vulnerable. To illustrate this, consider the following scenario. Suppose that a piece of data d_1 is used by a process p_1 executed by actor A. After that, the same piece of data is used by a process p_2 executed by actor B. These operations lead to the creation of two (partially overlapping) fragments of the provenance graph: one containing d_1 and p_1, and a relation `R1` indicating that p_1 used d_1, and a fragment containing d_1 and p_2, and a relation `R2` indicating that p_2 used d_1. Now, when these fragments are merged into a single provenance DAG and returned as a query result, it has two vulnerabilities. The first is that actor A can change relation `R1`, because it possesses the keys used to sign this relation. For example, A can make it point to a different process or change its relationship name. The same applies to actor B and relation `R2`. The second vulnerability is that *any* actor can delete any node or relation of the DAG as well as add new ones. This is because nodes do not secure information about the relationships they participate in. For example, any actor can delete p_1 and `R1`, because d_1 does not have information about the fact that it participates in `R1`. Consequently, we need a different mechanism to protect the integrity of provenance graphs. This mechanism is presented in the next section.

6.3 Securing the Querying and Analysis Stage

At this point, we can guarantee that the assertions generated by actors and stored in the Provenance Store have not been maliciously altered during the recording and storage stages. Thus, when the processing of information has finished, assertions can be queried to obtain provenance DAGs containing the provenance of a particular piece of data. To maintain the integrity and non-repudiation of these provenance DAGs during the querying stage, the Provenance Store inserts new cryptographic components. Then, after a provenance DAG has been created, it can be securely transported. To achieve this, we develop the *Secured Provenance Graph*, a data structure that is included in each node of a provenance DAG during its creation and is later used to verify its integrity. By including this structure, we can protect the provenance DAGs from any malicious

alteration performed by an attacker (who may even be an auditor). At the same time, the identity of the actor who created a provenance DAG becomes verifiable.

First, in Section 6.3.1, we present the Secured Provenance Graph. Then, in Section 6.3.2 we present the algorithm to verify the integrity of a Secured Provenance Graph. Finally, we secure the Query Protocol defined in Section 6.3.3, and show where in the protocol this graph is created.

6.3.1 Secured Provenance Graph

Let us consider a set of node identifiers Id, a set of data D, a set of hash-values H, and a set of relationships' names R. A *Secured Provenance Graph* $G = (V, E, Node, Edge)$ is a directed acyclic graph, where $V = Id$, $E \subseteq Id \times Id$, $Node : Id \rightarrow D \times H$ is the node labelling function, and edges are labelled using the function $Edge : E \rightarrow R$.

Furthermore, given a Secured Provenance Graph G, we include a reference to a piece of data and a hash-value into each node. Then, given a node $id \in V$, we obtain its corresponding data by the accessor $data_G(id)$ and its corresponding hash-value by the accessor $hash_G(id)$. We also create a lexicographically ordered list (the reason for doing this will be discussed later) of ancestors' identifiers by the accessor $ancestor_G(id)$. The hash-value contained in each node is then computed as:

$$compHash_G(id) = \mathtt{h}\left(data_G(id) \Big|\Big| \underset{id_i \in ancestor_G(id)}{\Big|\Big|} edge(id, id_i) \parallel hash_G(id_i)\right) \quad (6.8)$$

Equation (6.8) creates a hash-value that is used to verify not only the integrity of the data and the relationships related to id but also the integrity of the past (origin) of the data. This is achieved by including the hash-values of the ancestors of id, which creates an unforgeable reference to its past. The complete Secured Provenance Graph is protected by the signature of the Provenance Store, so it is not possible for another entity to reproduce it without being noticed. We compute this signature $S = Sign_{k_{PS}^{-1}}(G)$, which is attached to the corresponding provenance graph after the provenance graph is constructed. We also define the accessor $sign_G(G)$ to obtain the signature related to the graph G.

Figure 6.4 presents an example of a Secured Provenance Graph, in which nodes are represented by circles containing data d_i, the directed edges are labelled with relationships r_i and the hash-values associated with each node are represented as h_i.

We assume that a simple provenance graph is created by a querying service. After that, Equation (6.8) is used in each of the nodes of the simple provenance graph to create a Secured Provenance Graph.

Note that instead of data, nodes can contain references to data. If that is the case, the necessary credentials to access to the original piece of data and a hash-value of such a piece of data should be included, in addition to the reference itself. In that way, after the original data is retrieved using the corresponding credentials, its integrity can be checked using the included hash-value.

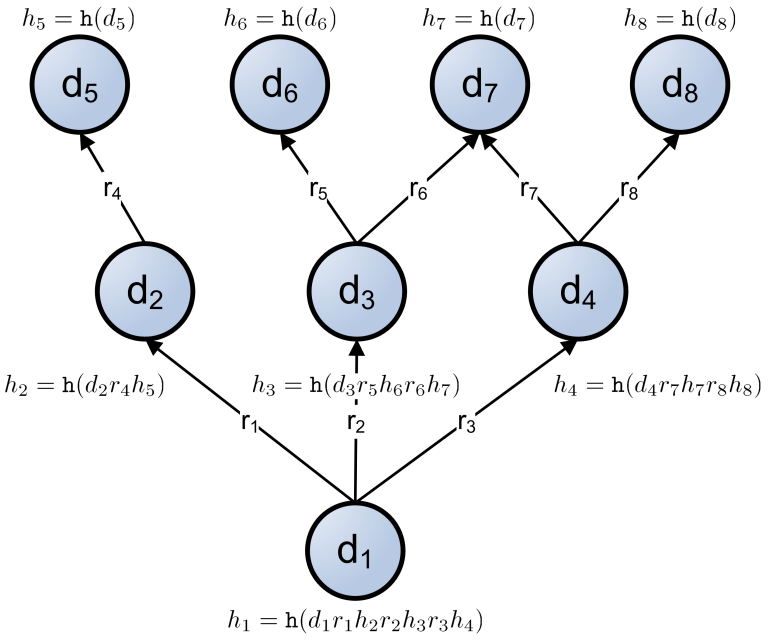


FIGURE 6.4: An example of a Secured Provenance Graph

It is important to note that the order of the relationships and the hash-values in each node is a very important issue. In graphs, the outgoing edges of a node are not ordered. However, if we want to create and later verify the hash-values contained in such a node, it is necessary to preserve certain order in the checking process. For example, the hash-value of node d_3, which is presented in Figure 6.4, can be created in two different ways. If we take r_5 in first place, we obtain the hash-value $h_3 = \mathtt{h}(d_3 r_5 h_6 r_6 h_7)$. But, if we take r_6 in first place, its hash-value is $h_3 = \mathtt{h}(d_3 r_6 h_7 r_5 h_6)$. Both hash-values represent the same node in the provenance graph. Nevertheless, if we do not know the order in which the hash-value was created, its checking will be incorrect as $\mathtt{h}(d_3 r_5 h_6 r_6 h_7)$ is different from $\mathtt{h}(d_3 r_6 h_7 r_5 h_6)$. In our case, the list of ancestors' identifiers is lexicographically ordered according to the relationship's names. Then, the correct hash-value is $h_3 = \mathtt{h}(d_3 r_5 h_6 r_6 h_7)$. Therefore, choosing and defining an order technique that all the principals know beforehand and maintain during the creation, recording and querying processes is decisive in the integrity checking of provenance graphs. If such an ordering technique is not used in all the mentioned processes, we are not able to check the integrity of the provenance information, even if the integrity is intact.

Note that a provenance graph can contain nodes with no relationships. This does not mean that such nodes do not have a "past". Instead, however, it means that the provenance graph does not contain the past of such nodes because it is not relevant for the analysis stage[7]. If for some reason, a problem is found in these nodes without explicit past, the auditor can request a provenance graph showing their past from the Provenance Store. Later, this new provenance graph can be checked.

6.3.2 Verifying a Secured Provenance Graph

During the querying stage, an auditor sends a provenance query to the Provenance Store which returns a provenance graph in response. If this provenance graph is a Secured Provenance Graph, its integrity should be checked before commencing a compliance analysis (see Chapter 5). By doing so, any tampering or forgery of the elements of the provenance graph can be detected, and the perpetrators can be identified. If the graph is valid, the compliance analysis is allowed to begin.

Algorithm 10 The Integrity Checking Algorithm

Input: $G = (V, E, Node, Edge)$ a Secured Provenance Graph and k_{PS} the Public Key of PS.

Output: 1 if the integrity of G is not compromised or 0, -1 otherwise.

```
1:  procedure INTEGRITYCHECK(G : Secured Provenance Graph, k_PS : publicKey)
2:      id : node identifier ∈ V
3:      if Ext_{k_PS}(sign_G(G)) ≠ h(G) then
4:          return 0                                  ▷ signature is incorrect
5:      end if
6:      for each id ∈ V do
7:          if hash_G(id) ≠ (compHash_G(id)) then
8:              return -1                             ▷ integrity is compromised in id
9:          end if
10:     end for
11:     return 1                                      ▷ Success
12: end procedure
```

In order to perform an integrity check of a Secured Provenance Graph, we introduce the INTEGRITYCHECK algorithm (see Algorithm 10). This algorithm first verifies the signature associated with the graph using the public key of the Provenance Store k_{PS}. This signature is used to check whether the content of any part of the complete graph was altered. If this signature cannot be verified, there is no reason to continue with the rest of the process, and the algorithm returns 0.

If the signature is found to be correct, the algorithm proceeds to verify the hash-value of each node in the graph. This is achieved by visiting each node in the graph, and computing a hash-value by calling the $compHash_G$ function (see Equation 6.8) for each

[7]The relevance of the information is defined by the provenance queries, such as the ones presented in Chapter 4

of these nodes. This new hash-value depends on the ancestors' hashes, which in turn depend on the hash-values of their ancestors, etcetera. This computed hash-value is compared against the hash-value contained in the node. If they are found to be different, the integrity of this node has been compromised. If, after visiting all the nodes, no problems are found, we can conclude that the integrity of all nodes is intact.

Now, if the integrity of any of the provenance graph nodes was found to be compromised, the algorithm indicates which one was altered by outputting its corresponding *id*. The auditor can access the information stored in the Provenance Store related this *id* in order to check whether it was altered since the recording stage (see Section 6.2.2.4). If, however, the integrity of the provenance DAG is intact, the audit process is allowed to begin. In this way, we can guarantee that the results derived from the analysis of a secured provenance graph are based on information that was not maliciously altered.

In this thesis, we assume that deletion of provenance information is not allowed since all assertions that our model records to be able to perform a successful audit are required (see Assumption 1 in Section 3.6). Moreover, if one or more assertions are deleted, Algorithm 10 will find an integrity problem, because the hash-values are incorrect. However, it is impossible to determine whether this is caused by the deletion of assertions, or by the fact that some assertions have been (maliciously) altered. To avoid this from happening, provenance repositories should implement appropriate access control techniques.

6.3.3 Securing the Query Request Protocol

To have a completely secured querying stage, the communication between auditors and Provenance Store also needs to be secured to avoid that a provenance graph is forged during this transmission. Thus, in this section we secure the `Query Request` protocol (see Section 4.3.4).

The process that models the Query Request protocol (Figure 6.5) is similar to the one explained in Section 4.3.4, but the former uses the secure messages defined in Section 6.1.4. The `Auditor` initiates the protocol by requesting the provenance of a certain data item (*item*) under certain scope (*scope*) in message `Q1`. Below, message `Q1` is formally described.

$$
\begin{aligned}
\text{Let encryptedData}_{Q1} &= \{id_{Q1}||\text{item}||\text{scope}\}_{k'_{AU-PS}} \\
\text{Let integrityHash}_{Q1} &= \mathtt{h}(id_{Q1}||\text{item}||\text{scope}) \\
\text{Let signatureWithHash}_{Q1} &= Sign_{k_{AU}^{-1}}(\mathtt{h}(id_{Q1}||\text{item}||\text{scope}||\text{integrityHash}_{Q1}))
\end{aligned}
$$

Q1=
secureMsg(
encryptedData$_{Q1}$, ▷ encrypted data
signatureWithHash$_{Q1}$, ▷ message signature with hash
integrityHash$_{Q1}$ ▷ provenance integrity hash-value
)

The PS resolves the query and returns the result *qresult* to Auditor in message Q2. The difference from the unsecured protocol from Section 4.3.4 is that here, the query result is represented as a Secured Provenance Graph which is created by the Provenance Store. Then, after the provenance query result is obtained, Equation (6.8) is used in each of the nodes of such result to create a Secured Provenance Graph and sent to Auditor. Below, message Q2 is formally described.

$$\begin{aligned} \text{Let encryptedData}_{Q2} &= \{id_{Q2}||\text{qresult}\}_{k'_{AU-PS}} \\ \text{Let integrityHash}_{Q2} &= \mathtt{h}(id_{Q2}||\text{qresult}) \\ \text{Let signatureWithHash}_{Q2} &= Sign_{k^{-1}_{PS}}(\mathtt{h}(id_{Q2}||\text{qresult}||\text{integrityHash}_{Q2})) \end{aligned}$$

Q2=
secureMsg(
encryptedData$_{Q2}$, ▷ encrypted data
signatureWithHash$_{Q2}$, ▷ message signature with hash
integrityHash$_{Q2}$ ▷ provenance integrity hash-value
)

When the Secured Provenance Graph is received by Auditor, its integrity is checked by calling the INTEGRITYCHECK procedure in Algorithm 10. In this model, this procedure is represented as part of the guard that is executed after the provenance query result is received. If the integrity of the provenance graph has not been compromised, an acknowledgement is sent to PS in message Q3 and the compliance verification process is allowed to begin. If not, the protocol terminates and indicates to the PS which node is incorrect. Below, message Q3 is formally described.

$$\begin{aligned} \text{Let encryptedData}_{Q3} &= \{id_{Q3}||\text{OK}\}_{k'_{AU-PS}} \\ \text{Let integrityHash}_{Q3} &= \mathtt{h}(\{id_{Q3}||\text{OK}) \\ \text{Let signatureWithHash}_{Q3} &= Sign_{k^{-1}_{AU}}(\mathtt{h}(\{id_{Q3}||\text{OK}||\text{integrityHash}_{Q3})) \end{aligned}$$

Q3=
secureMsg(
encryptedData$_{Q3}$, ▷ encrypted data
signatureWithHash$_{Q3}$, ▷ message signature with hash
integrityHash$_{Q3}$ ▷ provenance integrity hash-value
)

As in previous diagrams, we formally describe the interaction p-assertions ($\mathtt{I}_{Q1}$, $\mathtt{I}_{Q2}$ and $\mathtt{I}_{Q3}$ and relationship p-assertions ($\mathtt{R}_{Q1}$, $\mathtt{R}_{Q2}$ and $\mathtt{R}_{Q3}$) created during the execution of the Query Request Protocol.

Message $\mathtt{I}_{Q1}$

$$\text{Let signatureWithHash}_{I_{Q1}} = Sign_{k_{AU}^{-1}}(\mathtt{h}(id_{Q1}||\text{sender}))$$

$\mathtt{I}_{Q1}$=
ipa(
id_{Q1}, ▷ `Q1` Query Request message identifier
sender, ▷view
$\text{signatureWithHash}_{I_{Q1}}$ ▷message signature with hash
)

Message $\mathtt{I}_{Q2}$

$$\text{Let signatureWithHash}_{I_{Q2}} = Sign_{k_{AU}^{-1}}(\mathtt{h}(id_{Q2}||\text{receiver}))$$

$\mathtt{I}_{Q2}$=
ipa(
id_{Q2}, ▷`Q2` Query Request message identifier
receiver, ▷view
$\text{signatureWithHash}_{I_{Q2}}$ ▷message signature with hash
)

Message $\mathtt{I}_{Q3}$

$$\text{Let signatureWithHash}_{I_{Q3}} = Sign_{k_{AU}^{-1}}(\mathtt{h}(id_{Q3}||\text{sender}))$$

$\mathtt{I}_{Q3}$=
ipa(
id_{Q3}, ▷`Q3` Query Request message identifier
sender, ▷view
$\text{signatureWithHash}_{I_{Q3}}$ ▷message signature with hash
)

Message $\mathtt{R}_{Q1}$

$$\begin{aligned}
\text{Let encryptedData}_{R_{Q1}} &= \{id_{Q1}||\text{getDataItem}(\mathtt{Q1})||\text{inResponseTo}||id_{M3}\}_{k'_{AU-PS}} \\
\text{Let integrityHash}_{R_{Q1}} &= \mathtt{h}(id_{Q1}||\text{getDataItem}(\mathtt{Q1})||\text{inResponseTo}||\text{getHash}(\mathtt{R2})) \\
\text{Let signatureWithHash}_{R_{Q1}} &= Sign_{k_{AU}^{-1}}(\mathtt{h}(id_{Q1}||\text{getDataItem}(\mathtt{Q1})||\text{inResponseTo}|| \\
&\quad id_{M3}||\text{integrityHash}_{R_{Q1}}))
\end{aligned}$$

$\mathtt{R}_{Q1}$=
rpa(
$\text{encryptedData}_{R_{Q1}}$, ▷encrypted data
$\text{signatureWithHash}_{R_{Q1}}$, ▷message signature with hash
$\text{integrityHash}_{R_{Q1}}$ ▷provenance integrity hash-value
)

Message $\mathtt{R}_{Q2}$

$$
\begin{aligned}
\text{Let encryptedData}_{R_{Q2}} &= \{id_{Q2}||\text{getDataItem}(\mathtt{Q2})||\text{queryResultOf}||id_{Q1}\}_{k'_{AU-PS}} \\
\text{Let integrityHash}_{R_{Q2}} &= \mathtt{h}(id_{Q2}||\text{getDataItem}(\mathtt{Q2})||\text{queryResultOf}||\text{getHash}(\mathtt{Q1})) \\
\text{Let signatureWithHash}_{R_{Q2}} &= Sign_{k_{AU}^{-1}}(\mathtt{h}(id_{Q2}||\text{getDataItem}(\mathtt{Q2})||\text{queryResultOf}|| \\
&\quad id_{Q1}|\text{integrityHash}_{R_{Q2}}))
\end{aligned}
$$

$\mathtt{R}_{Q2}$=

rpa(

encryptedData$_{R_{Q2}}$, ▷encrypted data

signatureWithHash$_{R_{Q2}}$, ▷message signature with hash

integrityHash$_{R_{Q2}}$ ▷provenance integrity hash-value

)

Message $\mathtt{R}_{Q3}$

$$
\begin{aligned}
\text{Let encryptedData}_{R_{Q3}} &= \{id_{Q3}||\text{getDataItem}(\mathtt{Q3})||\text{inAckTo}||id_{Q2}\}_{k'_{AU-PS}} \\
\text{Let integrityHash}_{R_{Q3}} &= \mathtt{h}(id_{Q3}||\text{getDataItem}(\mathtt{Q3})||\text{inAckTo}||\text{getHash}(\mathtt{Q2})) \\
\text{Let signatureWithHash}_{R_{Q3}} &= Sign_{k_{AU}^{-1}}(\mathtt{h}(id_{Q3}||\text{getDataItem}(\mathtt{Q3})||\text{inAckTo}|| \\
&\quad id_{Q2}||\text{integrityHash}_{R_{Q3}}))
\end{aligned}
$$

$\mathtt{R}_{Q3}$=

rpa(

encryptedData$_{R_{Q3}}$, ▷encrypted data

signatureWithHash$_{R_{Q3}}$), ▷message signature with hash

integrityHash$_{R_{Q3}}$ ▷provenance integrity hash-value

)

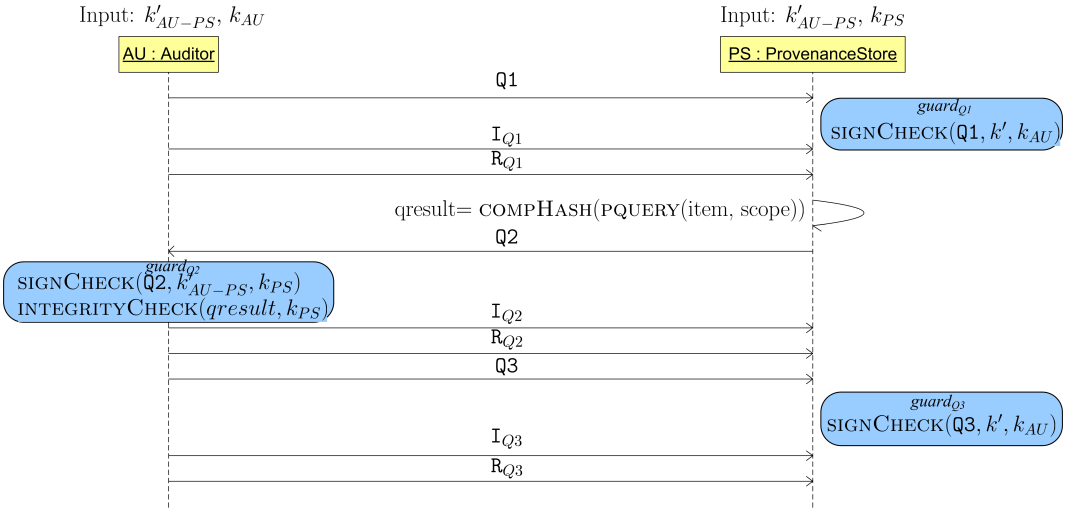


FIGURE 6.5: Query Request UMLSec Sequence Diagram

6.3.3.1 Verifying the Execution of Query Request

To verify the correct execution of the Query Request protocol, it contains several guards. These verify the integrity of the messages contained in this diagram, and are very similar to the guards explained in previous diagrams: each message signature and hash-values are verified and the content is decrypted to be later processed using Algorithms 8 and 9.

