prisma cloud

Improved Guidelines and Architecture for Secure Service Composition

Deliverable D7.6

Editor Name	Thomas Loruenser
Type	Report
Dissem. Level	Public
Release Date	July 31, 2017
Version	1.0

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 644962.

More information available at https://prismacloud.eu.

Copyright Statement

Document information

Project Context

Work Package	WP7 Composition of Next-Generation Secure Cloud Services
Task	T7.3 Architecture and Guidelines for Secure Service Composition
Dependencies	D7.4, D7.8, D5.3, D5.4, D5.7, D5.9, D5.10

Author List

Organization	Name	E-mail
AIT	Thomas Lorünser	thomas.loruenser@ait.ac.at
AIT	Andreas Happe	andreas.happe@ait.ac.at
AIT	Stephan Krenn	stephan.krenn@ait.ac.at
AIT	Aleksandar Hudic	aleksandar.hudic@ait.ac.at
XiTrust	Katrin Riemer	katrin.riemer@xitrust.com
UNIL	Thomas Länger	thomas.laenger@unil.ch
ETRA	Alberto Zambrano	
IBM	Micha Moffie	
ATOS	Angel Palomares	
UNEW	Ioannis Sfyrakis	ioannis.sfyrakis@newcastle.ac.uk
ETRA	Ana Martinez	
UNI PASSAU	Henrich C. Pöhls	hp@sec.uni-passau.de

Reviewer List

Organization	Name	E-mail
FCSR	Eleonora Ciceri	ciceri.eleonora@hsr.it
UNIL	Thomas Laenger	thomas.laenger@unil.ch

Version History

Version	Date	Reason/Change	Editor
0.01	2016-10-12	Started with content from D7.5.	Thomas Lorünser
0.02	2017-05-18	Added encryption proxy service description (EPaaS).	Alberto Zambrano
0.03	2017-05-18	Added BDA service description (BDAaaS).	Micha Moffie and Thomas Lorünser
0.04	2017-05-23	Added verifiable statistics service (VSaaS).	Angel Palomares and Thomas Lorünser
0.05	2017-06-01	Added infrastructure auditing service (IAaaS).	Ioannis Sfyrakis
0.06	2017-06-23	Added PIDM service description (PIDMaaS).	Ana Martinez
0.07	2017-07-06	Sanitizing service descriptions.	Thomas Lorünser
0.08	2017-07-13	Update IAaaS service.	Ioannis Sfyrakis
0.09	2017-07-14	Updated SAEaaS description and threat table.	Katrin Riemer
0.10	2017-07-17	Updated SAEaaS description with information on the XML message handling	Henrich C. Pöhls
0.11	2017-07-19	SAEaaS threat analysis on XML Format and info for SAE service, minor updates and additions	Henrich C. Pöhls
0.12	2017-07-20	Added design patterns subsection	Thomas Länger
0.13	2017-07-20	Update SWOT info for services.	Thomas Lorünser
0.20	2017-07-29	Address final reviewer comments.	Thomas Lorünser
1.0	2017-07-31	Final fixes for submission.	Thomas Lorünser

Executive Summary

This report covers the results of "Task 7.3 Architecture and guidelines for secure service composition". In particular, it presents the PRISMACLOUD architecture and the associated development methodology called *Cryptographic Software Development Lifecycle (CryptSDLC)*. It also provides recommendations for the implementation of a service development lifecycle on the basis of CryptSDLC and defines a project wide documentation standard for services. Additionally, the report provides a full set of service documentation according to the standard. The report is based on preliminary version *"D7.5 First version of guidelines and architecture for secure service composition'"*, which was released internally in M18.

The PRISMACLOUD project is a huge undertaking and produces outcome in many different disciplines and layers. The **PRISMACLOUD architecture** facilitates a way to structure and categorize the technical outcomes, but more importantly to improve service development processes and project communication. It provides a tangible abstraction of the complexity involved with the construction of cryptographically secured services and will provide the project context for the research and development activities in the different work packages.

Together with the architecture, we also established a **development methodology** which leverages the architectural layers in order to improve the quality and efficiency of application development and to maximize the potential reuse of existing work. The CryptSDLC methodology enriches well known security-by-design approaches with a work flow for cryptographic service design. Furthermore, it standardizes the special steps necessary when going from one layer to the other and aligns them to the general phases of classical SDL models. The major steps are Derive, Translate and Map from top to bottom and Prove, Deploy and Extract from bottom to top.

Based on the conceptual results, we specified the **guidelines for service development**, which give concrete recommendations for the implementation of the service development process. They specify in detail which development methodlogies should be adopted and how they can be enriched with the new concept. They define concrete phases and steps as well as document artefacts to support the work flow. Out of that, we also defined a **project standard** to be used in the service development for the PRISMACLOUD services.

In addition to the methodological work, this document also presents the **documentation of services** according to the proposed project standard. For this interim version, the two most mature services have been selected for the proposed guidelines to be applied. We selected the *Secure Archiving* and *Data Sharing* services which are both instantiations of the Secure Object Storage tool. This part was done to test and evaluate the proposed approach on a real world scenario. However, the final version of the document will contain an improved version of the methodology as well as the full documentation of all 8 PRISMACLOUD services.

This document is intended to be used by cloud service providers or middleware and system developers who intend to either adopt PRISMACLOUD services or make variants thereof.

It should serve as a starting point and reference, because it contains the core service documentation, i.e., the underlying key ideas and the benefits, service and deployment model information, as well as software development and assurance monitoring information for the PRISMACLOUD services.

Table of Contents

List of Figures

List of Tables

1 Introduction

This section is intended to give an overview of the document, introduce the expected content, explain the project context and give the reader some guidance for quick access to the most relevant information.

1.1 Scope of the document

The major purpose of this document is to describe how the PRISMACLOUD primitives, protocols and methods are combined and documented to build the PRISMACLOUD services. This document presents the results of work carried out in *Task 7.3 Architecture and guidelines for secure service composition*. It is the final documentation of the services developed in PRISMACLOUD and built on the preliminary results presented in *D7.5 "First guidelines and architecture for secure service composition'"*, which has been made internally available in M18.

To achieve this goal we first developed the PRISMACLOUD **architecture**, which provided the context for project work during the implementation phase of PRISMACLOUD. The research and development activities in the project have been grouped according to this 4-tier architectural model and it had been a valuable resource for structuring the work in the project. Together with the architecture we also established a **development methodology** which leverages the architectural layers in order to improve the quality and efficiency of application development and to maximize the potential reuse of existing work. The methodology is called *Cryptographic Service Development LifeCycle (CryptSDLC)* and extends well known security-by-design approaches with a work flow suitable for cryptographic design.