The only difference is in $\texttt{guard}_{Q2}$, which after verifying the signature of the message and decrypting it, it invokes the INTEGRITYCHECK procedure that verifies the integrity of the query results. If any problems are found, the compliance checking is not allowed to start according to the framework presented in Chapter 5.

6.4 Verifying the Secure Provenance-based Auditing Architecture

In the previous sections, we secured the entire provenance life cycle consisting of the recording, storage, querying and analysis stages. Taken together, these stages form the *Secure Provenance-based Auditing Architecture* that exhibits the four basic security characteristics (confidentiality, integrity, authentication, and non-repudiation) that we set out to establish at the beginning of this thesis.

To verify that these characteristics indeed hold, in this section we perform several types of attacks on this architecture, and demonstrate that these attacks fail.

To do this, we first discuss the Viki model checker we used to verify the architecture and give an example of its execution in Section 6.4.1.2 Then, in Section 6.4.2, we use the Viki model checker to verify that several known attacks are successfully thwarted by our architecture.

6.4.1 The Viki Model Checker

As mentioned in Section 2.3.2.3, the Viki model checker [80] can be used to verify that the security requirements expressed in UMLSec are maintained during the execution of the protocols presented in the previous sections. With that purpose, Viki takes as input a UMLSec sequence diagram and its associated adversary model, and returns the possible attacks that can be performed by the given attacker in the modelled protocol. Viki obtains the security requirements from the UMLSec elements and the predefined values used in sequence diagrams. These requirements are formalised in First-Order Logic and analysed with automatic theorem provers (Viki uses e-SETHEO [131] and SPASS [145]) to find flaws. If a flaw is found, a Prolog engine can be used to generate

the attack trace, which can provide a system designer with valuable insights on how to solve it.

The set of protocols presented in the previous sections are used as the input of the Viki tool to automatically generate attacks. In that way, we can ensure that these attacks are unsuccessful in our protocols. Since, in this context, a successful attack means that the system does not exhibit one of the security properties [125], Viki enables us to verify system security.

Before presenting the security verification process of our architecture, we first explain how Viki works. With that purpose, in the next section, we describe how to create the adversary model that is later used by Viki in the verification process.

6.4.1.1 The Adversary Model

To verify that the four security characteristics hold for the data exchanged in our sequence diagrams, we use UMLSec to define the information an adversary can get access to. This is achieved by the creation of an adversary model. The adversary model we use here represents an attacker that can eavesdrop, modify or insert messages into the communication channel with malicious intentions. This adversary model is based on an extended Dolev-Yao adversary model [50], in which an adversary is able to read all messages exchanged by the participants of a protocol to obtain sensitive information.

In the verification process that follows, we associate an adversary object with each sequence diagram to determine whether the modelled protocol indeed has the aforementioned security properties. To this end, the adversary is modelled by three set of values called `secret`, `initial knowledge` and `guard`. The values associated with `secret` are the data items that should be protected from the attacker. The values contained in the `initial knowledge` denote the information known by the attacker beforehand (e.g. public keys). Finally, the set `guard` contains the checks that are performed by the receiver of a message. These were defined in Section 6.2.2. Table 6.4 shows the `secret` and `initial knowledge` of the adversary models for each protocol.

Protocol	`secret`	`initial knowledge`
OTLS	k'_{C-S}	k_C, k_S, k_{CA}
Data Request	purpose, data and OK	k_{DC}, k_{DS}
Task Request	processData, result and OK	k_{DC}, k_{DP}
Query Request	item, scope, qresult and OK	k_{AU}, k_{PS}

TABLE 6.4: The `secret` and `initial knowledge` of the adversary objects used to verify our protocols

In what follows, we explain how Viki verifies the sequence diagrams with the corresponding adversary model.

6.4.1.2 An Example of the Viki Verification Process

In order to show how the Viki model checker can be used to verify a protocol modelled in UMLSec, we present an example taken from [79, 71]. In this example, which is shown in Figure 6.6, a flawed version of the OTLS protocol is used [19]. In this protocol, it is possible to perform a man-in-the-middle attack [139] by impersonating the `client`.

This UMLSec diagram is created using the notation described in Section 2.3.2.2 and acts as the input of the Viki model checker. As Figure 6.7 shows, Viki outputs the result "Proof Found", meaning that an attack was found. Put differently, the information contained in the secret set can become known by the attacker. This result is supported by the creation of a *Thousands of Problems for Theorem Provers* (TPTP) file [8], which shows how a man-in-the-middle attack can be performed. Thus, the attacker is able to obtain the secret data, which in this case is the symmetric session key.

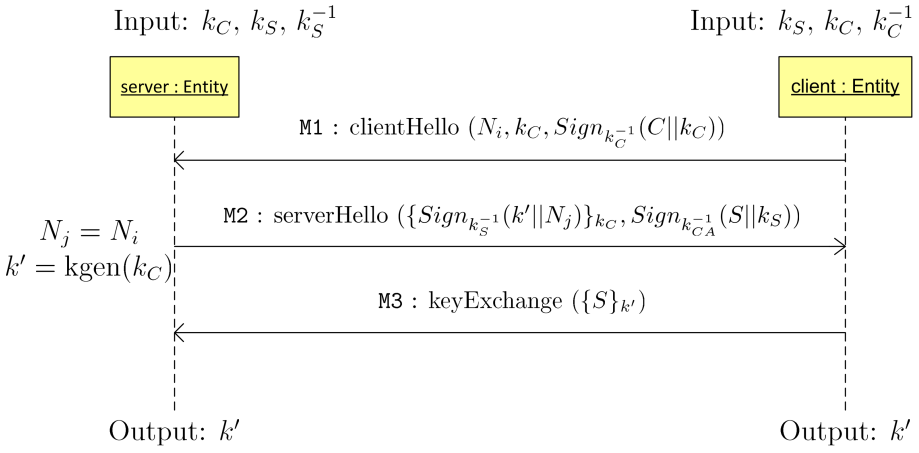


FIGURE 6.6: A version of OTLS with a security flaw

In more detail, Figure 6.8 depicts in the form of a sequence diagram the content of the corresponding TPTP file. This diagram shows how an attacker can perform the man-in-the-middle attack by impersonating the `client` and establishing communication with the `server`, while the `server` believes it is establishing communication with the `client`.

As presented in [71], this vulnerability can be removed by including the `client`'s public key (k_C) in the second message of the protocol. By including this parameter, a man-in-the-middle attack can be prevented and, as a result, an attacker will not be able to obtain the created secret key. Figure 6.9 presents the repaired version of the protocol. If we check this sequence diagram using Viki, we obtain the result "Completion Found" (See Figure 6.10). This means that the attacker cannot obtain the secret parameter (the symmetric session key) during the execution of this protocol.

[8] A TPTP file is produced by the TPTP library [132]. This library supports the testing and evaluation of Automated Theorem Proving(ATP) systems by defining general evaluation guidelines and using standard formats for inputs and outputs. Viki uses this library to analyse the possible attacks to the protocols.

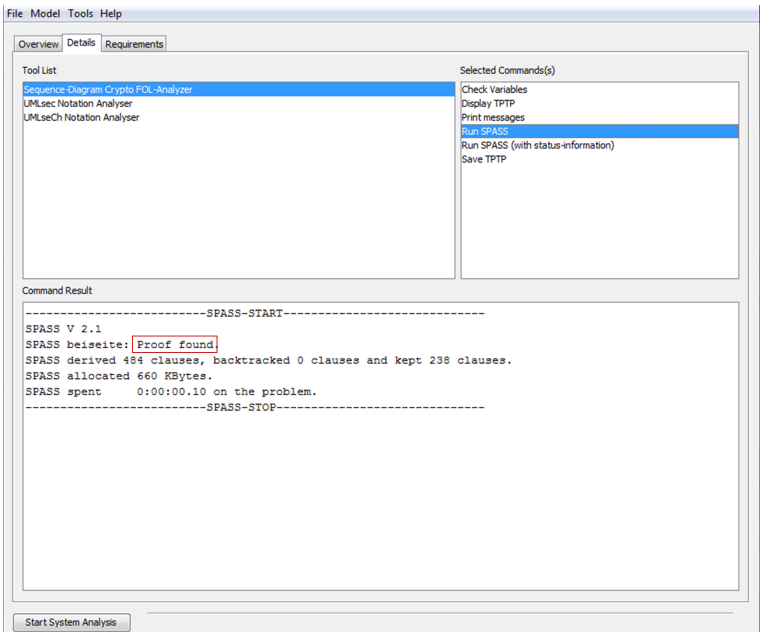


FIGURE 6.7: Viki verification result for the protocol in Figure 6.6

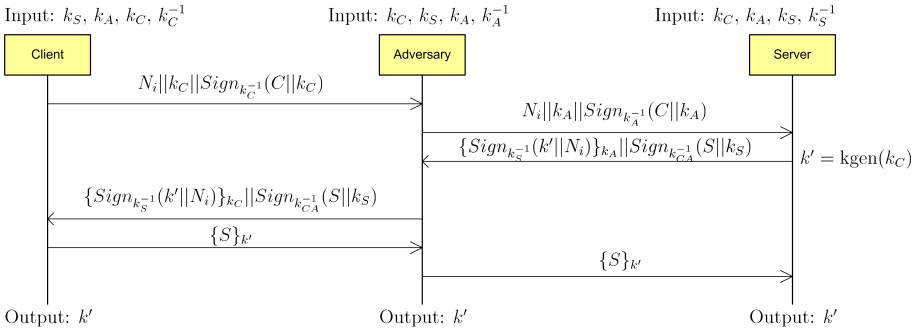


FIGURE 6.8: A Man-in-the-middle Attack on the OTLS Protocol with a security flaw

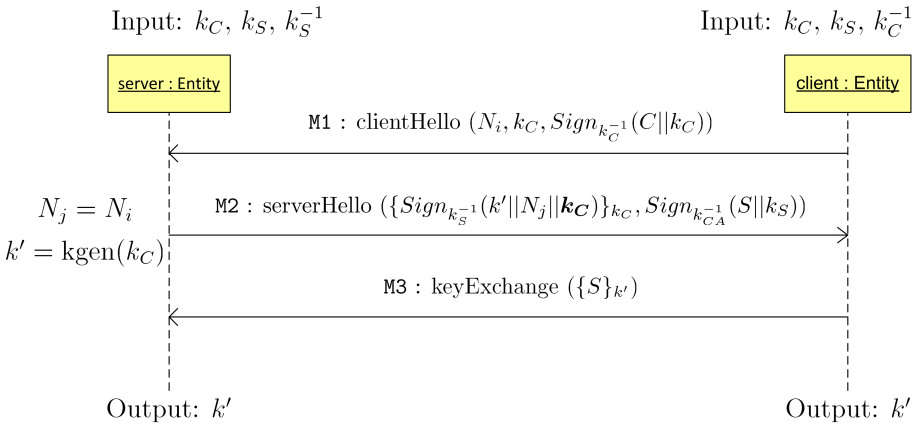


FIGURE 6.9: The OTLS protocol after repairing the security flaw present in Figure 6.6 (public key k_C is added to the second message)

In what follows, we verify the presented sequence diagrams using the Viki tool. This means that we evaluate the simultaneous execution of the adversary model and the

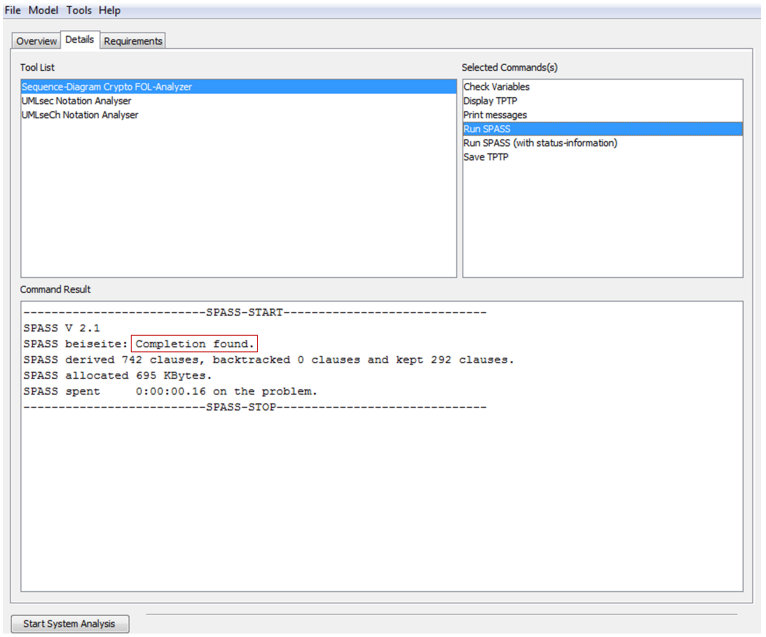


FIGURE 6.10: Viki verification result for the protocol in Figure 6.9

sequence diagram to identify whether attacks to the four basic security characteristics can be successfully performed.

6.4.2 Verification

To verify that the integrity property is maintained during the execution of the protocols, we use the Viki model checker discussed in the previous section. Using Viki, each sequence diagram and its corresponding adversary model are executed simultaneously to determine whether the adversary's attack was successful. In this context, a successful attack means that one of the basic security properties does not hold [125]. These attacks are defined as follows:

Definition 6.1 (Successful Confidentiality Attack)**.** A sequence of protocol transitions that lead to a piece of data contained in the `secret` set to be in the possession of the attacker.

Definition 6.2 (Successful Integrity Attack)**.** A sequence of protocol transitions that lead to a piece of data contained in the `secret` set to be modified without being noticed.

Definition 6.3 (Successful Non-Repudiation Attack)**.** A sequence of protocol transitions that lead to the possibility of denying sending of a piece of data contained in the `secret` set.

Definition 6.4 (Successful Authentication Attack)**.** A sequence of protocol transitions that lead to a piece of data contained in the `secret` set being sent in a message whose sender cannot be identified.

In what follows, we derive the set of properties that we will verify using Viki. If Viki finds no successful attacks, we can deduce that our protocols indeed exhibit these properties hold. If not, Viki gives the sequence of protocol transitions that invalidates the particular property.

Lemma 6.5 (Confidentiality Property)**.** *No successful confidentiality attack is possible for OTLS, Data Request, Task Request and Query Request Protocols.*

Lemma 6.5 states that an adversary is unable to obtain elements from the `secret` set during the execution of the OTLS, Data Request, Task Request and Query Request Protocols. This lemma is based on the fact that a piece of data contained in the set of secret values cannot be deduced by an adversary during the execution of the protocol. We verified these protocols using Viki, and we found that this lemma holds.

Lemma 6.6 (Integrity Property)**.** *No successful integrity attack is possible for OTLS, Data Request, Task Request and Query Request Protocols.*

Lemma 6.6 states that an adversary cannot modify a piece of data marked as `secret` during the execution of the OTLS, Data Request, Task Request and Query Request Protocols. This lemma is based on the collision resistant nature of the used cryptographic hash function, which guarantees that an adversary cannot alter the integrity of a piece of data (i.e. messages or assertions) without having a visible effect in the output. Moreover, an adversary can not insert a new piece of data without being detected. Using Viki, we verified these protocols and found that this lemma holds under the assumption that collision resistant hash functions are used.

Lemma 6.7 (Non-Repudiation Property)**.** *No successful non-repudiation attack is possible for OTLS, Data Request, Task Request and Query Request Protocols.*

Lemma 6.7 states that an adversary cannot deny having sent a message. This lemma relies on the trapdoor one-way function used in the signing algorithm and the collision resistant nature of the used hash function [96]. Both properties guarantee that an attacker cannot reproduce a signature of a message, if it does not possess the corresponding private key. Thus, assuming that the private key of the sender remains secret, we can guarantee that only the entity that owns the private key is able to sign a message. Using Viki, we verified that the modelled protocols are impervious to a non-repudiation attack. Therefore, this lemma holds.

Lemma 6.8 (Authentication Property)**.** *No successful authentication attack is possible for OTLS, Data Request, Task Request and Query Request Protocols.*

Lemma 6.8 states that an adversary cannot participate in the architecture without first being properly authenticated. The lemma relies on the properties provided by the used

authentication protocol, which in this case is OTLS. OTLS provide a multitude of security measures, such as, nonce numbers to avoid man-in-the-middle attacks, the implementation of a hand shake protocol to create symmetric session keys and the use of certificates issued by certified authorities [27]. By again using Viki, we found that an authentication attack on the protocols is not possible, thereby proving this lemma.

6.4.3 Securing Provenance

Now that we have proved that all protocols in the Provenance Based-Auditing Architecture are secure, we can show that the architecture itself is also secure. As we explained in Section 6.4.2, the Data Request, Task Request and Query Request protocols (Section 6.2.1) are secure, since they exhibit the properties of integrity, confidentiality, non-repudiation and authentication. Thus, based on this, we can derive the next theorem:

Theorem 6.9 (Secure Provenance Based-Auditing Architecture)**.** *A Provenance Based-Auditing Architecture is secure, since the properties of confidentiality, integrity, non-repudiation and authentication hold.*

Theorem 6.9 relies on the property lemmas derived from each of the protocols of the architecture, which were defined in the previous section. If each of the properties hold in all protocols of our architecture, we have a Secure Provenance Based-Auditing Architecture.

Since Theorem 6.9 holds for the Secured Provenance-based Auditing Architecture, we can conclude that, under the given assumptions, the architecture is secure and, therefore, the provenance information created by it is secure as well. Thus, the results derived from the analysis of this secured provenance data are based on correct information.

6.5 Discussion

Securing provenance is critical for making systems accountable and for measuring quality and trust in systems [103]. For that reason, researchers have developed different techniques to prevent (malicious) alteration of provenance while it is created, transported, recorded and queried. In doing so, trust in the result derived from its analysis can be significantly increased. Hence, researchers have sought to address the challenge of securing provenance information [135, 67, 31, 148, 68].

Tan *et al.* [135] expose and discuss the problem of securing provenance in a SOA-based provenance system. To ensure accountability, liability and integrity of provenance assertions, they use digital signatures providing non-repudiation. They discuss basic security issues within provenance systems and mention access control as a possible solution, but do not explain how this should be implemented in practice. Moreover, their

proposed solution does not provide all four basic security characteristics; they guarantee non-repudiation and integrity, but not confidentiality and authentication.

Hasan *et al.* [67] argue that the importance of securing provenance has not been sufficiently acknowledged within the fields of digital forensics, law, and regulatory compliance. They identify integrity, availability and confidentiality as the main properties that a provenance-aware system should exhibit to provide trustworthy provenance. They propose a provenance model [68], in which provenance is defined as the record of actions taken on a particular document over its lifetime. Here, a document can be a file, a database tuple or a network packet. In this approach, the provenance of a document is represented by a "provenance chain", which is a linear time-ordered sequence of provenance records pertaining to a specific version of the document. Because of this, in contrast to our DAG representation, non-linear relations are not expressible in their model. Moreover, since their approach attaches provenance to a document itself, relations that may exist with other documents are not recorded. Their method of guaranteeing the integrity of the provenance of a document is similar to ours, i.e. relations between subsequent versions of a document are secured by including hashes of the ancestors into the document. The difference is that, in our approach, only the hashes of the predecessors are considered, instead of all ancestors of a node. By doing so, our method is more scalable, since only references to the previous operations performed on a piece of data are stored, instead of references to its entire history.

Braun *et al.* [31] also discuss the problem of securing provenance. In their work, they use a similar model to PASOA [64], the model used in this thesis, in which provenance is represented as a graph. They argue provenance information differs from traditional data and, therefore, the existing security models used to protect "traditional information" do not apply to graphs and are therefore not easy to extend. Consequently, they present the challenge of constructing a security model for provenance graphs. This model should secure data and relationships in the graph, as the approach presented here. In contrast to our work, however, they only focus on how to represent different levels of access control on the elements of a provenance graph. In our approach this is addressed by limiting access to the information contained in the provenance DAG to the auditor, who is a trustworthy actor. However, users can decide to record references to the original data in the provenance assertions, restricting access to the information contained on the DAG. In that way, just the actors who have the right credentials can access this information.

Xu *et al.* [148] discuss the problem of managing provenance systems. They define several desirable requirements that secure provenance-aware systems should exhibit: access control, integrity, accountability and privacy protection. In order to satisfy these requirements they propose a framework which contains a layer that maintains the integrity of data and provenance information during storage, transferring and processing.

As in our solution, they argue that integrity of both data and provenance information is important. However, no concrete solution to this problem is presented.

6.6 Conclusions

In this chapter we secured the Provenance-based Auditing Architecture by providing the properties of confidentiality, integrity, authenticity and non-repudiation to the entire provenance life cycle of recording, storage, querying and analysis. To do this, we applied cryptographic techniques to the provenance-aware protocols of Data Request, Task Request and Query Request that we developed in Chapter 4, and showed how these techniques can be used to verify the correct execution of these protocols, i.e. that all four aforementioned properties hold. Furthermore, to ensure that provenance DAGs cannot be maliciously altered during the analysis stage, we developed the Secured Provenance Graph. This graph contains specially designed hash-values in each of its nodes along with a signature from the Provenance Store, which makes possible to detected malicious alterations. Using the Integrity Checking Algorithm, we demonstrated that the integrity of this graph can be easily verified.

We extensively and methodically analysed each protocol individually, and proved that the presented techniques indeed support the four security requirements by conducting an automatic verification of the presented protocols. Specifically, using the Viki automatic model checker, we verified that these properties hold during the execution of the protocols. Moreover, we demonstrated that we can verify that the provenance information, which is recorded in and queried from the Provenance Store, has not been tampered with. By offering these security properties, we can guarantee query results obtained from the Provenance Store are correct. Consequently, the analysis performed by the verification algorithms of the Compliance Framework developed in Chapter 5 is based on secure and correct DAGs.

From a more practical point of view, the method presented in this chapter can be easily implemented by adding the corresponding security functions to the library that provides provenance functionality. Indeed, to demonstrate that our methods are practically viable, we developed a working proof of concept by augmenting the PReServ library [62] (which allows developers to make applications provenance aware) with the algorithms described in this chapter.

In more detail, we implemented a Java program that simulates the scenario in Section 3.1, involving four actors: two data subjects that communicate with a single data controller, which, in turn, communicates with a data processor. The program runs a single process (i.e. it is not distributed), in which messages are (locally) exchanged using TCP/IP. Public-private key pairs were created for each actor before the execution of the scenario, which are stored on the local file system without any protection. Session

keys were created during the execution of the OTLS protocol and used to encrypt data sent over the communication channels, and decrypt data when it is received. Since key management and access control are outside the scope of this thesis, these aspects were not implemented in the prototype.

To test the correctness of our protocols, we performed a number of tests, in which various messages were altered during transmission or after storage in the provenance store. After the termination of each test, we ran Algorithm 9 from Section 6.2.2 on the assertions created in the provenance store in order to detect possible anomalies. The tests we performed involved the following:

- No (malicious) alterations
- An alteration of a piece of application data (e.g. the date of birth)
- An alteration in the name of a relationship between data (e.g. wasGeneratedBy)
- An alteration of the endpoints of a relationship (e.g. result wasGeneratedBy t_2, instead of result wasGeneratedBy t_1)

In all tests, the algorithms performed correctly: in the first case, no anomalies were detected (as expected), while in the second, third and forth cases, Algorithm 9 terminated with the error message "hash value is incorrect".

We also executed several types of provenance queries (as in Section 4.4), and verified that the result of these queries was not (maliciously) altered after creation. In more detail, we performed the same four tests as mentioned above. Here, again, our algorithms performed as expected. In the first case, no anomalies were detected, while in the second, third and forth cases, Algorithm 10 terminated with the error message "integrity is compromised in *id*", where *id* is the identifier of the specific element that was altered.

Needless to say, in its current form, our prototype has a number of limitations that restrict its application in the real world. First, the system should be made distributed, such that the various actors can run on different computers. This enables us to ascertain how well the system scales under conditions where information and actors are (physically) distributed. Second, access control to the provenance store needs to be implemented to prevent unauthorised access to the provenance assertions, because these contain (references to) private information. Third, keys should be properly managed (which involves creation, safe storage, distribution and revocation) by all actors.

Of these limitations, key management is perhaps the most critical one, since all required security properties of confidentiality, integrity, authentication and non-repudiation cannot be maintained without it. In light of this, we briefly discuss the challenges of key management, and how these challenges can be addressed in our system.

In more detail, there are three types of keys that are used in our proposed system that need to be taken in to consideration when implementing key management: session keys (such as k'_{C-S} in Figure 6.1), private keys and public keys (such as k_C^{-1} and k_C in Figure 6.1). Session keys should be treated differently from private and public keys.

Specifically, since session keys are used only once during the execution of a protocol between two actors, they do not need to be stored after the protocol has terminated. In more detail, the life time of a session key is the time it takes to execute a protocol. It is created at the start of the protocol using OTLS, and is used to encrypt information sent over the insecure channel, and decrypted immediately on receipt by the receiving actor. To protect this information after it is stored in the provenance store, it should be encrypted (using a different key than the session key) and access control should be implemented to prevent unauthorised access.

Compared to session keys, private and public keys have a much longer life. As a result, they need to be carefully stored over a longer period of time. Depending on the length of this period, two possibilities exist: actors generate a single private-public key pair which is used permanently, or they periodically discard their keys and generate new ones. In both cases, public keys of all actors can be stored in our Provenance Store, which is a central system that all actors have access to. Private keys need to be securely stored by the actors themselves. However, since absolute security does not exist, there is always a risk of a private key becoming compromised. As a result, the solution of permanently using a single key pair is more vulnerable than periodically generating new keys. However, the latter solution introduces a number of challenges, each of which needs to be addressed carefully and securely:

Key revocation When a new key pair is generated, all actors should be made aware that the key pair is no longer valid; accepting a signature made by a key that has been revoked is a security risk.

Key distribution Up-to-date public keys should be accessible by all actors at all times. These keys should also be verified and signed by a trusted third party to create a digital certificate, so that actors cannot deny having sent a message (non-repudiation).

Maintaining key history A history of public keys needs to be kept in order to ensure that signatures made using expired keys can still be verified, and to ensure the non-repudiation of all provenance information collected over time.

The most common way to address these challenges is the use of a Public Key Infrastructure (PKI) [17]. The responsibility of a PKI is the secure creation, management, distribution, usage, storage, and revocation of public-private pairs and their corresponding digital certificates. Of all PKIs currently existence, X.509 [14] is the most widely

used and proven system. In addition, it addresses the third challenge by maintaining a complete key history. It is therefore a strong candidate for managing keys in a prototype or deployed version of our system. However, as mentioned earlier, we consider this outside the scope of this thesis and part of future work.

Thus far, we have given formal proofs of the correctness of the individual protocols. There are several underlying assumptions that allow us to derive these proofs. However, in practise, some of these assumptions might not hold. For example, one of the assumptions we made in this chapter is that provenance information is always available to be used in the compliance analysis. In the next chapter, we discuss what happens when this assumption (and others) are dropped. More specifically, we present an analysis of the attacks that a Provenance-based Auditing System can be subjected to, explaining which of them are thwarted by the techniques developed in this thesis, and which of them need additional measures to avoid.

Chapter 7

System Evaluation

In the previous three chapters, we developed the three main contributions of this thesis: the Provenance-based Auditing Architecture (Chapter 4) for capturing provenance of processing of personal information, the Compliance Framework (Chapter 5) for verifying that personal information was processed in compliance with a set of processing rules, and the Secure Provenance-based Auditing Architecture (Chapter 6) that secures the entire provenance life cycle of recording, storage, querying and analysis. Taken together, these contributions ensure trustworthy audits of personal data processing.

However, until now, we have considered fairly idealised conditions under which a system that implements these three contributions (i.e. build according to the Secure Provenance-based Auditing Architecture) operates. In practise, these systems do not operate in isolation. Rather they interact with each other, have multiple actors, and are distributed. Therefore, it is necessary to identify the strengths and weaknesses of such systems under more practical and realistic conditions.

To do this, we explore the effect of the most common (and known) attacks on our system in an attempt to break the four basic security properties, and an additional one (for reasons that will become clear later on): confidentiality, integrity, authentication, non-repudiation, and availability. To facilitate this evaluation, we employ Attack Trees [123], a technique that is specifically designed to evaluate the security of an information system. In so doing, we determine whether our contributions are capable of protecting both application data and provenance. Moreover, if they fall short in this respect, we identify what is required to mitigate these new risks.

The main contribution of this chapter is to show that the Secured Provenance-based Architecture is resistant to a set of important attacks on the basic security properties under the assumption that vulnerabilities in the underlying operating system and attacks from malware are addressed elsewhere.

This chapter is organised as follows. First, in Section 7.1 we give an overview of Attack Trees and their underlying methodology. Next, in Section 7.2, we define the system that we analyse in this chapter, which is based on the architecture described in Chapter 4. Then, in Section 7.3, we give a summary of known attacks grouped by the aforementioned security properties they aim to break. Given these attacks, in Section 7.4, we present five attack trees and their corresponding analysis. Finally, we conclude in Section 7.5.

7.1 Attack Trees

Attack Trees provide a formal, methodical way of testing the security of systems, using a variety of attacks [123]. These attacks are represented as a tree structure in which the root node is the goal of the attack, and the leafs are the ways in which the goal can be achieved. Each node is a subgoal of its parent, and, in turn, its children are ways to achieve that subgoal. Attack trees have two types of nodes, AND and OR. The first type, AND nodes, are used to represent all steps that need to succeed to achieving a goal, whereas the second type, OR nodes, are used to represent alternatives. Thus, an OR node succeeds if one of its children succeeds, while an AND node succeeds if all of its children succeed.

Each node can also contain additional information about the goal it represents. Examples of additional information include whether the goal is possible or not (given the current configuration of the system), the cost of achieving the goal, and whether the attacker needs special resources to achieve it. Additionally, in our case, this information also includes whether the attack is prevented by one of the techniques developed in the previous chapters, or be detected by one of our assumptions from Section 3.6.

The aim of using Attack Trees is to understand all different ways in which a system can be attacked and to design countermeasures to prevent such attacks. Attack Trees also allow us to understand who the attackers are, and what their abilities, motivations, and goals are.

Now, in order to create Attack Trees to test the security of a system, Schneier [123] proposes a systematic methodology, which is presented below.

1. Identify all possible attack goals (threats) by considering the weaknesses of each component, property, layer, etc. of the system. Each attack goal is the root of a tree, and trees can share subtrees and nodes. Thus, the number of trees is equal to the number of attack goals.

2. Create attack trees by enumerating all possible ways in which the attack goals can be achieved. This is an iterative process which needs to be repeated until all

possible ways are described. If the system changes, attacks can be added (based on its new weaknesses) or removed (depending on whether they still apply).

3. For each possible attack goal, analyse the circumstances under which each leaf of the corresponding tree can be reached by creating scenarios for successful achievement of the goal. To do this, prerequisites, assumptions and the attacker capabilities should be included in these scenarios. Then, we decide how feasible each attack goal is and what could happen if the conditions in the scenario change.

4. After identifying all feasible attacks, we identify the means by which all these attacks can be prevented in a given scenario.

Using this methodology, we can define the security assumptions of a system and determine if a system is vulnerable to a particular type of attack. For that reason, Attack Trees are especially useful to perform architecture risk analysis.

A very useful characteristic of the attack trees is that they capture knowledge in a scalable and reusable form. The trees can be modified at the same time as systems change, and also can be reused by systems that have similar characteristics. However, one drawback of this technique is that there is no guarantee of completeness. Thus, a collection of attack trees does not necessarily represent all possible attacks. However, by regularly evaluating the system, unrepresented attacks may be found. In sum, Attack Trees help to conduct a high level analysis of security risks in a system to make the right choices to mitigate or avoid such risks.

In Section 7.3 we perform steps 1 and 2 of the methodology described above. Then, in Section 7.4, we perform steps 3 and 4. First, however, we describe the system we subject to analysis using the Attack Tree technique.

7.2 System Definition

The systems we analyse in this chapter are based on the Secure Provenance-based Auditing Architecture defined in Chapters 4 and 6. Each system consists of one or more Data Subjects and one or more Data Processors, and interacts with other systems with the same properties and structure. This leads to the interaction of various Data Controllers.

As mentioned in the initial assumptions (see Section 3.6), all entities record provenance truthfully. To enforce this, these entities can, for example, record provenance automatically using a library which can not be (maliciously) altered by its users.

A system with these characteristics presents a variety of security risks, which are mainly related to the communication between entities, recording of provenance information and

storage of information. These risks are analysed in our attack trees to show how the secure architecture mitigates them.

It is important to mention that this analysis focuses on the technical security risks that our system is exposed to. All the security issues related to the end-user, such as lost passwords or keys, creation of weak passwords or physical security, are outside the scope of this analysis. However, needless to say, administrators should implement the necessary measures to prevent and avoid these issues that could impact the security of the system.

7.3 System Attacks Analysis

Now that we have defined the system that can be subject to various attacks, we now describe the actual attacks that can be performed against our architecture, which would compromise the security of the audit process. By doing so, we cover steps 1 and 2 of the Attack Tree Methodology.

As mentioned in Chapters 4, 5 and 6, we wish to guarantee four security properties: *integrity*, *confidentiality*, *authentication* and *non-repudiation*. In this section, we add a fifth property: *availability*, which guarantees information is available when needed. Even though this property is out of the scope of this thesis, it is important because the lack of availability can have a detrimental impact on the execution of our system. For example, if for some reason the Provenance Store is not available due to an attack, the system will be not able to record provenance, resulting in either an incomplete record of provenance information, or a severe delay the execution of the application protocols (Data Request and Task Request). Therefore, the attacks that we define in the remainder of this section focus on breaking each of these five properties.

In the next sections, we describe what an attack on each property entails, and enumerate the possible attacks that can compromise these properties. These attacks are used later on to create the Attacks Trees, which are presented in Section 7.4.

7.3.1 Confidentiality Attacks

A successful attack on the confidentiality property means that information regarded as "confidential" is accessed by an attacker. Since all information in our system is encrypted before being transported, to breach confidentiality, an attacker needs to obtain (one or more) private keys to decrypt previously stolen information or needs to obtain (on or more) passwords to access a storage component containing confidential information. In the list below, we enumerate the ways in which an attacker can accomplish this.

Related-key Attacks In this attack, an adversary uses cryptanalysis techniques to observe the operation of a cipher to determine which keys were used or recently created. Examples of this are known-key attacks and compromised-key attacks [96]. These attacks can usually be performed on badly designed encryption algorithms. In such cases, keys can be guessed if the key generation algorithm creates weak keys or uses weak random number generators. One of our assumptions is that only well-designed encryption algorithms are used.

Password-based Attacks This attack takes place when an adversary obtains a valid password of a valid user, thereby gaining the rights of this a user, including those that allow access to confidential information. Examples of these attacks are password guessing attack, dictionary attack and brute force attack. The best way to avoid these attacks is implementing precomputation techniques (such as hashing passwords), salting techniques and requiring strong passwords [96]. Another important issue is the proper protection of passwords by users, such that they can not be easily stolen. In this case, we work under the assumption that user passwords are sufficiently strong and properly protected.

Sniffer Attack A sniffer or packet analyser is computer hardware or software that can intercept and log traffic passing through a network. These can be used to eavesdrop on the communication channel, for example to perform related-key attacks or attacks to break encryption algorithms. Once the attacker gets a valid key, all encrypted traffic can be decrypted. Here, we assume all the necessary physical measures to avoid this type of attack are taken.

Data Driven Attacks In this attack, malicious code is embedded in innocuous-looking data in an attempt to attack the system in different ways. Normally, this code is not recognised by firewalls so it is easily introduced into the system. We assume that our system is well protected against this type of attacks. Examples of these attacks are:

(a) Trojans: These are pieces of software that give unauthorised access to an adversary. In the case of breaking confidentiality, they are used to conduct password or key theft by unauthorised installation of software, unauthorised downloading and uploading files, key stroke logging (record what is typed by the user), etc.

(b) Trapdoors: These are weaknesses in software that is running on the system that allows someone with knowledge of these weaknesses to gain unauthorised access to confidential data. These trapdoors are used by programmers for security purposes to debug and test programs, but are not properly protected in production systems.

7.3.2 Integrity Attacks

A successful integrity attack means that information that has been transported or stored has been (maliciously) altered. To break the integrity property, an attacker needs to change the content of a sent message or the information stored in a persistent storage. The attacks that can be conducted to compromise information integrity are enumerated below:

Unauthorised Data Message Modification Attack In this attack, an adversary modifies the data contained in a message without being noticed. This directly affects the integrity of the information transported in a communication channel. To preserve the data integrity, it is important to use hash algorithms in messages (for example, as we did in Chapter 6). Moreover, it is important that the chosen hash algorithm is well designed to avoid hash attacks, which are described below.

Hash Attacks This type of attacks can take place if the implemented hash function is badly designed. In such cases, attacks such as Birthday Attack, Yuval's Birthday Attack, Pseudo-collision Attack or Chaining Attacks [96] can take place and compromise the integrity of data. One of the assumptions of our work is the use of a strong, well-designed hash function.

Password-based Attacks / Trojans / Trapdoors These attacks, which were explained above, can also be used by an attacker to modify or delete files, thereby affecting their integrity. Again, we assume that our system is well protected against these.

7.3.3 Authentication Attacks

A successful attack on the authentication property means that an attacker gains the ability to impersonate a valid user. To do this, the attacker needs to acquire some form of identification that belongs to such valid user, such as a private key or a password. In more detail, the attacks that can be conducted by an attacker to compromise authentication are as follows:

Impersonation In this type of attacks, an adversary assumes the identity of one of the legitimates parties in a system. These attacks include:

(a) Man in the middle (MITM): This is an attack in which a third person impersonates one of the parties involved in a communication protocol. To avoid this, is necessary to create a secure communication channel and freshly created nonces, which is done in most of authentication protocols. It is important to mention that the system is verified to be resistant to this attack by the model checker Viki in the protocols presented in Chapter 6.