Based on the architecture and the methodology we specify **guidelines for service development**, which are the second contribution of this document. The guidelines detail recommendations for the implementation of the service development process and define document artifacts that support the work flow. Out of that, we also defined a **project standard** to be used in the service development for the PRISMACLOUD services.

In addition to the methodological work, this document also presents the **service documentation** of all eight PRISMACLOUD services according to the proposed project standard. Therefore, this document is best suited for cloud service providers or middleware and system developers who intend to either adopt PRISMACLOUD services or make variants thereof. It serves as a starting point and reference, because it contains the core service documentation, i.e., its key ideas and benefits, service and deployment model information as well as software development information.

This report contributes to the following PRISMACLOUD Objectives:

- *Objective 4: Development of a methodology for secure service composition*: With the developed architecture and methodology for the composition of secure services it clearly contributes to this objective. Also the guidelines developed are key to

implement appropriate processes during service development and also support application level development in the best possible way and by design.

- *Objective 3: Creation of enabling technologies for cloud infrastructures*: The development of the PRISMACLOUD services as well as their standardized specification and documentation greatly improves the development process and increases the exploitation potential. Furthermore, the architecture and development methodology are intended to be broadly adopted and not only restricted to the specific service developments we do in PRISMACLOUD.

1.2 Relation to other project work

This report, like all other work done in "WP7 Composition of next-generation secure cloud services", is strongly related to all other technical work packages of the project. However, for D7.6 this fact is particularly true because it introduces the PRISMACLOUD architecture and defines a project standard for secure service development. This architecture helped to structure and align the efforts of all project partners towards the project goal, being the design, development, and validation of cryptographically enhanced cloud services with improved security and privacy, transcending the current state-of-the-art. The PRISMACLOUD architecture developed in WP7 is a central concept in PRISMACLOUD and is used throughout all other work packages.

In "WP3 End user and business deployment", business and governance models are developed according to the architecture, and usability issues are investigated at various levels. It is also in WP3, that cloud security patterns are developed and maintained for the single proposed cloud services in support of the development process of tools and services, and as basis for an impact analysis of what end users can expect with respect to security and privacy when their applications rely on the proposed cloud services.

In "WP4 Advancement of enabling cryptographic primitives, protocols and schemes" and "WP5 Basic building blocks for secure services" the architecture guides the research by defining the goals to be achieved by the single *tools* on the cryptographic layer. The current report includes an overview of the tools developed in WP5 in the Annex.

Please note, a preliminary version of the PRISMACLOUD architecture has already been introduced in *D6.4* to support the selection and specification of tools for software implementation. Development and communication of the architecture had already been started as early as M12, in order to streamline multidisciplinary project work in the project. However, the architecture has been developed in WP7 and is going to be fully introduced in this report.

This report further contains guidelines and defines project wide standards for service development influencing all tools and services development activities. The toolbox specification in WP5 is guided by the principles described in this report and the software development standards also apply to activities in WP6 (tool level) and WP7 (service level). The developed methodology is also closely related to the security and privacy by design guidelines developed in "T7.1 Security and privacy by design" and the respective deliverable D7.1.

The methodology is specifically addressing the complexity and multidisciplinary nature found when dealing with the integration of cryptography into services and applications. It shall help to lower the entry bar for faster adoption of novel cryptographic solutions in real world applications. It can be seen as a complementary more application design oriented extension to the security and privacy by design approach discussed in T7.1.

Finally, this report presents main steps in secure service design applied to the particular PRISMACLOUD services. It presents the services in detail and the steps undertaken to guarantee application security in service development as well as necessary operational steps after deployment. Therefore, it serves as a reference for service development and piloting in the project.

Furthermore, the report is complemented by other WP7 deliverables which are closely related to service development, deployment and usage. In particular, the software architecture and service level interfaces are documented in "D7.8 Software architecture and interface specification" (and follow-up deliverables D7.8 and D7.9), and "D7.3 Progress report on holistic security model for secure service composition" (and follow-up deliverable D7.4) are giving the application developers' and service level view on the PRISMACLOUD services.

1.3 Structure of the document

This document is structured into three parts. In Section 2—the first part—the PRISMACLOUD architecture is introduced and explained in detail as well as the development methodology.

The second part in Section 3 presents the guideline for composition and development of the PRISMACLOUD services. It combines widely accepted standards from application development with new ones specifically targeted at the cloud domain and further enriches them with PRISMACLOUD specific extensions that aid the development of secure cryptographic services.

The third part in Section 4 applies the guidelines to the services. It contains as standardized documentation of the eight PRISMACLOUD services, which have been selected in the project for piloting.

The summary and conclusion in Section 5 is dedicated to the experiences made during development of the project architecture, guidelines and standards for secure service development and the lessons learned during applying them for the documentation of the PRISMACLOUD services.

2 Prismacloud Architecture

The PRISMACLOUD project is a huge undertaking and produces outcome in many different disciplines and layers. To structure and categorize the technical outcomes, but more importantly to improve service development processes and project communication, we introduce the PRISMACLOUD architecture.

The architecture is accompanied with a novel methodology for secure service design and development and already provides a specific set of tools and services developed in the PRISMACLOUD project as a portfolio of standard solutions. This architecture is intended to provide a tangible abstraction of the complexity involved with the construction of cryptographically secured services and will provide the project context for the research and development activities in the different work packages. For instance in WP7, the PRISMACLOUD architecture will be used to describe the services from the viewpoint of service architects and software engineers.

While the PRISMACLOUD architecture and methodology has been developed within WP7, close cooperation has been established with all other work packages. A first version of the architecture has already been integrated in WP6 deliverable D6.4 in order to support a structured selection and specification of tools for software implementation. This was a first—successful—test for the architecture and its use within the project-wide working context. In this section we present the full architecture and how its intended use during and after the project.