(b) Interleaving attack, Reflection Attack, Misplaced trust in a server, Replay attack [96]: These attacks are a consequence of badly designed authentication protocols. As such, they are not discussed here, since one of our assumptions is the use of well designed protocols, such as TLS, OTLS or SSL, which have been widely analysed and used. Moreover, the protocols presented in Chapter 6 were verified by Viki and found to be resistant to this type of attack.

(c) Obtain keys of a valid user: When keys are used as a proof of identity, an attack can focus on acquiring such keys. To do this, an attacker can use various techniques, such as data driven attacks (trojans, viruses, etc), related key attacks or use password-based attacks, which were explained in Section 7.3.1.

Transitive Trust In this attack, an adversary obtains a user account using some of the means already described. This account is then used to get more or higher-level privileges or to create new accounts with administrator's privileges. This attack compromises access control to resources (information, equipment, identities) and, thereby, the integrity and confidentiality of information. The identity of the authorised user is also compromised. We assume that all the necessary access control techniques are implemented to protect the resources of our system (see Section 3.6).

7.3.4 Non-Repudiation Attacks

A successful attack on the non-repudiation property means that an attacker can modify the evidence of the sending and receipt of a message in order to deny having sent or received this message. To do this, the attacker needs to obtain the identity of a valid actor using any of the attacks presented in the previous section, or steal the private keys of a valid user to sign messages in his or her name. The attacks that can be conducted by an attacker to compromise non-repudiation are listed below:

Impersonation This attack involves an attacker assuming the identity of a valid actor, to be able to sign and send messages on its behalf.

Data driven attacks This attack involves an attacker stealing private keys or certificates of a valid actor in order to obtain its identity.

7.3.5 Availability Attacks

A successful attack on the availability of a system means that an attacker is able to prevent authorised users to access a system resource. To do this, the attacker saturates a component of the system with external communication requests forcing it to reset or

expend significant computation or bandwidth. This way, the component can no longer serve its intended purpose. These attacks also obstruct the communication between the intended users and the component such that adequate communication becomes impossible. In more detail, the following are common availability attacks:

Denial of Service (DoS) and Distributed Denial of Service (DDoS) These attacks compromise the availability of information or services in a system or network resource, and is a well-investigated issue [39, 96, 15]. One of our assumptions is that all the necessary prevention measures have been implemented to avoid these kind of attacks [39]. These include implementing route filters, disabling unused network services, using redundant and fault tolerant network configuration, etc.

Trojans These attacks were previously explained as a means to break confidentiality. However, they can also be used to conduct DDoS attacks by using infected computers to simultaneously launch a DoS attack on a predefined target. As previously stated, we assume the system is protected against these attacks.

Now that we have defined all possible attacks, and have identified the assumptions under which the security analysis of our system will be performed, we can now formalise these using Attack Trees.

7.4 Attack Trees Analysis

In this section, we create and analyse the Attack Trees using the previously presented attack analysis. Thus, in this section we cover steps 3 and 4 of the Attack Trees Methodology discussed in Section 7.1.

Corresponding to each of the five security properties discussed in the previous section, we create five attack trees with the following attack goals: disclosure of confidential information, compromise data integrity, compromise authentication, compromise non-repudiation, and compromise system availability.

Before presenting the analysis of our Attack Trees, we present a set of security assumptions under which this analysis is made. To differentiate these assumptions from the ones defined in Section 3.6, the security assumptions are labelled with the letter S.

S1 All cryptographic algorithms and protocols used in the architecture are well designed and cannot be forged. This includes encryption, digital signature and hash algorithms, as well as authentication protocols.

S2 Access Control measures are implemented in all system resources, including databases, Provenance Stores and processing rules.

S3 All information related to keys, certificates and passwords is securely stored and protected. Also, keys, certificates and passwords are created using well-designed methods.

S4 All necessary prevention measures to avoid "Sniffer Attacks", "Data Driven Attacks", "DoS" and "DDoS" have been implemented.

S5 We assume that all the actors use a library that allow them to automatically record the provenance information we identified in Section 4.2. This library cannot be deleted or accessed by attackers to intentionally modify its behaviour. The same applies to the Usage Rules Definition (see Section 5.2), which cannot be deleted or modified without previous authorisation of the administrator in charge.

Now, in the attack trees presented in this section, each node is given a colour. The blue nodes are attacks that are not addressed in this thesis, and considered *outside the scope* of our work. These attacks mainly relate to availability and authentication. The red nodes are attacks that can be prevented by the assumptions listed above. These nodes contain the labels of the corresponding assumptions. Finally, the green nodes are the attacks are *within the scope of our work*, i.e. those that are addressed by the contributions presented in this thesis. In these nodes, we include the corresponding Lemma from Section 6.4 that addresses the corresponding attack.

In the upcoming subsections, we present the Attack Trees corresponding to the attacks identified in the previous section.

7.4.1 Information Confidentiality Attack Tree

The Information Confidentiality Attack Tree is presented in Figure 7.1. This tree shows attacks that can lead to the disclosure of confidential information.

In this figure, the nodes labelled a, b and c represent attacks that are prevented by the previously defined security assumptions: "Unauthorised access to data stored in the Provenance Store", "Unauthorised access to data stored in the Local Database" and "Unauthorised access to data usage rules". These three attacks are related to the Access Control of the databases in our architecture. We assume these databases implement the techniques discussed in 3.4.9 to prevent these attacks (Assumption S2).

The nodes labelled d and e, which are within the scope of our work, differentiate between the disclosure of data in the provenance assertions and the disclosure of data during communication.

The attack corresponding to node d, "Disclosure of data in the assertions", is addressed by Lemma 6.5. The goal of d can be accomplished by two different attacks: "Cross-reference" in node $d.1$ and "Inference" in node $d.2$. In the "Cross-reference" attack,

an attacker, who can see the provenance that different entities are recording, combines information from different sources to obtain private information that was not present in the sources individually. In the "Inference" attack, an intruder uses this information to infer new private information. To see why these attacks fail in our system, recall that the information contained in provenance assertions can either be a copy of application data or a reference to it. Now, if only references are maintained in these assertions, only the auditor should have access to the original information. Thus, in this approach, both attacks (cross-referencing and inference), can be prevented since attackers only have access to references, which they can not resolve as they do not have the credentials to access the original information.

The attack corresponding to node e, "Disclosure of data in the communication", has three ways of succeeding: $e.1$ "Impersonation", $e.2$ "Sniffer Attack" and $e.3$ "Data driven attacks". As mentioned in Assumption S4, we assume the necessary measures have been taken to avoid the latter two attacks ($e.2$ and $e.3$) are successful.

To prevent attack $e.1$, "Impersonation", we first need to ensure that all the keys and password are created using well-designed cryptographic algorithms and are stored in a secure place. This is guaranteed by Assumption S3. Furthermore, we need to guarantee that our communication protocol cannot be attacked. This was verified in Chapter 6 by formalising the protocol and the communication in the architecture (Lemma 6.5).

Therefore, following the analysis of this tree, we can ensure that confidentiality attacks are unsuccessful under the given security assumptions.

7.4.2 Information Integrity Attack Tree

The Information Integrity Attack Tree is presented in Figure 7.2, and shows the possible attacks that compromise the integrity of information.

In Figure 7.2, there are two nodes, b and c, that are prevented by the assumptions: "Unauthorised manipulation of data stored in the local databases" and "Unauthorised manipulation of data usage rules". As previously mentioned, these attacks are related to Access Control of the databases, for which we assume the necessary techniques have been implemented (Assumption S2). Moreover, Assumption S5 makes tampering with the data usage rules not possible, and ensures that provenance is securely recorded.

The remaining nodes are within the scope of our work: a "Unauthorised manipulation of data stored in the PS", d "Unauthorised manipulation of assertions" and e "Unauthorised manipulation of data in the communication". These nodes represent attacks that compromise the integrity of data contained in the Provenance Store, data contained in assertions and data contained in the messages, respectively. These attacks are addressed in Lemma 6.6, which was presented in Chapter 6. More specifically, the goals

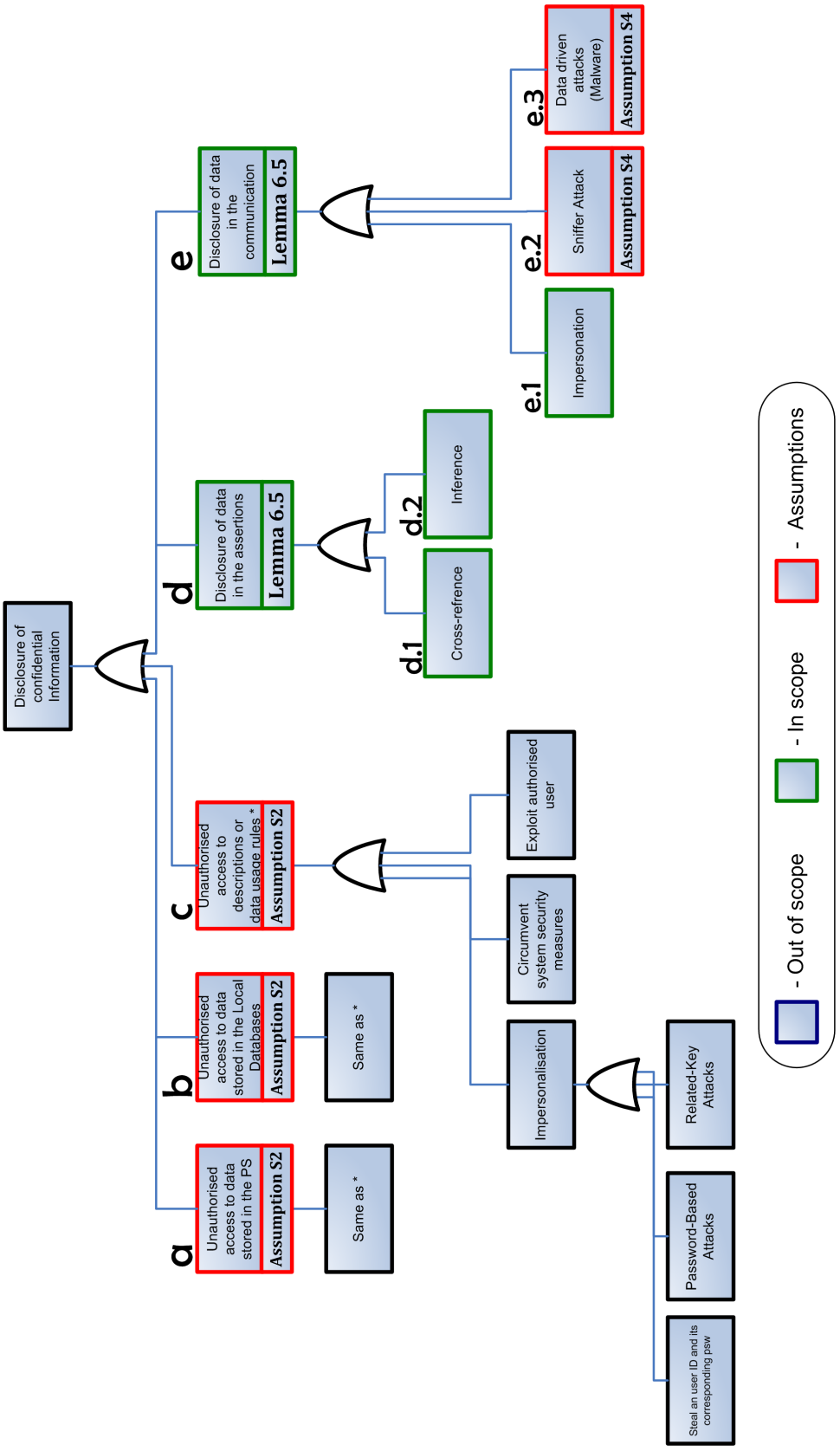


FIGURE 7.1: Information Confidentiality Attack Tree

associated to node a, d and e cannot be accomplished because the cryptographically protected p-assertions (see Section 6.2.1) make malicious alteration of the information stored in the Provenance Store not possible. However, cryptographic protection is based on several additional assumptions. For node a, these are the same as for node c, which was previously discussed.

Node d contains three different attacks. The first, d.1, "Replicate plain data", is addressed by the Secure Provenance Graph (Section 6.3.1), which prevents corruption of the integrity of the information and relations contained in this graph. The second, d.2, "Impersonation", is addressed in the formalisation and verification our communication protocol (as discussed in Section 7.4.1). The last one, d.3 "Hash-attacks" is prevented by Assumption S1.

In node e, "Unauthorised manipulation of data in the communication", we guarantee the integrity of data by implementing the OTLS protocol, which creates a secure communication channel. Since we assume that the encryption algorithms and hash functions used within this protocol are well-designed (Assumption S1 and S2), the nodes e.1 "Encryption Attacks" and e.2 "Hash-attacks" are coloured red. Finally, the attack corresponding to node e.3, "Impersonation", was addressed in Section 7.4.1.

Since all five nodes represent unsuccessful attacks, we can ensure that, under the given assumptions, the integrity of information is maintained.

7.4.3 Authentication Attack Tree

The Information Authentication Attack Tree is depicted in Figure 7.3, showing attacks to compromise authentication in the system.

Analysing this tree, we initially have nodes a, "Compromise assertions end-point authentication", b, "Compromise entities mutual authentication", and c, "Multiple user or system identities". Nodes a and b are related to the different properties that authentication protocols and algorithms can provide to the information: end-point authentication and mutual authentication.

In node a, end-point authentication is used to identify which entity sends a message. In the attack associated with this node, we only need to prevent the "Impersonation" attack (which was discussed in Section 7.4.1), assuming that the used hash techniques are well-designed (Assumption S1) and that the necessary measures to avoid data driven attacks have been implemented (Assumption S4).

In node b, mutual authentication verifies the identity of two or more users that established communication. This is addressed by the implementation of the OTLS protocol, which was formalised in Chapter 6, specifically in Lemma 6.8.

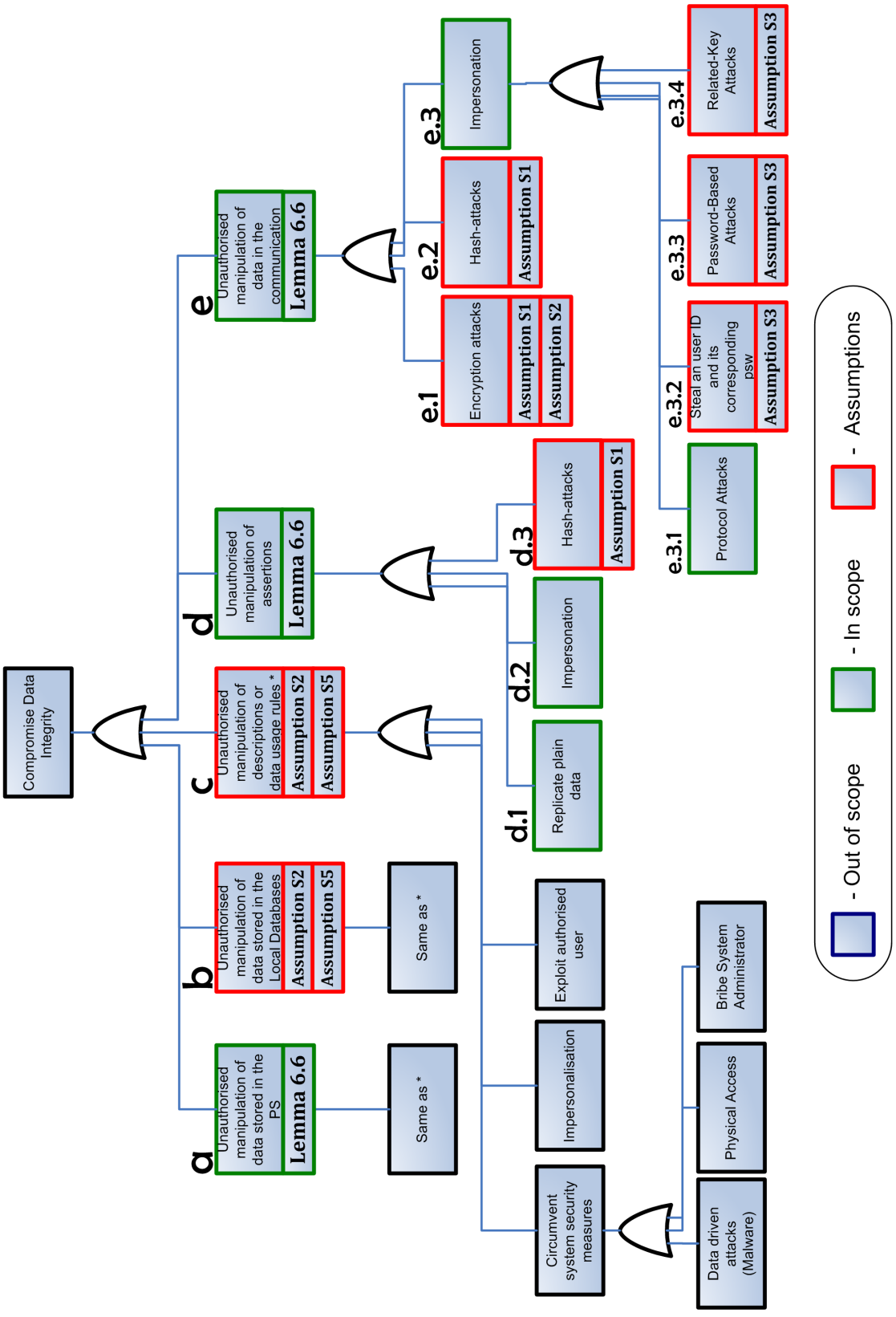


FIGURE 7.2: Information Integrity Attack Tree

In both cases, the strength and secure storage of private keys and certificates is an important issue, which is addressed by Assumption S3. The remaining issues related to the design of a secure protocol and a hash algorithm that can guarantee end-point authentication and mutual authentication were explained and were solved by the formalisation presented in Chapter 6.

Finally, node *c*, is an attack that involves the abuse of multiple identities. For example, turning to our running example of the On-line Sales Scenario, a manager that plays both the role of stock manager and HR manager is able to manually cross-reference orders with job applications, which is in non-compliance with the Auditing Requirements. This problem is the focus of the Identity Management research area [49, 83] that deals with identifying individuals in a system to control the access to resources by placing restrictions on the established identities. One of the challenges within this area is the management of multiple identities that refer to the same individual or system. To avoid these challenges, we assume that in our architecture each user has only one identity. DiMicco and Millen, and Koch and Moslein [49, 83] discuss the possible solutions to this problems and its future developments.

Thus, after analysing this tree, we can conclude that under the given assumptions the aforementioned authentication attacks can be prevented.

7.4.4 Non-Repudiation Attack Tree

The Non-Repudiation Attack Tree presented in Figure 7.4 contains all possible attacks related to the non-repudiation property. In this tree, the attack goal is "Compromise Non-repudiation", which is divided in two subgoals.

Node *a*, the first subgoal, represents the attack "Compromise non-repudiation of messages", which is related to the messages sent between the entities in our architecture. Here, the most important issue is the creation and storage of keys and certificates (Assumption S3), since all non-repudiation techniques require entities to have a secure way of identification.

The same applies to the attack in node *b*, the second subgoal, which is called "Compromise non-repudiation of assertions". Here, we use digital signatures to check the identity of the actors that are recording provenance information. These issues were addressed by Lemma 6.7, which states that non-repudiation holds in the protocols of Data Request, Task Request and Query Request.

Therefore, we can guarantee that under these assumptions, these non-repudiation attacks cannot succeed.

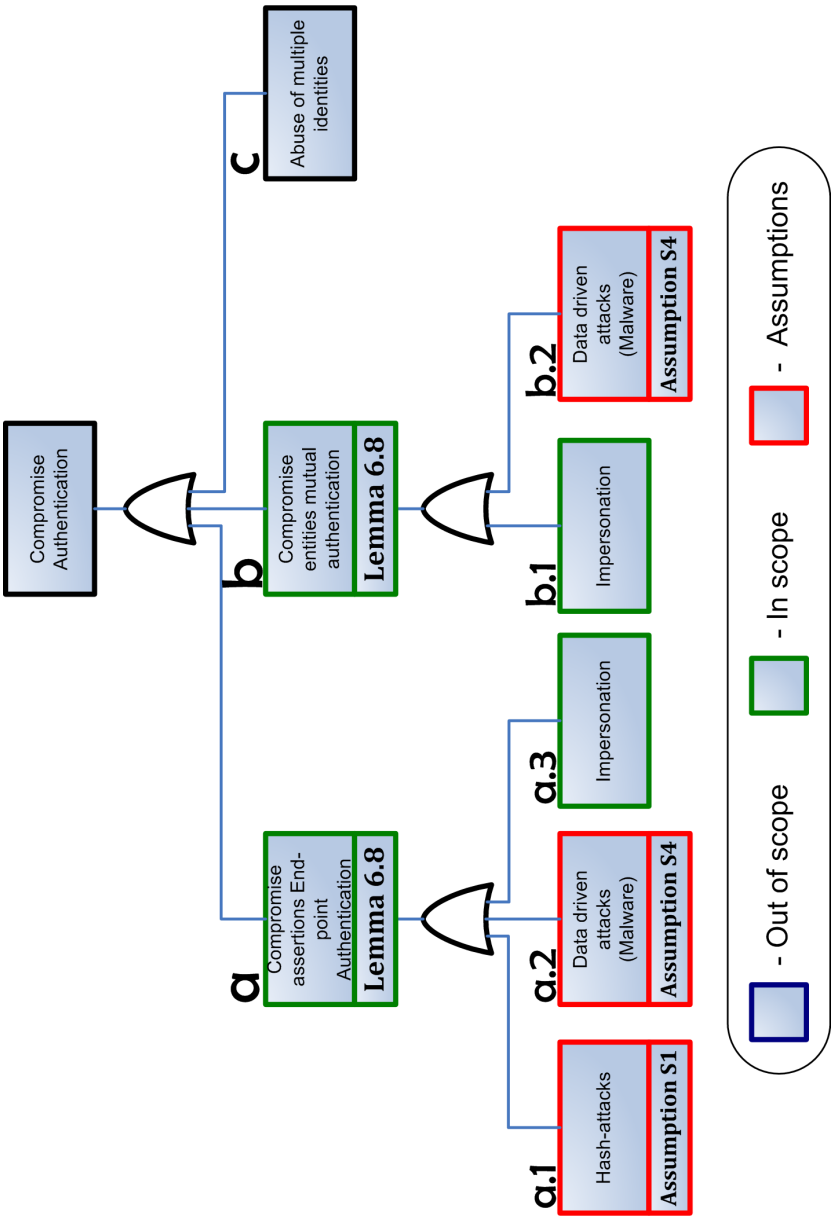


FIGURE 7.3: Authentication Attack Tree

7.4.5 Information Availability Attack Tree

The Information Availability Attack Tree is presented in Figure 7.5. As mentioned in Section 7.3, this attack is outside the scope of this work. However, it is important to mention the measures that should be taken to avoid this attack takes place in our system.

As Figure 7.5 shows, there are three main attacks that could affect the availability of a resource: *a* "Databases Attacks", *b* "Network Attacks" and *c* "Client Attacks".

Node *a* contains three subgoals (children). To prevent attacks *a*.1 "Physical Attacks" and *a*.2 "Data Driven Attacks", all necessary physical and technical measures should be implemented in the system (Assumption S4) [15]. Attack *a*.3 "Crash Databases", has

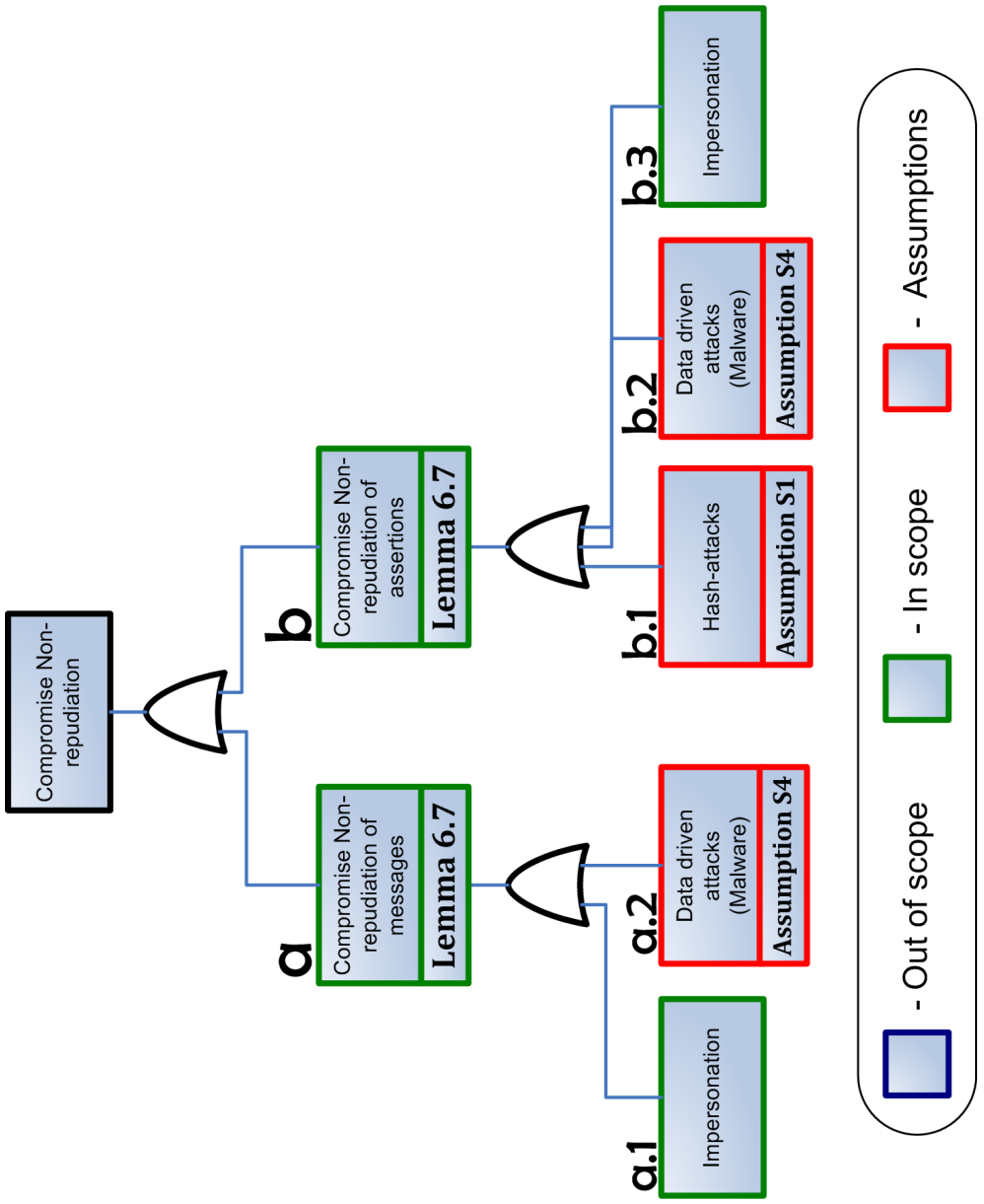


FIGURE 7.4: Non-Repudiation Attack Tree

been the focus of much investigation, which aims to enable databases to recover from a crash without losing data [89]. Thus, to prevent these attacks, we assume that these measures have been implemented in the local databases of our system (Assumption S2). In the case of Provenance Store failures, measures such as the ones presented by Chen and Moreau [38] should be taken into consideration when systems' provenance stores are designed and deployed.

Node *b*, "Network Attacks", can be carried out by implementing *b*.1 "Data Driven Attacks" or *b*.2 "Physical Attacks". To thwart these attacks, different preventive and corrective measures should be followed. Examples of these measures are the correct implementation of firewalls, intrusion detection systems and physical security. A detailed

description of these measures can be found in [76]. We assume that all of them are implemented in our system.

The attack associated with node c, "Client Attacks", can be carried out by achieving one of its four subgoals. The first one, $c.1$, "Hack recording querying audit software", is prevented by Assumptions S5, which states that the necessary measures have been taken to protect the library used by entities to record provenance. The second and third subgoals $c.2$ "Delete Data Descriptions" and $c.3$ "Delete Usage Rules", represent attacks against one of the basic components of our architecture (i.e. provenance information and processing rules, respectively). For this reason, these should be protected and securely stored by implementing proper access control restrictions. Finally, the fourth subgoal, $c.4$ "OS Attacks", raises another issue that is out of the scope of this work. However, it should be taken into account in the maintenance of our system by ensuring that the operating system is regularly updated.

This concludes our description and analysis of the Attack Trees. Using these trees, we have successfully shown that the attacks defined in Section 7.3 can be prevented, either by the contributions proposed in this thesis, or the assumptions we made at the start of section.

7.5 Conclusions

In this chapter, we completed the security analysis of a system designed according to the Secure Provenance-based Auditing Architecture. This analysis was made using Attack Trees, which is formal and methodical way of describing the security attacks systems can be subjected to.

The presented analysis focused on the four security properties that we have addressed in this thesis: confidentiality, integrity, authentication and non-repudiation. We also added availability as an important property. Even though this property was not addressed in this thesis, we present some preventive actions that need to be taken in order to guarantee the proper execution of a system.

First, we thoroughly enumerated the possible attacks on each of the components of the architecture in an attempt to break the aforementioned security properties. Then, we created and analysed five Attack Trees that formalise the attacks identified in the previous step, and discussed which of these attacks are addressed by the contributions of this thesis, and which are prevented by the security assumptions derived from the assumptions in Section 7.3. If neither, we discussed which further measures should be implemented.

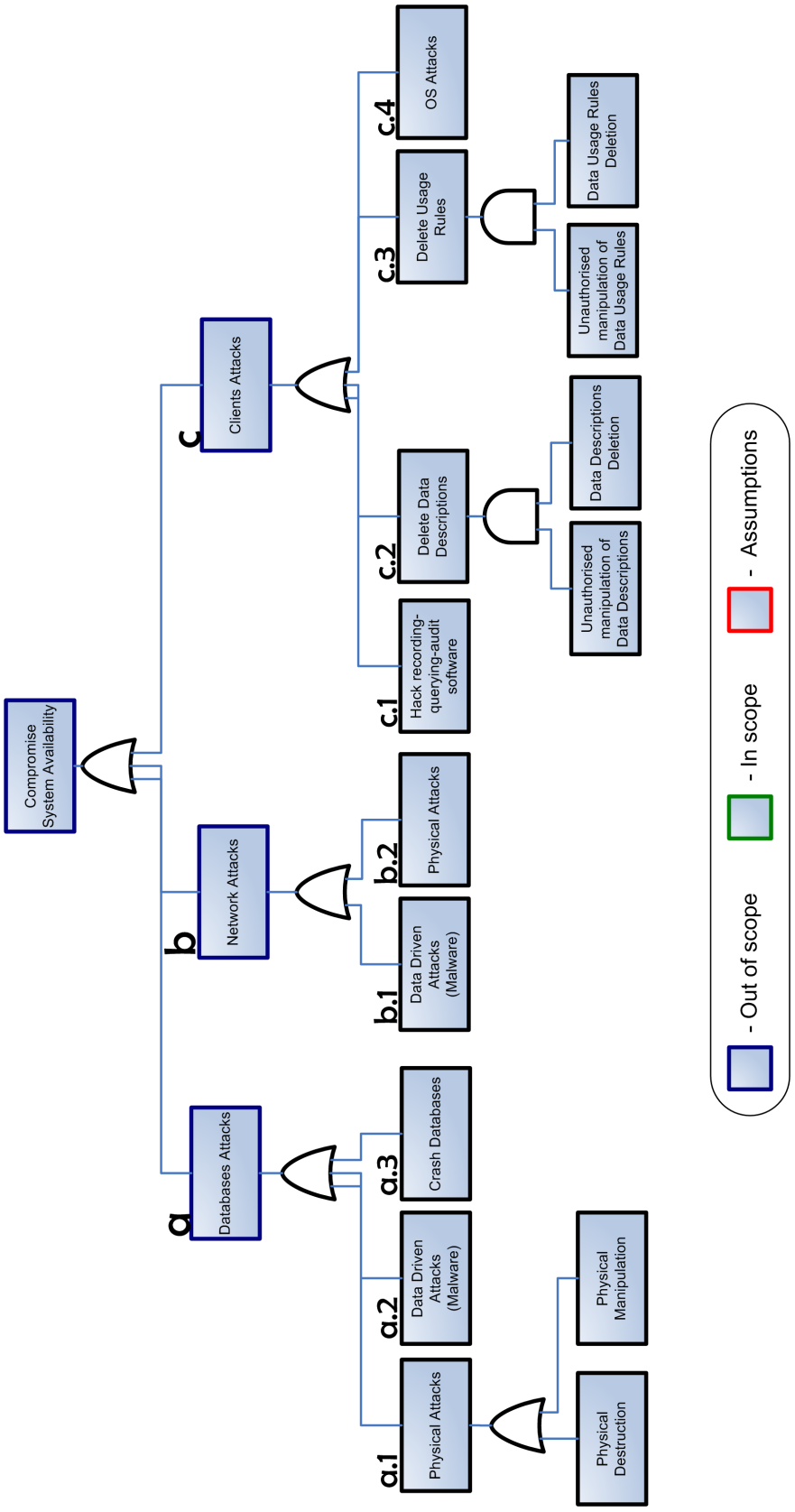


FIGURE 7.5: Information Availability Attack Tree

After this methodical security analysis, we can conclude that under the given assumptions and by implementing the explaining measures, a system that is designed according to our architecture is secure and none of the mentioned attacks can be performed without being detected or solved. Thus, taken together, our contributions ensure that provenance is securely recorded, stored, queried, and analysed.

Chapter 8

Conclusion and Future Work

In recent years, an increasing number of on-line services have started to offer personalised services that require users to disclose personal information. By disclosing personal information, users get access to a wide range of new functionality, such as recommendations or customisation. However, this disclosure also creates a risk that personal information is misused, resulting in harm to its owner.

Within this context, it becomes necessary to be able to verify whether personal information was processed in the correct way, i.e. according to the corresponding data protection legislation. In order to achieve this, information processing should be made *transparent* so it can later be determined whether the use of such information was appropriate. Specifically, if information processing is transparent, auditors can analyse the way in which information was processed and make organisations responsible for any misuse (information accountability). This analysis procedure is called audit.

Against this background, this thesis has proposed a solution to the problem of how to automatically conduct audits, i.e. to verify that personal information has been used according to the Data Protection Act. We have proposed that provenance is an effective means of providing process transparency. When it is used for this purpose, provenance can be seen as electronic evidence of past processing. This provenance can later be automatically analysed (by means of an automatic audit) to decide whether processing was performed in compliance with the data protection legislation.

However, due to the open and distributed nature of current systems, the use of provenance can involve communication between multiple components over potentially unsecured networks. Moreover, even after being securely stored, provenance can still be tampered with by the auditor who conducts the analysis. Consequently, without additional means of protection, there are multiple ways in which provenance might be altered, tampered with, or fall in the hands of untrusted parties. In such cases, the integrity of the obtained audits results cannot be ensured.

Thus, we identified the need for *securing* provenance. To address this problem, we developed the Secure Provenance-based Architecture that protects not only provenance but also the related application data during its collection, storage, communication, query and analysis. By securing the complete life cycle of provenance and application data, this architecture ensures the correctness of the audit results derived from this information.

In addition, using the provenance captured by this secure architecture, we developed the Compliance Framework for automatically verifying the compliance of data processing against a set of Auditing Requirements derived from the DPA. To perform this automatic verification, it needs both provenance and the processing rules in machine-readable format. To do this, the former is represented as a *Provenance Graph* using the Open Provenance Model, which captures the causal dependencies between data and processes. The latter is also represented as a graph, called a Usage Rules Definition graph, which captures the relations between the data classes that can be used, and the allowable operations that can be performed on that data. Thus, processing rules form patterns that provenance must contain to be in compliance.

8.1 Contributions

We now revisit the main contributions made in this thesis in more detail.

8.1.1 Provenance-based Auditing Architecture

First, we derived a set of requirements from a case study of the Data Protection Act. These requirements were divided into two groups, Auditing Requirements and Security Requirements. These groups show that both security and auditing functionalities are necessary to create systems that successfully and securely audit the use of personal information. Consequently, we argued that any system architecture for processing personal information should support both.

To address the Auditing Requirements, the first group of requirements, we developed the Provenance-based Auditing Architecture, which consists of a set of three protocols designed to enable existing systems to capture provenance at execution time, i.e. make them provenance-aware. These three protocols, Data Request, Task Request and Query Request, represent the recording, storing and querying of provenance (the final stage, analysis, is addressed by the Compliance Framework). The architecture and its associated protocols were developed using the PrIMe methodology, which allowed us to identify which provenance should be gathered in order to verify compliance with the Auditing Requirements.

Finally, we demonstrated that, using the provenance collected by the architecture, we can effectively achieve process transparency, i.e. use provenance as electronic evidence

of past processing. As a proof of concept, we presented a manual analysis of the provenance to verify if personal information was processed in accordance with to the Auditing Requirements.

8.1.2 Compliance Framework

Second, we developed the Compliance Framework, a provenance-based auditing framework for automatically auditing the use of personal information by analysing the provenance collected by the Provenance-based Auditing Architecture. Thus, this framework allows for the automatic verification of the Auditing Requirements, and, as such, whether processing of personal information was performed in compliance with the Data Protection Act.

The Compliance Framework consists of three components. The first component, the Processing View, represents provenance as a DAG, showing the relation between data and the processes that act upon it. Put differently, the Processing View captures how personal information was used. The second component, the Usage Rules Definition, encodes the Auditing Requirements using a graph-based rule representation. The third component is a set of algorithms for automatically verifying whether personal information was processed according to the derived requirements. To do so, these algorithms compare the Processing View against the Usage Rules Definition to detect and report any violations of the rules. In summary, the Compliance Framework demonstrates that through the automatic comparison of provenance against processing rules, we can support *information accountability*.

8.1.3 Secure Provenance-based Auditing Architecture

Third, we developed the Secure Provenance-based Auditing Architecture, to guarantee provenance nor application data can be tampered with during its life cycle of recording, storage, querying and analysis. To do this, it secures the main components of the Provenance-based Auditing Architecture: the Data Request, Task Request and Query Request protocols, as well as the provenance graphs. By securing the former, we ensure that provenance is securely created and transmitted and stored in unsecured networks. By securing the latter, we can guarantee provenance cannot be maliciously altered during storage and analysis stage. By doing so, even the auditor is unable to modify the electronic evidence of the processing of personal information.

We formalised the security of the architecture by modelling its protocols using UMLsec. This allowed us to automatically and conclusively verify that the architecture exhibits the essential security properties of confidentiality, integrity, authentication and non-repudiation. By using the Viki model checker, we showed that these properties indeed

hold. As a result, we can guarantee the correctness of audits based on the provenance collected by the Secure Provenance-based Auditing Architecture.

Taken together, these contributions represent a significant advance in the use of provenance for achieving information accountability. However, there are still a few more open questions that need to be addressed.

8.2 Future Work

Specifically, we identify three directions for future work that are aimed to broaden the scope and improve the practical applicability of our research.

As discussed in assumption 7 (See Chapter 3), we assume that there exists a well-defined ontology that defines personal information and the processes that act upon it. This assumption is vital in the creation of the Processing View and the Usage Rules Definition, as well as in the correct execution of the algorithms that are part of the Compliance Framework, because in order to analyse provenance against processing rules, both need to be described using the same concepts and language. Thus, we believe that further research should be conducted in order to create a standard vocabulary that defines processing of personal information.