2.1 Motivation and Idea

The development of the specific PRISMACLOUD architecture was driven by two factors: on one hand, we were looking for ways to present the project research and outcome in a structured and understandable way. On the other hand, we needed a tool to improve communication between the experts of the different disciplines involved in the project. Especially the interface between cryptographic researchers and software architects turned out to be very challenging and before the release of the architecture it was not clear how the security properties developed at the cryptographic layer and based on strong notions can be transferred to the cloud applications. Furthermore, the latter seems to be extremely challenging for the domain of cloud computing. Cloud computing is a new paradigm for the delivery of IT resources and mainly builds on the outsourcing paradigm. It's also an extremely agile environment introducing enormous new challenges for cryptography. If a service needs to be well supported by cryptography it typically requires various different primitives to be applied in order to fully provide the required properties. Furthermore, they are typically only added in an ad-hoc fashion without access to the required cryptographic expertise. Even worse, the existing primitives proposed by research often only partially provide what is needed or are not compatible with used data formats, encoding or the like. In those cases the cryptographic support is often not added at all but replaced by additional technical or organizational means at higher layers. It is clear, that this alternative solutions can never have the same strength as the pure cryptographic support

and often introduce additional overhead and complexity which further worsens the quality of the overall product.

Historically, we did not face this problem on a broad scale. Cryptography was mainly used to solve well understood problems in a reliable way and without support for flexible scenarios. Most prominently it was used to establish secure channels in the Internet, to protect the communication from point-to-point in general, to prove the authenticity of software or documents, or protect passwords and other data when stored. All this tasks have well defined security requirements which are not supposed to change over time or per application. The same is true for the underlying trust models. However, from the past we learned that even such standard designs could miserably fail if they had been done in an ad-hoc fashion. In fact, we know of many examples where ad-hoc integration of cryptography failed and thus its benefits vanished. Even worse, the analysis of first approaches towards secure channels showed that combining secure cryptographic primitives in the wrong way can lead to vulnerable protocols which even undermine the security properties of the underlying primitives. The most prominent example for a protocol which experienced such a problem is the secure socket layer (SSL) protocol. It is intended to establish a secure communication channel between two peers over an untrusted transmission system. The wrong combination of encryption (ENC) modes and message authentication codes (MAC) led to padding oracle attacks [Kra01] which allowed to break the otherwise secure encryption scheme. This design problem was caused by a lack of understanding of methods to combine primitives to schemes in a secure way. In the last 15 years a large body of research emerged around this topic—and provable security in general—and today we have access to methodologies which greatly assist this design phase.

Today we face a similar situation within cloud computing. We know how to apply secure channels to protect communication between clients and service providers, but even without the impact of cloud infrastructures we have only little knowledge about how to realize secure services that protect data over their whole lifecycle.

In addition, existing requirements do not map perfectly upon the new cloud environment. A simple example for this problem is cloud storage. Modern cloud storage does not resemble traditional disk storage anymore as it should also support collaboration in dynamic user groups. This complicates application of technologies from point-to-point communication system as this communication pattern cannot be easily supported by existing cryptographic tools.

A second problem when building secure services is related to their software implementation. Even if all algorithms and protocols were chosen correctly, their software implementation requires considerable expertise not generally available to application developers. The implementation should resist attacks exploiting available side channels and also built with security in mind.

In summary, we aim to integrate sound cryptographic design, side channel resistance, efficient implementation techniques as well as secure software development into an *integrated secure services development lifecycle*. We want to create building blocks for future projects as well as document how to integrate those building blocks in a secure manner. Further-

more, we must avoid problems known from ad-hoc design of cryptographic systems but introduce a holistic methodology that fosters reuse of work.

The goals of the architecture are:

- incorporate cryptographically sound design methodologies
- support adoption by efficient and secure implementations of generic building blocks
- give guidance for use of cryptography in a developer friendly way
- reduce configuration and integration errors as far as possible without limiting the flexibility
- foster exploitation of results outside their pilot domains
- enable fast adoption of project results

In the following we are presenting the architecture developed in PRISMACLOUD which is both a conceptual approach to structure functionality as well as a methodology for secure service design and composition.

2.2 Architecture Layers

The PRISMACLOUD architecture is shown in Figure 1 and comprises four different layers, three of which are of major interest for this document, namely: primitives, tools and services. Subsequently, we introduce them and discuss the ideas behind them.

2.2.1 Primitives Layers

The lowest layer consists of *cryptographic primitives and protocols* which represent basic cryptographic building blocks, e.g., signature schemes or cryptographic protocols. They typically provide very specific functionality and are limited in use. Furthermore, the analysis and development of cryptographic primitives is done by cryptographers with strong mathematical background, ideally in a provable manner, i.e., by rigorous mathematical methods and models. The very specialized knowledge required for this work is not shared by software developers.

In this layer we conduct cryptographic research and will advance its state-of-the-art. As shown in Figure 1 and summarized in Table 1, in PRISMACLOUD we particularly focus on 11 primitives which are essential for building our services but require additional research to provide the required functionality and efficiency for application usage. Thus, the research directions followed on the primitives and protocols layer is ultimately driven by the requirements derived from the use cases and their services. The analysis of the state of the art in the first project phase revealed gaps which have to be filled during the project.

Major outcome producted within this layer will be new cryptographic primitives and protocols, specifically addressing features required in the project. The results of cryptographic research is continuously published for open access and presented on scientific venues as recorded in WP9 and documented in internal reports of WP4 and WP5.