A second important issue is the protection of privacy in provenance. This issue has already been raised by the research community. In this work, we discuss the importance of anonymity of users as a mean to protect privacy of individuals (see Anonymity Preservation Requirement, Section 3.4). However, this requirement is only verified in the use of data application. Since provenance itself also contains personal information (in the form of a reference or an actual copy), without proper protection, the disclosure of provenance can lead to the disclosure of personal information. Thus, there exists a sensitive trade off between the desire to protect privacy of individuals and to achieve information transparency. Consequently, further investigation should be carry out to develop privacy protecting provenance frameworks.

Finally, one of the main motivations of our work is the Data Protection Act. As discussed in Chapter 3, the Data Protection Act states the possibility of auditing organisations to verify that their data processing is in compliance. We believe that the work presented here can be used as a mean to automatically conduct these audits. However, in order to make the results of these automatic audits admissible as legal evidence, further research is needed into the legal aspects of this electronic evidence, and the necessary protocols that should be followed.

In the next sections, these directions of future work are discussed in further detail.

8.2.1 Defining a Standard Vocabulary

As we explained in Chapter 5, where we discussed the Compliance Framework, in order to successfully verify that personal information was processed according to a set of processing rules, it is necessary to have a standard vocabulary. This vocabulary should be used in the provenance representation (Processing View) and in the processing rules (Usage Rules Definition). This vocabulary should allow us to correctly describe the main components of the processing of personal information: purposes, tasks, collect data and used data (classes and types). Even though the ICO's Register attempts to define these components, we demonstrated that for achieving processing transparency it is incomplete. Moreover, it does not define a proper language that can be reused in a computational system. For example, an organisation can define "on-line sales" as purpose to indicate that they will collect information from users to sale products using an on-line application. Another organisation can define "e-sales" as purpose to indicate the same activity.

By developing a standard vocabulary for processing personal information, the set of algorithms for automatic analysis of personal data processing can be applied anywhere personal information is processed, instead of having to develop domain-specific instantiations of the algorithms for each separate organisation. We believe that it is important that this vocabulary is implemented using Semantic Web Technologies, i.e. as a standard ontology that can be (re)used by any organisation that is managing personal information. Thus, the algorithms presented in Chapter 5 can be implemented using RDF. In that way, the Compliance Framework can be widely used to verify the compliance with the Data Protection Act.

8.2.2 Privacy protecting provenance

As we argued in the introduction, processing transparency is a key issue in ensuring information accountability. By supporting processing transparency, auditors are able to analyse how data was processed and which data was used. However, processing transparency should also protect privacy of individuals. This way, if provenance contains sensitive information about a user, the identity of this user should be protected. In that sense, there exists a trade-off between process transparency and privacy protection. How much personal information should be disclosed on provenance to ensure process transparency without breaking privacy of the involved individuals?

Even though, there are other research areas (such as databases) in which this issue has been investigated, little research has focussed on this issue in the provenance community [45] Therefore, we believe that it is necessary to develop new techniques that find a balance between process transparency and privacy protection, i.e. create a privacy-aware audit approach.

8.2.3 Provenance as legal evidence

Our Secure Provenance-based Auditing Architecture creates unforgeable evidence about the processing of information. This information is later used to decide whether DPA principles where correctly followed during this processing. If a principle was violated, the Compliance Framework can detect who was responsible, and how and where this violation occurred. Hence, from a legal point of view, the provenance obtained and analysed by our architecture can be seen as "digital evidence". Digital evidence or electronic evidence is "any probative information stored or transmitted in digital form that a party to a court case may use at trial" [36]. For that reason, we believe we have provided all the necessary means to ensure that the evidence created by the Secure Provenance-based Auditing Architecture can be used in a court as evidence of misuse of information. Consequently, it is important to explore the more practical issues involved in achieving this, such as the protocols involved in handling evidence, and the necessary properties that need to hold to make electronic evidence admissible. To achieve this, computer scientists should work in close collaboration with experts in the field of (Internet) law.

Addressing these open questions combined with the contributions of this thesis is a clear step forward in mitigating the risks of disclosing personal information, making organisations accountable for the way in which they manage this information, and increasing the trustworthiness of on-line services.

Appendix A

ArgoUML UMLsec Diagrams

In Chapter 6, we presented the set of UMLsec sequence diagrams that were used to verify that the Provenance-based Auditing Architecture exhibits security properties of authentication, confidentiality, integrity and non-repudiation. For reasons of clarity and consistency, these sequence diagrams are a graphic representation of the original diagrams that were verified with the Viki tool. In the interest of completeness, however, the original UMLsec diagrams are presented in this appendix.

The original diagrams were created using ArgoUML version 0.32.3, a visual editor developed to create UML diagrams. This editor generates UML models that follow Viki requirements and, therefore, can be directly used as input to Viki.

Unfortunately, ArgoUML does not visualise the contents of the messages or tag values within the UML diagrams. Hence, in this Chapter we not only present the original UMLsec diagram of each protocol of the *Secure Provenance-based Auditing Architecture* but also the Graphical User Interface elements showing the contents of each message and the contents of the adversary entity, which includes three set of tag values called secret, initial knowledge and guard.

This appendix is organised as follows. First, in Section A.1 we present the UMLsec diagram of the OTLS protocol including the message contents and the tag values. Similarly, we present the diagram of the Data Request protocol in Section A.2 and the diagram of the Task Request protocol in Section A.3. Finally, the results of the verification process are presented in Section A.4.

A.1 OTLS UMLsec Sequence Diagram

This section presents the ArgoUML UMLsec sequence diagram of the OTLS protocol created in ArgoUML (Figure A.1) along with the content of the messages of this protocol (Figure A.2) and the tag values contained in the attacker entity (Figure A.3) [1]

These diagrams are a different representation of the sequence diagram presented in Figure 6.1. In Figure A.1 the entities S, C, PS and A are instances of the `Server`, `Client`, `ProvenanceStore` and `Adversary` actors, respectively. The messages presented in Figure A.2 are equivalent to the messages in Figure 6.1. However, these messages use a different notation, which is equivalent to the notation presented in Equations (2.2) to (2.8). This notation is used by Viki and presented in Table A.1 [78].

Operation	Notation
A concatenated with B	$A :: B$
A equal to B	$A = B$
Logical conjunction	&
Head of A	$head(A)$
Tail of A	$tail(A)$
Private key k	$inv(k)$
A symmetrically encrypted using the key k	$symenc(A, k)$
A symmetrically decrypted using the key k	$dec(A, k)$
A asymmetrically encrypted using the key k	$enc(A, k)$
A symmetrically decrypted using the key k	$dec(A, k)$
A signed using the key k	$sign(A, k)$
A extracted using the key k	$ext(A, k)$
Hash of A	$hash(A)$
Key generation function	$kgen()$
Extract the i^{th} element from message M	M_i

Table A.1: Viki Notation

The last operation M_i in Table A.1 provides a way to access individual elements of each message. For example, if we have a message M containing three elements $M(e_1, e_2, e_3)$, M_1 retrieves the first element of M, which in this case is e_1. Similarly, M_2 and M_3 retrieve e_2 and e_3, respectively. Note that, in Viki variables do not need to be explicitly declared, instead they are implicitly declared on first use.

Finally, the tag values presented in Figure A.3 are the same as the ones presented as blue ovals in Figure 6.1 along with the ones presented in Table 6.4.

[1] The original OTLS UMLsec Diagram can be found in `http://tinyurl.com/6uoend6` This diagram is contained in a zargo file that can be opened in the visual editor ArgoUML.

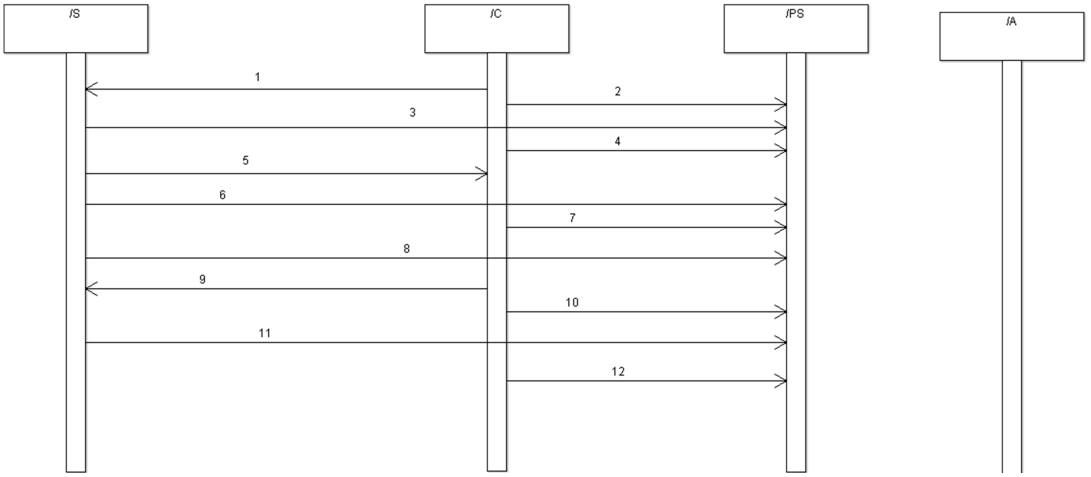


FIGURE A.1: OTLS UMLSec Sequence Diagram

a⇒ lpa1(id1,sender,sign(hash(id1::sender),inv(k_c)))
a⇒ lpa2(id1,receiver,sign(hash(id1::receiver),inv(k_s)))
a⇒ lpa3(id2,sender,sign(hash(id2::sender),inv(k_s)))
a⇒ lpa4(id2,receiver,sign(hash(id2::receiver),inv(k_c)))
a⇒ lpa5(id3,sender,sign(hash(id3::sender),inv(k_c)))
a⇒ lpa6(id3,receiver,sign(hash(id3::receiver),inv(k_s)))
a⇒ SimpleRpa(symenc(id1::n1,k_cps),sign(hash(id1::n1::hash(id1::n1)),inv(k_c)),hash(id1::n1))
a⇒ Rpa1(symenc(id1::n1::inResponseTo::id2 ,k_sps),
sign(hash(id1::n1::inResponseTo::id2 ::hash(id1::n1::inResponseTo::id2:: SimpleRpa_3)),inv(k_s)),hash(id1::n1::inResponseTo::id2:: SimpleRpa_3))
a⇒ Rpa2(symenc(id2::n2::inResponseTo::id3,k_cps),
sign(hash(id2::n2::inResponseTo::id3::hash(id2::n2::inResponseTo::id3:: Rpa1_3)),inv(k_c)),hash(id2::n2::inResponseTo::id3:: Rpa1_3))
a⇒ ClientHello(id1,n1,k_c,sign(c::k_c, inv(k_c)), hash(id1::c::k_c))
a⇒ ServerHello(id2,enc(sign(kgen(ClientHello_3)::n2::ClientHello_3,inv(k_s)),ClientHello_3),sign(s::k_s,inv(k_ca)),hash(id2::s::k_s))
a⇒ KeyExchange(id3,symenc(key,k_s),hash(id1::c::k_c))

FIGURE A.2: OTLS UMLSec Sequence Diagram Message Content

Target: ClassifierRole (A) TD

Tag	Value
initial knowledge	k_a
initial knowledge	inv(k_a)
secret	key
notation	new
guard_notation	new
guard_5	snd(ext(ClientHello_4,ClientHello_3))=ClientHello_3
guard_9	(fst(ext(ServerHello_3,k_ca))=s)&(snd(ext(dec(ServerHello_2,inv(k_c)),snd(ext(ServerHello_3,k_ca))))= n)&(thd(ext(dec(ServerHello_2,inv(k_c)),snd(ext(ServerHello_3,k_ca))))=k_c)
initial knowledge	k_cps
initial knowledge	k_c
initial knowledge	k_s
initial knowledge	k_sps
guard_2	ext(lpa1_3,k_c)=hash(lpa1_1::lpa1_2)
guard_3	ext(lpa2_3,k_s)=hash(lpa2_1::lpa2_2)
guard_6	ext(lpa3_3,k_s)=hash(lpa3_1::lpa3_2)
guard_7	ext(lpa4_3,k_c)=hash(lpa4_1::lpa4_2)
guard_10	ext(lpa5_3,k_c)=hash(lpa5_1::lpa5_2)
guard_11	ext(lpa6_3,k_s)=hash(lpa6_1::lpa6_2)
guard_4	(hash(dec(SimpleRpa_1,k_cps)::SimpleRpa_3)=ext(SimpleRpa_2,k_c))&(hash(dec(SimpleRpa_1,k_cps))=SimpleRpa_3)
guard_8	(hash(dec(Rpa1_1,k_cps)::Rpa1_3)=ext(Rpa1_2,k_c))&(hash(dec(Rpa1_1,k_cps))=Rpa1_3)
guard_12	(hash(dec(Rpa2_1,k_cps)::Rpa2_3)=ext(Rpa2_2,k_c))&(hash(dec(Rpa2_1,k_cps))=Rpa2_3)
initial knowledge	

FIGURE A.3: OTLS UMLSec Sequence Diagram Adversary Content

A.2 Data Request UMLsec Sequence Diagram

This section presents the ArgoUML UMLsec sequence diagram of the Data Request protocol related to the sequence diagram presented in Figure 6.2. In Figure A.4 the entities

DS, *DC*, *PS* and *A* are instances of the `DataSubject`, `DataController`, `ProvenanceStore` and `Adversary` actors, respectively [2].

As in Section A.1, the messages presented in Figure A.5 are the same ones presented in Figure 6.2 but use the notation presented in Table A.1. The tag values presented in Figure A.6 are also the same as the ones presented as blue ovals in Figure 6.2 and the ones presented in Table 6.4.

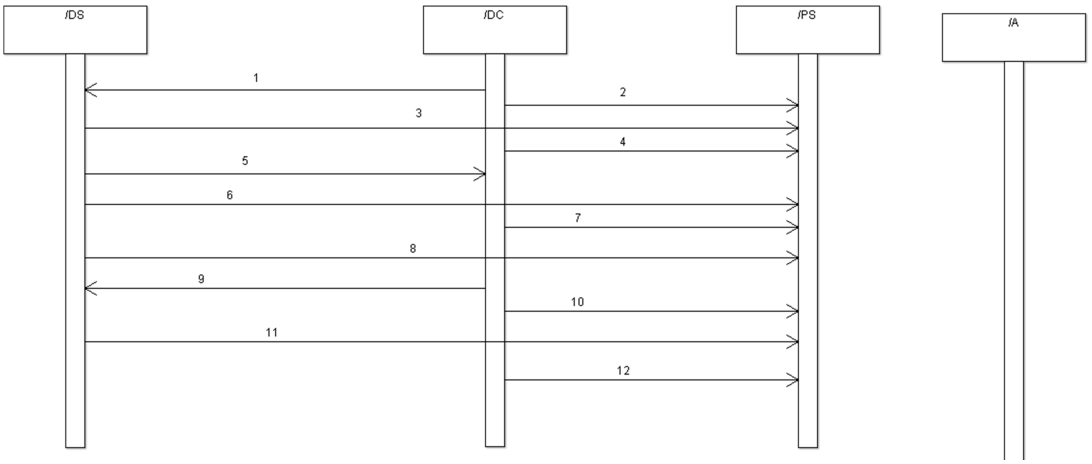


FIGURE A.4: Data Request UMLSec Sequence Diagram

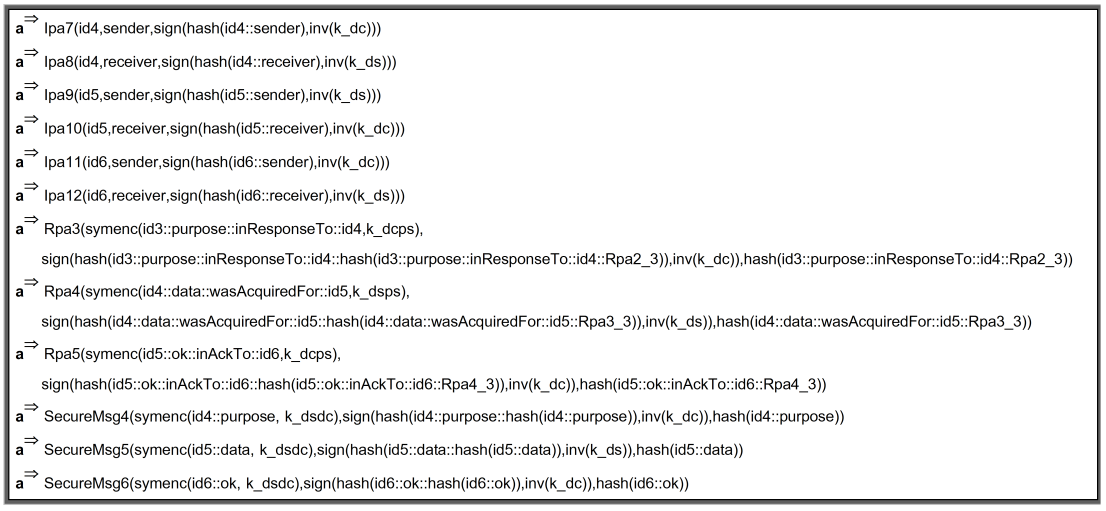


FIGURE A.5: Data Request UMLSec Sequence Diagram Message Content

A.3 Task Request UMLsec Sequence Diagram

This section presents the ArgoUML UMLsec sequence diagram of the Task Request protocol related to the sequence diagram presented in Figure 6.3. Therefore, in Figure A.7

[2]The original Data Request UMLsec Diagram can be found in `http://tinyurl.com/7aamyoa`. Again, this diagram is contained in a zargo file that can be opened in the visual editor ArgoUML

Target: ClassifierRole (A) TD

Tag	Value
initial knowledge	k_dc
initial knowledge	k_ds
secret	data
notation	new
guard_notation	new
initial knowledge	k_ps
secret	purpose
guard_1	ext(SecureMsg4_2,k_dc)=hash(dec(SecureMsg4_1,k_dsdc))
guard_2	ext(lpa7_3,k_dc)=hash(lpa7_l::pa7_2)
guard_3	ext(lpa8_3,k_ds)=hash(lpa8_1::lpa8_2)
guard_4	(hash(dec(Rpa3_1,k_dcps)::Rpa3_3)=ext(Rpa3_2,k_dc))&(hash(dec(Rpa3_1,k_dcps))=Rpa3_3)
guard_5	ext(SecureMsg5_2,k_ds)=hash(dec(SecureMsg5_1,k_dsdc))
guard_6	ext(lpa9_3,k_ds)=hash(lpa9_1::lpa9_2)
guard_7	ext(lpa10_3,k_dc)=hash(lpa10_1::lpa10_2)
guard_8	(hash(dec(Rpa4_1,k_dsps)::Rpa4_3)=ext(Rpa4_2,k_ds))&(hash(dec(Rpa4_1,k_dsps))=Rpa4_3)
guard_9	ext(SecureMsg6_2,k_dc)=dec(SecureMsg6_1,k_dsdc)
guard_10	ext(lpa11_3,k_dc)=hash(lpa11_1::lpa11_2)
guard_11	ext(lpa12_3,k_ds)=hash(lpa12_1::lpa12_2)
guard_12	(hash(dec(Rpa5_1,k_dcps)::Rpa5_3)=ext(Rpa5_2,k_dc))&(hash(dec(Rpa5_1,k_dcps))=Rpa5_3)
secret	ok

FIGURE A.6: Data Request UMLSec Sequence Diagram Adversary Content

the entities DP, DC, PS and A are instances of the `DataProcessor`, `DataController`, `ProvenanceStore` and `Adversary` actors, respectively [3].

As in previous diagrams, the messages presented in Figure A.8 and the tag values presented in Figure A.6 are the same ones presented in Figure 6.2.

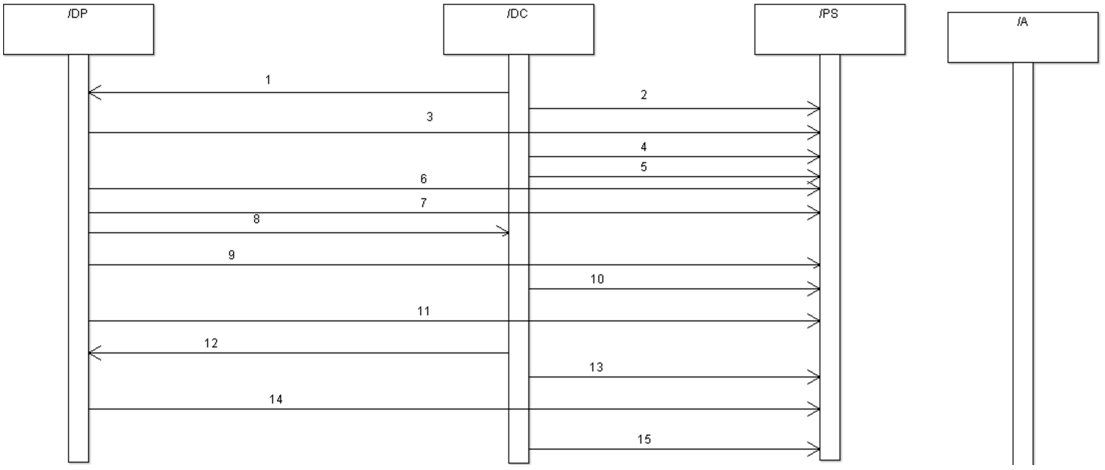


FIGURE A.7: Task Request UMLSec Sequence Diagram

A.4 Viki Results

The UMLSec diagrams presented in the previous sections are created using the notation described in Table A.1 and act as input to the Viki model checker. In this section, we present the results obtained after verifying these diagrams with Viki. As the results for each of the above diagrams are the same, we only show a single figure representing the output of the verification process of all the diagrams.

[3]The original Task Request UMLsec Diagram can be found in `http://tinyurl.com/74hzubt`. Again, this diagram is contained in a zargo file that can be opened in the visual editor ArgoUML

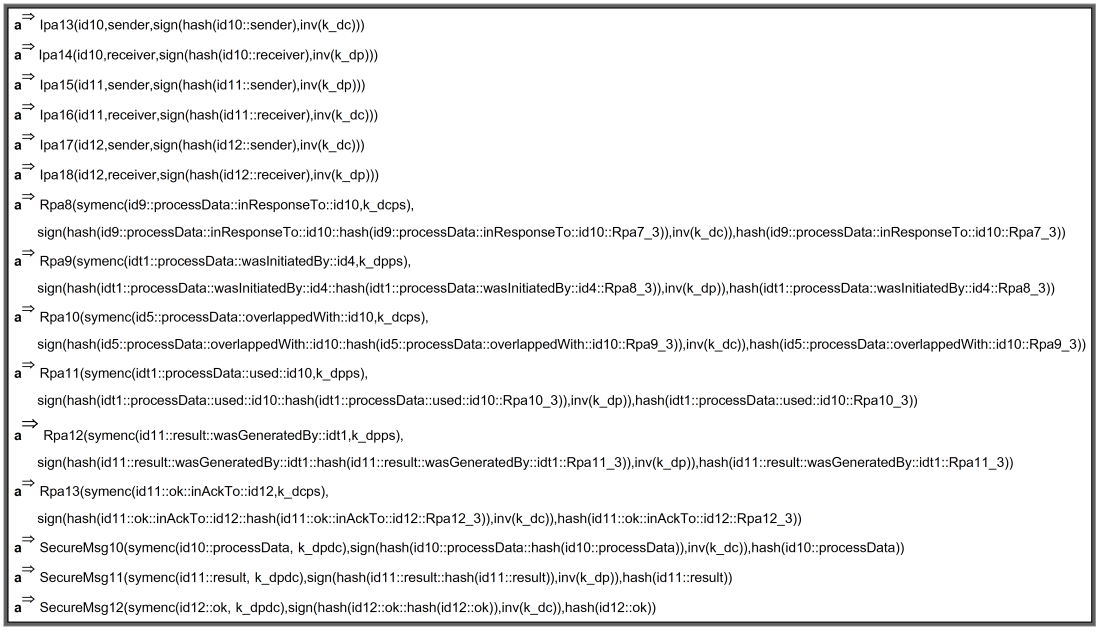
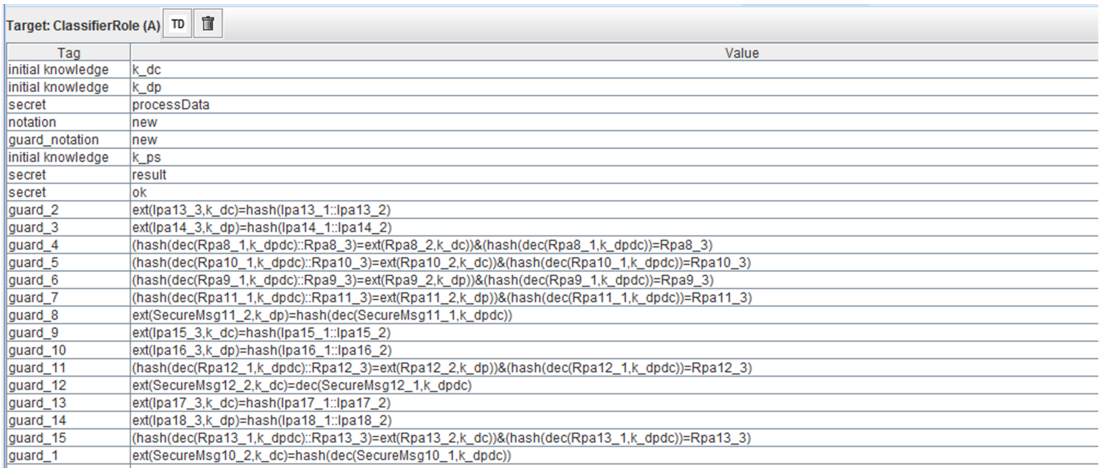


FIGURE A.8: Task Request UMLSec Sequence Diagram Message Content



Target: ClassifierRole (A) TD

Tag	Value
initial knowledge	k_dc
initial knowledge	k_dp
secret	processData
notation	new
guard_notation	new
initial knowledge	k_ps
secret	result
secret	ok
guard_2	ext(lpa13_3,k_dc)=hash(lpa13_1::lpa13_2)
guard_3	ext(lpa14_3,k_dp)=hash(lpa14_1::lpa14_2)
guard_4	(hash(dec(Rpa8_1,k_dpdc)::Rpa8_3)=ext(Rpa8_2,k_dc))&(hash(dec(Rpa8_1,k_dpdc))=Rpa8_3)
guard_5	(hash(dec(Rpa10_1,k_dpdc)::Rpa10_3)=ext(Rpa10_2,k_dc))&(hash(dec(Rpa10_1,k_dpdc))=Rpa10_3)
guard_6	(hash(dec(Rpa9_1,k_dpdc)::Rpa9_3)=ext(Rpa9_2,k_dp))&(hash(dec(Rpa9_1,k_dpdc))=Rpa9_3)
guard_7	(hash(dec(Rpa11_1,k_dpdc)::Rpa11_3)=ext(Rpa11_2,k_dp))&(hash(dec(Rpa11_1,k_dpdc))=Rpa11_3)
guard_8	ext(SecureMsg11_2,k_dp)=hash(dec(SecureMsg11_1,k_dpdc))
guard_9	ext(lpa15_3,k_dc)=hash(lpa15_1::lpa15_2)
guard_10	ext(lpa16_3,k_dp)=hash(lpa16_1::lpa16_2)
guard_11	(hash(dec(Rpa12_1,k_dpdc)::Rpa12_3)=ext(Rpa12_2,k_dp))&(hash(dec(Rpa12_1,k_dpdc))=Rpa12_3)
guard_12	ext(SecureMsg12_2,k_dc)=dec(SecureMsg12_1,k_dpdc)
guard_13	ext(lpa17_3,k_dc)=hash(lpa17_1::lpa17_2)
guard_14	ext(lpa18_3,k_dp)=hash(lpa18_1::lpa18_2)
guard_15	(hash(dec(Rpa13_1,k_dpdc)::Rpa13_3)=ext(Rpa13_2,k_dc))&(hash(dec(Rpa13_1,k_dpdc))=Rpa13_3)
guard_1	ext(SecureMsg10_2,k_dc)=hash(dec(SecureMsg10_1,k_dpdc))

FIGURE A.9: Task Request UMLSec Sequence Diagram Adversary Content

As Figure A.10 shows, Viki outputs the result "Completion Found", meaning that during the execution of the corresponding sequence diagram, the attacker cannot obtain any of the parameters designated as "secret" in the tag values.

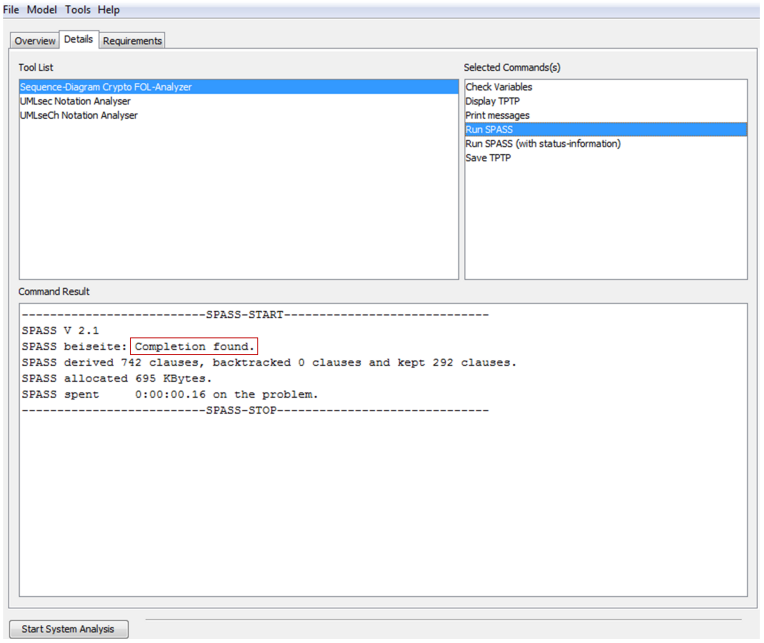


FIGURE A.10: Viki Result of the UMLSec Sequence Diagram

Bibliography

[1] Trusted Computer System Evaluation Criteria, 1985.

[2] Trusted Computer System Evaluation Criteria (5200.28-STD), 1985.

[3] ISO/IEC-7498-2: Information Processing Systems – Open Systems Interconnection – Basic Reference Model – Part 2: Security Architecture, 1989.

[4] ISO/IEC-9796: Information Technology – Security Techniques –Digital Signature Scheme Giving Message Recovery, 1991.

[5] Directive 95/46/EC of the European Parliament, October 1995.

[6] Health Insurance Portability and Accountability Act. `http://www.hhs.gov/ocr/hipaa`, 1996.

[7] Data Protection Act. `http://www.opsi.gov.uk/acts/acts1998/19980029.htm`, 1998.

[8] Safe Harbor. `http://www.export.gov/safeharbor/index.html`, 2000.

[9] OMG Unified Modeling Language Specification v1.5, March 2003.

[10] Public Sector Data Sharing: Guidance on the Law, November 2003.

[11] ISO/IEC-17799: Information Technology – Security Techniques – Code of Practice for Information Security Management, 2005.

[12] Protection of Private Data – First Report of Session 2007–08. `http://www.publications.parliament.uk/pa/cm200708/cmselect/cmjust/154/154.pdf`, 2007.

[13] Data Handling Review. `http://www.cabinetoffice.gov.uk/resource-library/data-handling-procedures-government`, November 2008.

[14] ITU-T Recommendation X.509: Information Technology – Open Systems Interconnection – The Directory: Public-key and Attribute Certificate Frameworks, 2008.

[15] Strategies to Protect Against Distributed Denial of Service (DDoS) Attacks. `http://www.cisco.com/en/US/tech/tk59/technologies_white_paper09186a0080174a5b.shtml`, April 2008.

[16] FIPS-186-3: Digital Signature Standard (DSS), 2009.

[17] Carlisle Adams and Steve Lloyd. *Understanding PKI: concepts, standards, and deployment considerations*. Addison-Wesley, 2003.

[18] Debra Anderson, Thane Frivold, Ann Tamaru, and Alfonso Valdes. *Next Generation Intrusion Detection Expert System (NIDES) Software Users Manual*. California, USA, 1994.

[19] George Apostolopoulos, Vinod Peris, and Debanjan Saha. Transport Layer Security: How much does it really cost? In *Conference on Computer Communications (IEEE InfoCom)*, pages 717–725, New York, 1999. IEEE Computer Society.

[20] Ahmed Awad, Matthias Weidlich, and Mathias Weske. Specification, Verification and Explanation of Violation for Data Aware Compliance Rules. In *Lecture Notes In Computer Science*, volume 5900, pages 500–515, Stockholm, 2009. Springer-Verlag.

[21] Stefan Axelsson. Intrusion Detection Systems: A Survey and Taxonomy. Technical Report 99-15, Department of Computer Engineering, Chalmers University of Technology, Goteborg, Sweden, March 2000.

[22] D. Banning, G. Ellingwood, C. Franklin, C. Muckenhirn, and D. Price. Auditing of distributed systems. In *14th National Computer Security Conference*, pages 59–68, October 1991.

[23] R S Barga and L A Digiampietri. Automatic capture and efficient storage of e-Science experiment provenance. *Concurrency and Computation: Practice and Experience*, 20:419–429, 2008.

[24] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In *Lecture Notes in Computer Science*, volume 1109 of *16th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO '96)*, pages 1–15. Springer-Verlag, 1996.

[25] Matt Bishop. A Standard Audit Trail Format. In *18th National Information Systems Security Conference*, pages 136–145. IEEE, 1995.

[26] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked Data - The Story So Far. *To appear on International Journal on Semantic Web and Information Systems*, (Special Issue on Linked Data), 2010.

[27] S. Blake-Wilson, M. Nystrom, D. Hopwood, J. Mikkelsen, and T. Wright. RFC 3546: Transport Layer Security (TLS) Extensions, June 2003.

[28] Manuel Blum and Silvio Micali. How to Generate Cryptographically Strong Sequences of Pseudo-random Bits. *SIAM Journal on Computing*, 13(4):850–864, November 1984.

[29] S Bowers, T M McPhillips, and B Ludäscher. Provenance in collection-oriented scientific workflows. *Concurrency and Computation: Practice and Experience*, 20:519–529, 2008.

[30] Gilles Brassard. *Modern cryptology : A Tutorial*. Number 325 in LNCS. Springer-Verlag, New York, 1988.

[31] Uri Braun, Avraham Shinnar, and Margo Seltzer. Securing Provenance. In USENIX Association, editor, *3rd USENIX Workshop on Hot Topics in Security*, USENIX HotSec, pages 1–5, Berkeley, CA, USA, July 2008.

[32] Peter Buneman, Adriane Chapman, and James Cheney. Provenance management in curated databases. In *SIGMOD Conference*, pages 539–550, 2006.

[33] Peter Buneman, James Cheney, and Stijn Vansummeren. On the expressiveness of implicit provenance in query and update languages. *ACM Transactions on Database Systems (TODS)*, 33(4):1–47, 2008.

[34] Peter Buneman, Sanjeev Khanna, and Wang-Chiew Tan. Why and Where: A Characterization of Data Provenance. In *Lecture Notes in Computer Science*, volume 1973, pages 316–325. Springer-Verlag, 2001.

[35] S P Callahan, J Freire, E Santos, C E Scheidegger, Cláudio T Silva, and H T Vo. VisTrails: visualization meets data management. In *SIGMOD Conference*, pages 745–747, 2006.

[36] Eoghan Casey. *Digital Evidence and Computer Crime: Forensic Science, Computers and the Internet*. Elsevier Academic Press, 2nd. edition, 2004.

[37] Bee-Chung Chen, Daniel Kifer, Kristen LeFevre, and Ashwin Machanavajjhala. Privacy-Preserving Data Publishing. *Foundations and Trends in Databases*, 2(12):1–167, 2009.

[38] Zheng Chen and Luc Moreau. Provenance and Annotation of Data and Processes. In *Lecture Notes in Computer Science*, volume 5272, pages 92–105, Berlin, Heidelberg, 2008. Springer-Verlag.

[39] Tzi-cher Chiueh and Shibiao Lin. A Survey on Solutions to Distributed Denial of Service Attacks. RPE report TR-201, Stony Brook University, Stony Brook, NY, September 2006.

[40] Alison Chorley, Pete Edwards, Alun Preece, and John Farrington. Tools for tracing evidence in social science. In *Proceedings of the Third International Conference on eSocial Science*, pages 1–10, October 2007.

[41] Gary G. Christoph, Kathleen A. Jackson, Michael C. Neuman, Christine L. B. Siciliano, Dennis D. Simmonds, Cathy A. Stallings, and Joseph L. Thompson. UNICORN: Misuse Detection for UNICOS. In *Proceedings of the 1995 ACM/IEEE conference on Supercomputing*, pages 56–66, 1995.

[42] V. Ciriani, S. Capitani di Vimercati, S. Foresti, and P. Samarati. k-anonymity. In Ting Yu and Sushil Jajodia, editors, *Secure Data Management in Decentralized Systems*, volume 33 of *Advances in Information Security*, pages 323–353. Springer US, 2007. 10.1007/978-0-387-27696-0-10.

[43] Cristian Coarfa, Peter Druschel, and Dan S. Wallach. Performance analysis of TLS Web servers. *ACM Transactions on Computer Systems*, 24(1):39–69, February 2006.

[44] Francisco Curbera, Yurdaer Doganata, Axel Martens, Nirmal K. Mukhi, and Aleksander Slominski. Business Provenance — A Technology to Increase Traceability of End-to-End Operations. In *OTM 2008 Confederated International Conferences*, pages 100–119, Monterrey, Mexico, 2008. Springer-Verlag.

[45] Susan Davidson. On Provenance and Privacy. In Deborah McGuinness, James Michaelis, and Luc Moreau, editors, *Provenance and Annotation of Data and Processes (IPAW 2010)*, volume 6378 of *Lecture Notes in Computer Science*, pages 3–10, Troy, NY, 2010. Springer Berlin / Heidelberg.

[46] Donald Watts Davies and W.L. Price. *Security for Computer Networks*. JohnWiley & Sons, New York, 2nd edition, 1989.

[47] Nicholas Del˜Rio, Paulo Pinheiro˜da Silva, and Raed Aldouri. Identifying and Explaining Map Quality Through Provenance: A User Study. In *Proceedings of IJCAI 2009 Workshop on Explanation-Aware Computing (ExACT 2009)*, pages 110–117, Pasadena, CA, USA, July 2009.

[48] W. Diffie. The first ten years of public key cryptology. In *Contemporary Cryptology: The Science of Information Integrity*, pages 135–175, 1992.

[49] Joan Morris DiMicco and David R. Millen. Identity management: multiple presentations of self in facebook. *Conference on Supporting Group Work*, 2007.

[50] D. Dolev and A. Yao. On the security of public key protocols. *Information Theory, IEEE Transactions on*, 29(2):198–208, March 1983.

[51] Thai Duong and Juliano Rizzo. Here Come The $\oplus$ Ninjas. In *Ekoparty Security Conference*, May 2011.

[52] Cynthia Dwork. Differential Privacy. In *33rd International Colloquium on Automata, Languages and Programming*, pages 1–12, Venice, Italy, 2006. Springer Verlag.

[53] Juliana Freire, David Koop, Emanuele Santos, and Cláudio T Silva. Provenance for Computational Tasks: A Survey. *Computing in Science and Engineering*, 10(3):11–21, 2008.