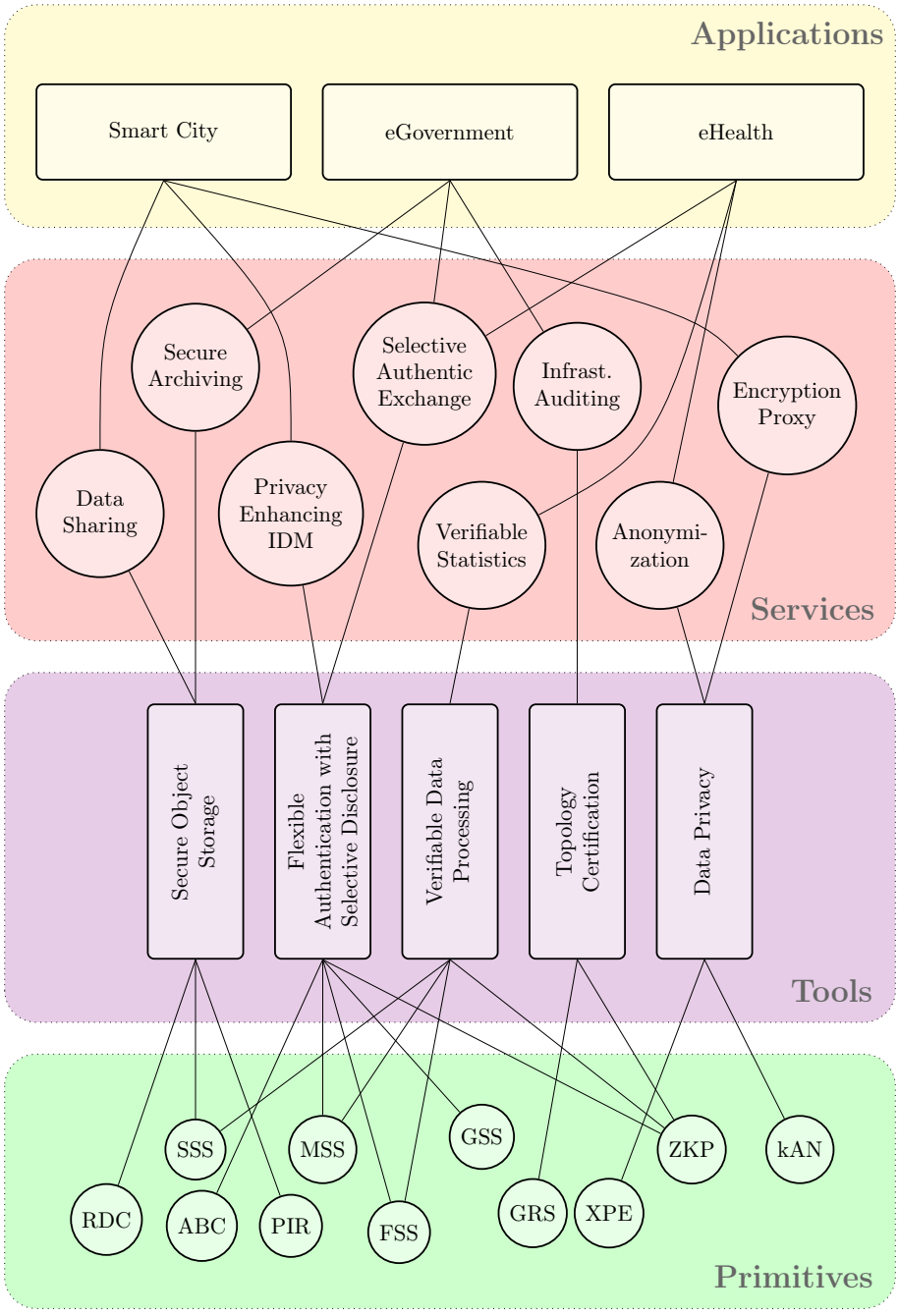


Figure 1: The PRISMACLOUD Architecture

Table 1: Portfolio of PRISMACLOUD primitives and protocols.

ID	Name	Abbrev.
PC.Primitive.1	Secret Sharing Schemes	SSS
PC.Primitive.2	Remote Data Checking	RDC
PC.Primitive.3	Private Information Retrieval	PIR
PC.Primitive.4	Malleable Signature Schemes	MSS
PC.Primitive.5	Functional Signature Schemes	FSS
PC.Primitive.6	Attribute-Based Credentials	ABC
PC.Primitive.7	Group Signature Schemes	GSS
PC.Primitive.8	Zero-Knowledge Proofs	ZKP
PC.Primitive.9	Graph Signature Schemes	GRS
PC.Primitive.10	Format- and Order-Preserving Encryption	XPE
PC.Primitive.11	k-Anonymity	KAN

Table 2: Portfolio of PRISMACLOUD tools.

ID	Name	Abbrev.
PC.Tool.1	Secure Object Storage	SECSTOR
PC.Tool.2	Flexible Authentication with Selective Disclosure	FLEXAUTH
PC.Tool.3	Verifiable Data Processing	VERIDAP
PC.Tool.4	Topology Certification Tool	TOPOCERT
PC.Tool.5	Data Privacy Tool	DATAPRIV

2.2.2 Tools Layer

The second layer is denoted as *tools*. In the context of the PRISMACLOUD architecture, tools are a concept used to better communicate techniques to software developers and architects in an more accessible way. They provide higher level functionality as a combination of primitives. However, the design of tools is still based on rigorous cryptographically sound models and ideally still provides provable security for realistic adversary models. A tool can therefore be regarded as an abstract concept or piece of software, e.g., a library, which is composed of various primitives which can be parametrized in various different ways.

As shown in Table 2 in PRISMACLOUD we are developing 5 such tools targeted at specific application scenarios. The outcome on this level will be twofold: on the one hand, we define and specify parametrisable PRISMACLOUD tools which can be used to build or augment services with cryptographic features. On the other hand, the tools and their most important features are going to be implemented during the project. They will be available in form of software libraries which will be used to build the services in the pilots.

The PRISMACLOUD tools will be the foundation for mid-term exploitation of PRISMACLOUD results and should foster their fast adoption. Each of them will be provided with a full set

of documentation targeted at cloud service architects and implementers, as well as software libraries of generic cryptographic components to speed up the service development process.

The following information will be provided in the respective tool design documents:

- Description of the abstract cryptographic tool by the following approach:
  - Specifying the tool and the underlying functionalities and protocols. The tools specify how the cryptographic building blocks have to be combined in a secure way and documents the underlying primitive and protocols in a form accessible to implementers and service architects.
  - Additionally, a feature matrix is presented to describe the achievable goals in a comprehensive way and identify dependencies and contradicting aspects. The feature matrix shall be mapped against service requirements and therefore guides the selection of the right configuration of the tool in specific service scenarios.
  - Configuration and usage guidelines will describe how the tool should be used and how erroneous usage can be avoided.
- A software framework consisting of one or multiple libraries providing core functionality.
- Guidance for the operational phase once a tool is deployed in a service or application, e.g., information about key management including revocation mechanisms or deployment recommendations for the distributed object storage tool.

The tools are developed in WP5; the development of the software components is part of the implementation work package (WP6). A high level overview of the current status of the tools is provided in Section A. The full specification of all tools will be released around month M30 of the project and will be available in the following deliverables[1]

ID	Title	Due
D5.3	Design and specification of the secure object storage tool	M30
D5.4	Design and specification of the data privacy tool	M30
D5.7	Design and specification of the topology certification tool	M32
D5.9	Design of flexible authentication with selective disclosure tool	M28
D5.10	Design and specification of the verifiable data processing tool	M32
D6.6	Final release of software implementation of selected components	M30

2.2.3 Service Layer

Cloud computing is radically changing the way we are consuming IT resources. It is also changing the way applications are built. Away from monolithic architectures, we are facing a shift towards service oriented architectures (SOA) where services can be flexibly interconnected and deployed in distributed fashion spreading traditional perimeters. In PRISMACLOUD we follow this trend and develop a portfolio of security enhanced *services*

[1] Deliverable names have been changed due to reviewer comments

Table 3: Portfolio of PRISMACLOUD services

ID	Name	Abbrev.	Model	Deploy.	Tool
PC.Service.1	Secure Archiving	SA	IaaS	private	SECSTOR
PC.Service.2	Data Sharing	DS	SaaS	public	SECSTOR
PC.Service.3	Selective Authentic Exchange	SAE	IaaS	private	FLEXAUTH
PC.Service.4	Privacy Enhancing IDM	PIDM	PaaS	public	FLEXAUTH
PC.Service.5	Verifiable Statistics	VS	PaaS	private	VERIDAP
PC.Service.6	Infrastructure Auditing	IA	PaaS	public	TOPOCERT
PC.Service.7	Encryption Proxy	EP	SaaS	private	DATPRIV
PC.Service.8	Anonymization	BDA	SaaS	private	DATPRIV

which can be integrated into larger applications in a flexible way. Furthermore, this services are built atop sound cryptographic concepts, i.e., out of the PRISMACLOUD tools, and therefore provide strong security guarantees. This approach is different to the broadly applied ad-hoc integration of cryptographic solutions into applications and helps to avoid common pitfalls in system design and implementation.

PRISMACLOUD services are built in a layered approach and a service can be seen as a customization of a particular tool for one specific application—thus we call a service an instantiation of a tool. In particular, a service is a way to deliver the tool to system and application developers, the users of the tools, in an preconfigured and accessible form. They will be able to integrate the services without deeper understanding of tools and primitives and ideally without even being an IT security expert.

In PRISMACLOUD we have selected a portfolio of 8 services to be specified and developed in detail. They will be used to demonstrate the benefits of the PRISMACLOUD tools and show how to use the PRISMACLOUD development methodology. In the following we are introducing the PRISMACLOUD services which are enumerated in Table 3, a comprehensive description of them is given in Section 4. Recall that a service can be seen as customization of one or multiple particular tools for one dedicated application and deployment scenario. It provides a specific set of features which has been identified as particularly useful for the class of applications the service is targeted.

The PRISMACLOUD *Secure Archiving Service (SA or SAaaS)* is a generic infrastructure service which can easily be integrated into cloud based backup scenarios while providing a demonstrable higher level of data privacy and availability than current cloud-based archiving solutions. The service model for this service is IaaS.

The PRISMACLOUD *Data Sharing Service (DS or DSaaS)* allows multiple parties to securely store data in a cloud-of-clouds network such that no single storage node learns plaintext data, while still enabling the owner to share the data with other users of the system, i.e., the data sharing service supports secure collaboration without the need to trust one single storage provider. The service is based on web technologies and enables ubiquitous easy access via web browser, i.e., it can be categorized as SaaS model.

The *Selective Authentic Exchange Service (SEA or SEAaaS)* enables users to move their authentic documents to a cloud service and then delegate the selective sharing of parts of these documents to another party, while maintaining the authenticity of the selected parts. The other party can then verify the authenticity of the received data. The service model of this service is PaaS and for different parts of this service, e.g. redact or verify, it is intended to be deployed in a public cloud model. For the signature generation the use of private signature generation keys is needed, thus this demands higher security, but following the "remote signature" model from the eIDAS regulation this might be deployed as a service in the public cloud as well. If the data within the document (before being selectively removed) requires heightened confidentiality protection, the cloud must offer this, otherwise at least the services for sign and redact (which handle the data before some parts are removed) should be deployed privately.

The *Privacy Enhancing ID Management Service (PIDM of PIDMaaS)* offers the capability of a privacy enhanced identity management. In particular, it allows users to store their attribute credentials obtained from some entity (e.g, a service provider or an authority) in this component and to realize a selective attribute disclosure functionality. The service model of this service is PaaS and the service is intended to be deployed in a public cloud.

The *Verifiable Statistics Service (VS or VSaaS)* provides the functionality to delegate the computation of verifiable statistics on authenticated data in a secure way. The computations have the feature of being public verifiability, i.e., any verifier can check whether an outsourced computation has been performed correctly, or not. The service model for this service is PaaS and the service should be deployed privately if data confidentiality is required.

The *Infrastructure Auditing Service (IA or IAaaS)* offers the capability to certify and prove properties of the topology of a cloud infrastructure without disclosing sensitive information about the actual infrastructure's blueprint. The service model associated to this service is IaaS and the service can be deployed in a public cloud.

The *Encryption Proxy Service (EP or EPaaS)* supports moving legacy applications to the cloud by encrypting sensitive information identified within HTTP traffic in a format and/or order preserving way. The delivery model associated to this service is SaaS and the gateway is intended for private cloud deployment but it is an enabler for hybrid cloud applications.

The *Anonymization Service (BDA or BDAaaS)* enables users to anonymize large data sets, and in particular database tables, i.e. its' acronym is derived from big data anonymization. The service allows users to identify private and sensitive information in the data sets and produce an anonymized version of the data set. The delivery model associated to this service is SaaS and the gateway is intended to be deployed in a private cloud but it is an enabler for hybrid cloud applications.

Please note, that we assigned them service and deployment models, although they could not always be unambiguously identified. We roughly categorized them to give the reader a first intuition. The assignment of the service model is also illustrated in Figure 2.

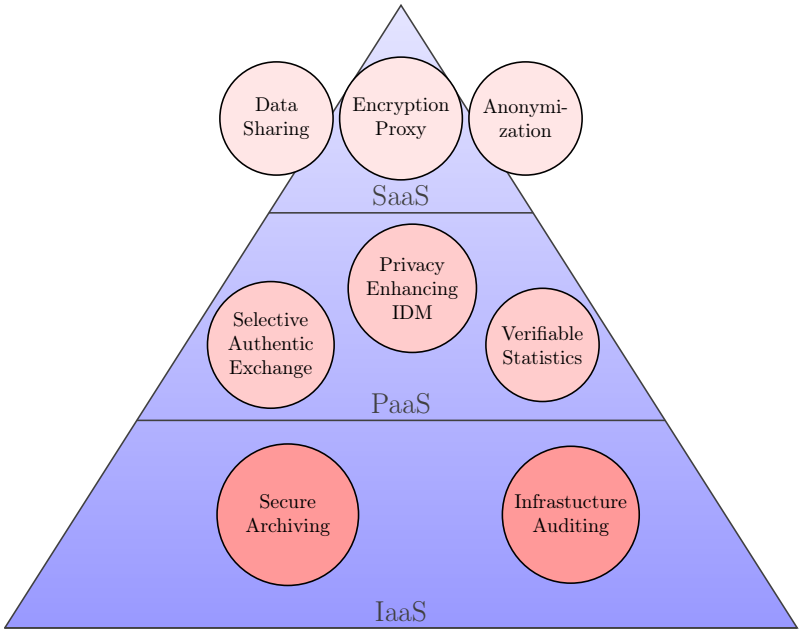


Figure 2: The PRISMACLOUD services in the reference architecture.