[54] James Frew and Peter Slaughter. ES3: A Demonstration of Transparent Provenance for Scientific Computation . In Juliana Freire, David Koop, and Luc Moreau, editors, *Provenance and Annotation of Data and Processes*, volume 5272 of *Lecture Notes in Computer Science*, pages 200–207, Berlin, Heidelberg, 2008. Springer-Verlag.

[55] Yolanda Gil and Christian Fritz. Reasoning about the Appropriate Use of Private Data through Computational Workflows. In *AAAI Spring Symposium on Privacy Management 2010*, pages 23–25, 2010.

[56] Boris Glavic and Gustavo Alonso. Perm: Processing Provenance and Data on the Same Data Model through Query Rewriting. In *2009 IEEE 25th International Conference on Data Engineering (ICDE'09)*, pages 174–185. IEEE Computer Society, March 2009.

[57] Boris Glavic and Klaus R. Dittrich. Data Provenance: A Categorization of Existing Approaches. In *Database Systems for Business, Technology and Web (BTW)*, pages 227–241, 2007.

[58] UK Government. UK's Government Public Data. `http://www.data.gov.uk/`.

[59] USA Government. USA's Government Public Data. `http://www.data.gov/`.

[60] M Greenwood, C Goble, R Stevens, J Zhao, and Others. Provenance of e-Science Experiments - experience from Bioinformatics. In Simon Cox, editor, *OST e-Science Second All Hands Meeting 2003 (AHM'03)*, Nottingham, UK, September 2003.

[61] Paul Groth. *On the Record: Provenance in Large Scale, Open, Distributed Systems.* PhD thesis, University of Southampton, ECS, 2005.

[62] Paul Groth, Simon Miles, and Luc Moreau. PReServ 0.3.1, P-assertion Recording for Services. `http://twiki.gridprovenance.org/bin/view/PASOA/SoftWare`, August 2007.

[63] Paul Groth, Simon Miles, and Luc Moreau. A Model of Process Documentation to Determine Provenance in Mash-ups. *Transactions on Internet Technology (TOIT)*, 9:1–31, 2009.

[64] Paul Groth and Luc Moreau. Recording Process Documentation for Provenance. *IEEE Transactions on Parallel and Distributed Systems*, 20(9):1246–1259, September 2009.

[65] Barbara Guttman. *An Introduction to Computer Security: The NIST Handbook*. NIST, NIST Special Publication 800-12 edition, 1995.

[66] Chris Hanson, Tim Berners-lee, Lalana Kagal, Gerald Jay Sussman, and Daniel Weitzner. Data-Purpose Algebra: Modeling Data Usage Policies. In *Eighth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY '07)*, pages 173–177, June 2007.

[67] Ragib Hasan, Radu Sion, and Marianne Winslett. Introducing Secure Provenance: Problems and Challenges. In *Proceedings of the ACM Workshop on Storage Security and Survivability (StorageSS)*, pages 13–18. ACM Press, 2007.

[68] Ragib Hasan, Radu Sion, and Marianne Winslett. The Case of the Fake Picasso: Preventing History Forgery with Secure Provenance. In *FAST '09 7th conference on File and storage technologies (2009)*, pages 1–14, Berkeley, CA, USA, 2009. USENIX Association.

[69] Judith Hochberg, Kathleen Jackson, Cathy Stallings, J. F. McClary, David DuBois, and Josephine Ford. NADIR: An automated system for detecting network intrusion and misuse. *Computers & Security*, 12(3):235–248, May 1993.

[70] David A. Holland, Margo I. Seltzer, Uri Braun, and Kiran-Kumar Muniswamy-Reddy. PASSing the provenance challenge. *Concurrency and Computation: Practice & Experience*, 20(5):531–540, 2008.

[71] Siv Hilde Houmb, Geri Georg, Jan Jürjens, and Robert France. An Integrated Security Verification and Security Solution Design Trade-off Analysis. *Integrating Security and Software Engineering: Advances and Future Visions*, pages 190–219, 2007.

[72] Jun Ho Huh and Andrew Martin. Trusted logging requirements for grid computing. In *Third Asia-Pacific Trusted Infrastructure Technologies Conference*, pages 30–42, 2008.

[73] K. Ilgun, R.A. Kemmerer, and P.A. Porras. State transition analysis: a rule-based intrusion detection approach. *IEEE Transactions on Software Engineering*, 21(3):181–199, March 1995.

[74] Kathleen Jackson, David DuBois, and Cathy Stallings. An expert system application for network intrusion detection. In *14th National Computer Security Conference*, pages 215–225, January 1991.

[75] W Jiang and C Clifton. A secure distributed framework for achieving k-anonymity. *VLDB Journal*, 15(4):316–333, November 2006.

[76] Andy Koronios Jill Slay. *IT Security Risk Management*. Wiley, Australia, 2006.

[77] Patrick Juola. Authorship attribution. *Foundations and Trends in Information Retrieval*, 1(3):233–334, 2006.

[78] Jan Jürjens. Viki v. 2.0.4, UMLsec Tool. `http://inky.cs.tu-dortmund.de/main2/jj/umlsectool/downloads.html`, 2002.

[79] Jan Jürjens. *Secure Systems Development with UML*. Springer, 2005.

[80] Jan Jürjens. Using interface specifications for verifying crypto-protocol implementations. In *Foundations of Interface Technologies*. FIT'08 @ ETAPS, 2008.

[81] Ted Kang and Lalana Kagal. Enabling Privacy-awareness in Social Networks. In *Intelligent Information Privacy Management Symposium at the AAAI Spring Symposium*, pages 98–103, 2010.

[82] Tamás Kifor, László Varga, Sergio Álvarez, Javier Vázquez-Salceda, and Steven Willmott. Privacy Issues of Provenance in Electronic Healthcare Record Systems. *Journal of Autonomic and Trusted Computing (JoATC)*, 3:1–10, 2008.

[83] Michael Koch and Kathrin Möslein. Identities Management for E-Commerce and Collaboration Applications. *International Journal of Electronic Commerce IJEC*, 9:11–29, 2005.

[84] David W. Kravitz. Digital Signature Algorithm. U.S. Patent 5,231,668, 1993.

[85] Sandeep Kumar. *Classification and Detection of Computer Intrusions*. Phd thesis, Purdue University, West Lafayette, Indiana, USA, 1995.

[86] Sandeep Kumar and Eugene H. Spafford. A Pattern Matching Model for Misuse Intrusion Detection. In *17th National Computer Security Conference*, pages 11–21, 1994.

[87] Brian Neil Levine and Marc Liberatore. DEX: Digital evidence provenance supporting reproducibility and comparison. *Digital Investigation*, 6:S48–S56, 2009.

[88] Boots UK Limited. Data protection register. `http://www.ico.gov.uk/ESDWebPages/DoSearch.asp?reg=4488469`, December 2002.

[89] Gary Locke and Patrick D. Gallagher. Privilege Management (Draft NISTIR 7657). http://csrc.nist.gov/publications/drafts/nistir-7657/draft-nistir-7657_privilege-management.pdf, 2009.

[90] Wentian Lu and Gerome Miklau. Auditing a Database under Retention Restrictions. In *IEEE International Conference on Data Engineering (ICDE)*, pages 42–53, 2009.

[91] Teresa F. Lunt. IDES: An Intelligent System for Detecting Intruders. In *Computer Security, Treat and Countermeasures Symposium*, pages 110–121, Rome, Italy, 1990.

[92] Linh Thao Ly, Stefanie Rinderle-Ma, and Peter Dadam. Design and Verification of Instantiable Compliance Rule Graphs in Process-Aware Information Systems. In *22nd International Conference on Advanced Information Systems Engineering (CAiSE'10)*, pages 9–23, 2010.

[93] Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke, and Muthuramakrishnan Venkitasubramaniam. ℓ-diversity: Privacy beyond k-anonymity. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):52–62, 2007.

[94] Luther Martin. Key-Management Infrastructure for Protecting Stored Data. *Computer*, 41(6):103–104, June 2008.

[95] Terry Mayfield, Virgil D Gligor, Janet A Cugini, John M Boone, and Robert W Dobry. Security Criteria for Distributed Systems: Functional Requirements. Technical report, Institute for Defense Analyses, USA, September 1995.

[96] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[97] Gerome Miklau, Brian Neil Levine, and Patrick Stahlberg. Securing history: Privacy and accountability in database systems. In *Conference on Innovative Data Systems Research (CIDR)*, pages 387–396, 2007.

[98] Simon Miles. Electronically querying for the provenance of entities. In *Proceedings of the International Provenance and Annotation Workshop IPAW*, pages 184–192. Springer, November 2006.

[99] Simon Miles, Paul Groth, and Luc Moreau. Provenance Query API Functional Specification. `http://twiki.pasoa.ecs.soton.ac.uk/bin/view/PASOA/ProvenanceQueryFunctionalSpec`, March 2006.

[100] Simon Miles, Paul Groth, Steve Munroe, and Luc Moreau. PrIMe: A Methodology for Developing Provenance-Aware Applications. *ACM Transactions on Software Engineering and Methodology*, 20(3):8–18, August 2011.

[101] Simon Miles, S Wong, Weijian Fang, Paul Groth, K Zauner, and Luc Moreau. Provenance-based validation of e-science experiments. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(1):28–38, March 2007.

[102] C. J. Mitchell, F. C. Piper, and P. R. Wild. *Digital Signatures*, chapter 6 of Contemporary cryptology: The science of information integrity, pages 325–378. IEEE Press, Piscataway NJ, 1992.

[103] Luc Moreau. The foundations for provenance on the web. *Foundations and Trends in Web Science (In Press)*, November 2010.

[104] Luc Moreau, Ben Clifford, Juliana Freire, Yolanda Gil, Joe Futrelle, Natalia Kwasnikowska, Simon Miles, Paolo Missier, Jim Myers, Yogesh Simmhan, Eric Stephan, Jan Van Den Bussche, and Beth Pale. The Open Provenance Model Core Specification (v1.1). *Future Generation Computer Systems*, pages 1–30, 2010.

[105] Luc Moreau, Paul Groth, Simon Miles, Javier Vázquez, John Ibbotson, Sheng Jiang, Steve Munroe, Omer Rana, Andreas Schreiber, Victor Tan, and László Varga. The provenance of electronic data. *Communications of the ACM*, 51(4):52–58, April 2008.

[106] Abdelaziz Mounji, Baudouin Le Charlier, Denis Zampunieris, and NajiHabra. Distributed audit trail analysis. In *ISOC 1995 Symposium On Network and Distributed System Security*, pages 102–112, 1995.

[107] Shubha U. Nabar, Krishnaram Kenthapadi, Nina Mishra, and Rajeev Motwani. *A Survey of Query Auditing Techniques for Data Privacy*, volume 34, chapter 17, pages 415–431. Springer US, 34 edition, 2008.

[108] BBC News. Data lost by revenue and customs. `http://news.bbc.co.uk/1/hi/uk/7103911.stm`, 21 November 2007.

[109] BBC News. UK's families put on fraud alert. `http://news.bbc.co.uk/1/hi/uk_politics/7103566.stm`, 20 November 2007.

[110] Kaisa Nyberg and Rainer Rueppel. Message Recovery for Signature Schemes Based on the Discrete Logarithm Problem. In Alfredo Santis, editor, *Advances in Cryptology – EUROCRYPT'94*, volume 950 of *Lecture Notes in Computer Science*, pages 182–193, Berlin/Heidelberg, 1995. Springer-Verlag.

[111] Information Commissioner's Office. Data Protection Register. `http://www.ico.gov.uk/`, 1998.

[112] Hartig Olaf and Jun Zhao. Using Web Data Provenance for Quality Assessment. In *1st Int. Workshop on the Role of Semantic Web in Provenance Management (SWPM) at ISWC*, pages 6–16, Washington, USA, 2009.

[113] W Olin Sibert. Auditing in a Distributed System: SunOS MLS Audit Trails. In *11th National Computer Security Conference*, pages 82–90, 1988.

[114] Lorna Philip, Alison Chorley, John Farrington, and Pete Edwards. Data Provenance, Evidence-Based Policy Assessment, and e-Social Science. In *Proceedings of the Third International Conference on eSocial Science*, pages 1–10, 2007.

[115] J. Picciotto. The Design of an Effective Auditing Subsystem. In *IEEE Symposium on Security and Privacy*, pages 13–22, Oakland, CA, USA, April 1987. IEEE Computer Society.

[116] Bogdan Popa. 15-year-old student breaks into school computer system. `http://news.softpedia.com/news/15-year-old-Student-Breaks-into-School-Computer-System-86076.shtml`, May 2008. Softpedia Web Page.

[117] Bart Preneel. Cryptographic hash functions. *European Transactions on Telecommunications*, 5(4):431–448, 1994.

[118] Marsh Ray and Steve Dispensa. Renegotiating TLS. Technical report, PhoneFactor, Inc., November 2009.

[119] Christoph Ringelstein and Steffen Staab. PAPEL: A Language and Model for Provenance-Aware Policy Definition and Execution. In *8th International Business Process Management Conference*, pages 195–210, 2010.

[120] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120 – 126, 1978.

[121] Martin Roesch. Snort - Lightweight Intrusion Detection for Networks. In *Conference on Systems Administration*, pages 229–238, 1999.

[122] S.I. Schaen and B.W. McKenney. Network auditing: issues and recommendations. In *Proceedings Seventh Annual Computer Security Applications Conference*, pages 66–79, San Antonio, TX, USA, 1991. IEEE Comput. Soc. Press.

[123] Bruce Schneier. *Secrets and lies: digital security in a networked world*. John Wiley & Sons, 1st edition, 2000.

[124] Kenneth .F. Seiden and Jeffrey .P. Melanson. The auditing facility for a VMM security kernel. In *IEEE Computer Society Symposium on Research in Security and Privacy*, pages 262–277, Oakland, CA , USA, 1990. IEEE Comput. Soc. Press.

[125] Pasha Shabalin. Model Checking UMLsec Models. Master's thesis, Department of Informatics, TU München, Germany, 2004.

[126] Hovav Shacham and Dan Boneh. Fast-track session establishment for TLS. In Mahesh Tripunitara, editor, *Proceedings of NDSS 2002*, pages 195–202. Internet Society (ISOC), February 2002. Extended abstract of SBR04 journal paper.

[127] Hovav Shacham, Dan Boneh, and Eric Rescorla. Client-side caching for TLS. *ACM Transactions on Information and System Security*, 7(4):553–575, 2004.

[128] Y Simmhan, B Plale, and D Gannon. A survey of data provenance in e-science. *SIGMOD Record*, 34:31–36, 2005.

[129] Stephen E. Smaha. svr4++, A Common Audit Trail Interchange Format for UNIX. Technical report, Haystack Laboratories, Inc., Austin, Texas, USA, Version 2.2 1994.

[130] Steven R Snapp, James Brentano, Gihan Dias, Terrance Goan, Louis Todd Heberlein, Che-lin Ho, Karl Levitt, Biswanath Mukherjee, Stephen Smaha, Tim Grance, Daniel Teal, and Douglas Mansur. DIDS(Distributed Intrusion Detection System) -Motivation, Architecture, and An Early Prototype. In *14th National Computer Security Conference*, pages 167–176, October 1991.

[131] Gernot Stenz and Andreas Wolf. e-SETHEO: An Automated Theorem Prover. In *TABLEAUX '00: Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, pages 436–440, London, UK, 2000. Springer-Verlag.

[132] Geoff Sutcliffe and Christian Suttner. The TPTP Problem Library for Automated Theorem Proving. `http://www.cs.miami.edu/~tptp/`.

[133] L Sweeney. k-anonymity: A model for protecting privacy. *International Journal on Uncertainty, Fuzziness and Knowledgebased Systems*, 10(5), 2002.

[134] Committee On National Security Systems. National Information Assurance (IA) Glossary. `www.cnss.gov`, June 2006.

[135] Victor Tan, Paul Groth, Simon Miles, Sheng Jiang, Steve Munroe, Sofia Tsasakou, and Luc Moreau. Security Issues in a SOA-based Provenance System. In *Proceedings of the International Provenance and Annotation Workshop (IPAW'06)*, pages 203–211, Chicago, Illinois, 2006. Springer-Verlag.

[136] The Open Group. *Distributed Audit Service (XDAS) Preliminary Specification P441*. Berkshire, United Kingdom, 1997.

[137] Giorgos Tylissanakis and Yiannis Cotronis. Data Provenance and Reproducibility in Grid Based Scientific Workflows. In *Workshops at the Grid and Pervasive Computing Conference (GPC '09)*, pages 42–49, Washington, DC, USA, 2009. IEEE Computer Society.

[138] Ozlem Uzuner, Yuan Luo, and Peter Szolovits. Evaluating the state-of-the-art in automatic de-identification. *Journal of the American Medical Informatics Association : JAMIA*, 14(5):550–63, January 2007.

[139] Henk C. A. Van Tilborg. *Encyclopedia of Cryptography and Security*. Springer, 2005.

[140] Serge Vaudenay. Security Flaws Induced by CBC Padding - Applications to SSL, IPSEC, WTLS ... In *EUROCRYPT '02 Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques: Advances in*

Cryptology, pages 534–546, Amsterdam, The Netherlands, May 2002. Springer-Verlag.

[141] Jeffrey A. Vaughan, Limin Jia, Karl Mazurak, and Steve Zdancewic. Evidence-based audit. In *21th IEEE Computer Security Foundations Symposium*, pages 177–191. IEEE Computer Society, 2008.

[142] Javier Vázquez-Salceda and Sergio Alvarez-Napagao. Using SOA Provenance to Implement Norm enforcement in e -Institutions. In Jomi Fred Hübner, Eric Matson, Olivier Boissier, and Virginia Dignum, editors, *Coordination, Organizations, Institutions and Norms in Agent Systems IV*, volume 5428 of *Lecture Notes in Computer Science*, pages 188–203, Berlin, Heidelberg, 2009. Springer-Verlag.

[143] Dominique de Waleffe and Jean-Jacques Quisquater. Better login protocols for computer networks. In *Computer Security and Industrial Cryptography - State of the Art and Evolution, ESAT Course*, pages 50–70, London, UK, 1993. Springer-Verlag.

[144] Y. Richard Wang and Stuart E. Madnick. A Polygen Model for Heterogeneous Database Systems:The Source Tagging Perspective. In *16th International Conference on Very Large Data Bases (VLDB '90)*, pages 519–538, San Francisco, CA, USA, 1990. Morgan Kaufmann Publishers Inc.

[145] Christoph Weidenbach, Uwe Brahm, Thomas Hillenbrand, Enno Keen, Christian Theobald, and Dalibor Topic. S PASS Version 2.0. In *CADE-18: Proceedings of the 18th International Conference on Automated Deduction*, pages 275–279, London, UK, 2002. Springer-Verlag.

[146] Daniel J. Weitzner, Harold Abelson, Tim Berners-Lee, Joan Feigenbaum, James Hendler, and Gerald Jay Sussman. Information accountability. *Communications of the ACM*, 51(6):82–87, 2008.

[147] Jenifer Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *Second Biennial Conference on Innovative Data Systems Research (CIDR 2005)*, pages 262–276, Asilomar, California, 2005.

[148] Shouhuai Xu, Qun Ni, Elisa Bertino, and Ravi Sandhu. A Characterization of the problem of secure provenance management. In *International Conference on Intelligence and Security Informatics*, pages 310–314, Texas, USA, June 2009. IEEE.

[149] Jun Zhao, Carole Goble, Robert Stevens, and D. Mining Taverna's semantic web of provenance. *Concurrency and Computation: Practice and Experience*, 20(5):463–472, 2008.

[150] Yong Zhao, Michael Wilde, and Ian Foster. Applying the Virtual Data Provenance Model . In Luc Moreau and Ian Foster, editors, *Provenance and Annotation of Data*, volume 4145 of *Lecture Notes in Computer Science*, pages 148–161, Berlin, Heidelberg, 2006. Springer-Verlag.