The PRISMACLOUD services are among the *most important outcome* of the project in terms of exploitation and will enable partners to commercialize them in a short timeframe after the project. The services will be well defined and specified throughout WP7 and implemented in WP6 and WP8. In addition to being specified, the services will also be tested and evaluated during piloting of the use cases in WP8. In Section 4 we fully document the PRISMACLOUD services and provide all relevant information for fast adoption. Together with the software and API documentation in D7.7 and the usage guidelines and SLA recommendations in D7.3 all information for rapid service deployment within new infrastructure is given.

Regarding exploitation and thus the respective target audience, the PRISMACLOUD services are intended to be used by cloud service providers as is. Nevertheless, if a new use case requires a modified version, all documentation is provided to develop a variant with minimal effort to enable a short time to market. This is achieved as developers don't have to start from scratch but can start from the well established PRISMACLOUD services and tools and thus are able to reuse available knowledge and software.

2.2.4 Application Layer

The application layer contains the applications targeted at real end users. Modern cloud based applications try to leverage the cloud as good as possible and support their scalability and elasticity through modularization and the use of service oriented principles. For this reason we introduced the concept of PRISMACLOUD services being a modern way to expose security functionality to cloud architects and application developers.

In PRISMACLOUD we are developing 4 applications to demonstrate the capabilities of the PRISMACLOUD services and their security benefits. They emerge from the use cases designed in the project and cover various domains where sensitive data is handled. Additionally, to show how the competitive finacial advantage gained through the usage of PRISMACLOUD services, we developed a business model for each service in D3.4. All applications will be showcased in the pilot:

Privacy Enhanced SIMON: The SIMON system is the result of an EU FP7 project led by partner ETRA and implements an electronic disable batch for parking. In PRISMACLOUD we are extending the existing prototype with the Encryption Proxy Service (EPaaS) to enable the operator to outsource the main database to a public cloud in a privacy preserving manner, i.e., without revealing personal information about users. In addition, the SIMON system will also be coupled with the Privacy Enhancing Identity Management Service (PIDMaaS) to reduce the amount of personal data collected in the central database and therefore protect the users by adoption of advanced data minimization techniques.

Evidence Sharing Platform: This application is intended to realize a privacy preserving evidence sharing platform on the basis of the Data Sharing Service (DSaaS). The evidences will manually collected and then stored on a cloud based platform such that no single cloud provider can infer plaintext of stored personal data or tamper with them.

e-Government IaaS Cloud: The partner LISPA will use two PRISMACLOUD services to improve their cloud offerings in the governmental sector. They are mainly an IaaS provider and will use the Secure Archiving Service (SAaaS) to add a highly reliable dispersed backup option for their customers. They will be able to leverage additional cloud offerings—including public ones—and be able to exploit hybrid cloud storage deployments. Additionally, with the integration of the Infrastructure Auditing Service (IAaaS) they will be able to increase users trust in their offerings by being able to prove that users get what they pay for without revealing information about their internal infrastructure.

Healthcare Data Sharing Platform: In this use case, the healthcare platform of partner FCSR[2] will be augmented and cloudified by the use of PRISMACLOUD services. They will use the Selective Authentic Data Sharing Service (SEAaaS) service to let patients manage their medical reports in a privacy friendly way while providing strong authentic guarantees. The Verifiable Statistics Service (VSaaS) will enable patients under medical observation equipped with telemedicine to optimize their medication. Finally, the use of the Anonymization Service (BDAaaS) will showcase how collaboration between researchers on clinical data can be improved without infringing the privacy of the patients involved.

2.3 Benefits of the Architecture

The main goal of the architecture is to support the development process of cryptographic applications. The model helps to cope with the complexity and interdisciplinary nature of cryptographic applications. It is based on the experiences made in the PRISMACLOUD project, which is a huge undertaking with people from very different disciplines involved,

[2]which is a result of EU FP7 project TClouds

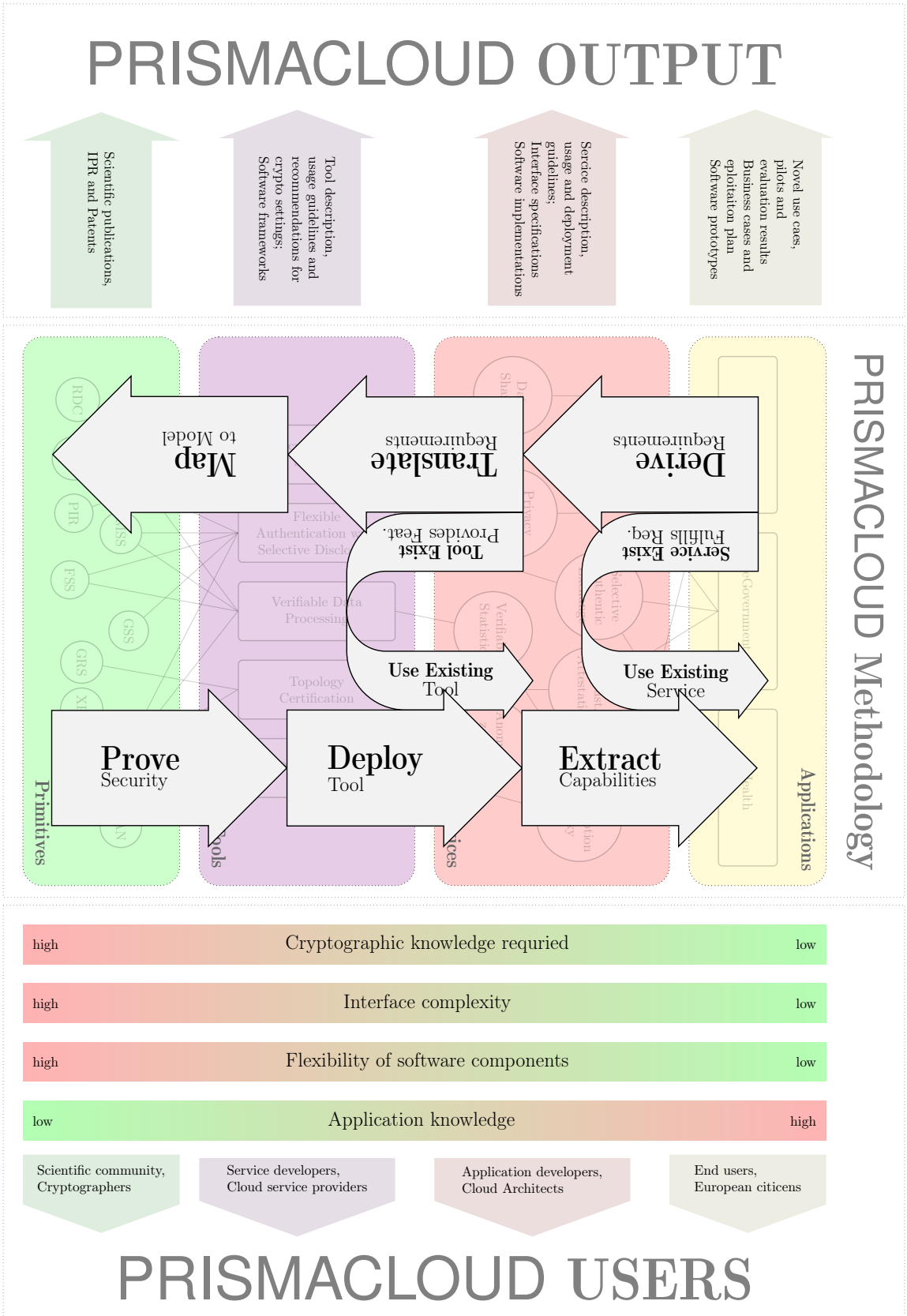


Figure 3: Overview of the PRISMACLOUD *CryptSDLC* Methodology.

all targeted at a single goal: the development of cloud services which are secure by design and leverage feasible cryptography.

Today's applications in emerging ICT domains like cloud computing or the Internet of Things (IoT) introduce many new challenges to cryptography. Cryptographic design in this domain is not an arms race but requires the design of primitives with improved or new functionality. In PRISMACLOUD we are tailoring and optimizing primitives to fulfill properties needed by upper layers. We believe that adding missing functionality at the lowest possible layer will positively impact the security of the application. If the functionality is not addressed at the lower layer but addressed by alternative security controls on higher layers it introduces additional overhead in complexity and administration which can also lower the security of the system.

To combine the primitives in the right way and also configure them accordingly, the tools layer was introduced. A tool is an abstract concept which resembles a typical basic application as good as possible. However, tools are still designed in a formal way to guarantee the security properties but are still independent from implementation. A tool has a feature set and comes with according usage and configuration guidelines. During service development the features will be mapped against the requirements and therefore define an instantiation of the tool for the specific service. Typically, not all features can be used everywhere because they often imply additional computation or network load and they can also be contradictory, e.g., in the secure object storage tool proactive security is not possible with computational security modes (which would offer better computational and storage efficiency).

Beside being a concept and guideline, a tool is also accompanied with a software implementation of the core cryptographic functionality. These are libraries which are implemented by experts in cryptographic software design. Even if the formal concept is provable secure, software implementation of cryptographic protocols are error prone and hard to get right. They have to be correct as good as possible but also side channel resistant; the latter is an art in its own. Having a sound formal cryptographic design of basic functionality and good software implementations is a major step towards secure applications.

Building cryptographic services out of these tools is much easier than without this intermediate step. Tools encapsulate the inner workings of the cryptographic functionality and offer a defined software interface, e.g., the interface for malleable signature generation needs some additional information but the software calls can resemble those for classical signatures. Service design can thus be done by software developers needing only a limited level of knowledge in cryptography. Having the tools layer, a service can be built in less time by less specialized software developers. In a sense, the service is a way to deliver the tool to the cloud customers in a preconfigured way and in a specific deployment. In the design of the service, the components of the tool have to be mapped on the stakeholders roles. When implementing the service in software, the libraries can be configured adequately and included into the service logic. There are many potential services that can be build out of a tool depending on the targeted customers and business domains. To demonstrate this concept we developed two different service concepts based on the very same secure object storage tool underneath. The secure archiving service as well as the

data sharing service follow a completely different deployment philosophy and use complimentary features. However, they are both built from the same underlying secure object storage tool. This is only one example that demonstrates the flexibility of the developed architecture, it is also done for the tools Flexible Authentication with Selective Disclosure (FLEXAUTH) and Data Privacy (DATAPRIV) as shown in Figure 1. Separating tools and services in PRISMACLOUD guarantees the best possible reuse of work and allows formal modelling to the highest level possible with reasonable effort.

The application layer is covering the integration of services in applications with real end users. The application developer itself is typically the cloud customer when viewed from the services perspective. Here the PRISMACLOUD model seamlessly integrates with the traditional development model. PRISMACLOUD services should be easily used and integrated by cloud architects and their security benefits should be accessible in a understandable language, ideally also supported by strengthened service level agreements (SLAs).

2.4 A New Development Methodology

To fully leverage the benefits of the PRISMACLOUD architecture we also designed a tailored secure software development lifecycle. The methodology is called *Cryptographic Software Development Lifecycle (CryptSDLC)* and defines a way to traverse the layers in the architecture. Furthermore, it standardizes the special steps necessary when going from one layer to the other and aligns them to the general phases of classical SDL models.

The major steps of CryptSDLC have already been included in Figure 3 and are marked by the big white arrows in the middle. The major steps are *Derive, Translate and Map* from top to bottom and *Prove, Deploy and Extract* from bottom to top. The CryptoSDLC is based on conventional software development lifecycles, like Microsoft SDL, but introduces another dimension. It augments and details the requirements, design and development phases with additional steps specifically dealing with cryptographic design tasks.

In the requirements phase we define the following steps, which implement a top-down approach for cryptographic requirements gathering:

- **Derive Requirements:** Based on the requirements gathered we derive the most important ones for the core cryptographic service we want to use or build.
- **Translate Requirements:** The requirements are translated into a more formal language which can be understood and used by cryptographers to start their research and design.
- **Map to Model:** To trigger research on primitives and protocols the identified gaps on the tools layer have to be mapped to research goals in cryptography for specific primitives or protocols.

In the design and development phases we use a bottom-up approach and define following steps to go up in the PRISMACLOUD model:

- **Prove Security:** Tool should be built by formal methods used in cryptography as good as possible. The goal is to support features by the definition of provably secure

protocols.

- **Deploy Tool:** The components of a tool are arranged in the service architecture in a way such that by reasoning it gets clear how the features of the tool translate the security requirements fulfilled by the service.
- **Extract Capabilities:** Based on the features of the tool and the deployment model specific service properties called capabilities can be extracted. They are exposed as an additional property to the upper application layer.

Although we define this holistic approach going down to the lowest layer, the reuse of existing work is a major goal of the whole process. In all layers the step down to lower layers is only performed if the requirements cannot be already achieved with available solutions.

Reuse is particularly forced by following policy:

- *Application layer:* Only develop a dedicated service if the application requirements cannot be fulfilled otherwise.
- *Service layer:* If possible, develop the new service on the basis of existing templates by only adding missing features supported by the underlying tool. Only develop on the lower tool layer, if needed features are not already provided by tools.
- *Tool layer:* Only develop a new tool if given tools cannot provide the required features or an existing cannot be extended with the required feature. For new designs, always do a state of the art analysis if missing features can be provided by existing research results or specify the gap if not. Then trigger research activities on the primitive layer.

CryptSDL has been implemented and successfully tested within PRISMACLOUD up to the service level; details can be found in the subsequent sections of this document.

2.5 Design Patterns for Inter Domain Communication

PRISMACLOUD as a project for the development, design, implementation, and feasibility demonstration of novel secure and privacy aware cloud services employs scientists and specialists of many domains in all its layers of architectural abstraction. The following bullets list for each layer, which group of individuals needs expertise on that particular layer–*plus at least in the adjoining layer above and below (if there is a layer above or below)* during the development process. For example, a tool designer with main expertise in the *Tools Layer* needs knowledge of the capabilities of the cryptographic primitives developed and configured in the *primitives layer*, as well as of the requirements postulated by the experts of the *Services Layer*.

Which group of individuals needs expertise in a particular layer:

- **Primitives Layer:**
  - cryptographers
- **Tools Layer:**
  - tool designers (specialised software engineers)

- **Services Layer:**
  - service designers
  - usability and HCI experts
  - cloud service providers and sub-providers (GDPR: ' controllers' and 'processors'[3])
- **Applications Layer:**
  - business model developers
  - general domain experts

- **On several or all layers:**
  - project communicators
  - IT security specialists

Also during end user application and service provisioning, which is the scope of the following bullets, cloud providers (or in terms of the GDPR cloud controllers or processors) need not only have expertise in the services layer, but also in the Tools and the adjoining Applications Layer. But note, that during provisioning and (end-)use no expertise is required on the primitives layer. This is due to a deliberate design choice of the PRISMACLOUD architecture–the encapsulation of cryptographic functionality in the tools, thus hiding the cryptographic details, and all the caveats, implications, and sources of error from service developers.

During end user application and service provisioning:

- **Primitives Layer:**
  - n/a
- **Tools Layer:**
  - n/a
- **Services Layer:**
  - cloud sub-providers (GDPR: 'processors[4]')
  - cloud providers (GDPR: 'controllers') [5]
- **Applications Layer:**
  - individual or organisation (GDPR: data subject)
  - individual or organisational cloud customer
  - GDPR: recipient (to whom data is disclosed)
  - GDPR: third parties (others, authorised to process data)[6]

- **On several or all layers:**
  - cloud auditor

[3]This is not exactly an 1:1 relation as cloud providers are entities that "determine the purposes and means of the processing of personal data" ([Eur16], Art. 4, Par. 7 and may be the processors themselves. Processors are defined in the GDPR as entities "which processes personal data on behalf of the controller" (ibid. Par 8)

[4]ibid.

[5]In its conceptual framework [a4c14], the FP7 A4Cloud project provides additional two sub-categories in the cloud provision chain, being mainly active in a trading commerce: cloud carriers and cloud brokers.

[6]For all the GDPR definitions, see [Eur16] Article 4

– GDPR: 'supervisory authorities'

PRISMACLOUD develops specific communication tools to support the layered development process governed by the Cryptographic Software Development Lifecycle (CryptSDLC), as well as to support the diffusion of PRISMACLOUD paradigms and capabilities among prospective providers and end users. These tools are *cloud security and privacy patterns* and *HCI–human computer interaction–patterns*.

Both cloud security and privacy patterns and HCI patterns are a kind of design patterns, which are used to codify expert knowledge and requirements within a specific scope in a way that the information remains accessible across domains of involved actors. The main idea is that a design pattern shall "describe(s) a problem which occurs over and over again (...) and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over (...)" [AIS77]. This is done by describing the (empirical) background of the pattern, i.e. the "problem", and giving instructions for the "solution" in natural language in a framework of categories.

The concept was invented in Berkeley, CA., in the 1970s for application in architectural design (architecture for towns, buildings, construction [AIS77][7]), and has later on been modified for application in several information technology sub domains. The first application of design patterns in information technologies was in software architecture in the 1990s when object oriented design and re-usability required efficient communication of complex issues across different domains of involved people [GHJV94]. Later on, the concept was used for the specification of security and privacy concerns in security and privacy patterns [SFBH+06, DG13], as well as for human computer interaction aspects in HCI patterns [FHKP+11]. Since several years, there exist collections and catalogues of cloud security and privacy patterns specifically for modelling threats and solutions in the cloud context.

In PRISMACLOUD we initially proposed a set of nine cloud security and privacy patterns in D2.2 "Domain Independent Generic Security Models" of M12 (End of Jan, 2016). These patterns covered approximately the cases we initially at project begin had devised for the cryptographic functionalities we wanted to cover in PRISMACLOUD. We used these original patterns as common reference and "communication means" between tools developers, service developers, and application developers in the actual process of developing the PRISMACLOUD architecture and the CryptSDLC method.

In D3.6 "Secure Cloud Usage for End Users, Progress Report" (delivered in M24), we developed a new method for codifying properties of our tools and services in cloud security and privacy patterns (new is the dedicated application of patterns for the documentation of specific tools and services, contrary to a more generic, implementation independent approach, usually prevalent in design patterns). Now that the tools and the services are "feature frozen" and specified in detail (with the current M30, end of July, 2017, deliverables round) the cloud security patterns are being iteratively further developed. This is done in task 3.4 "Secure cloud usage for end-users" of the end user and business

[7] The editor yet again has to point out to the interested reader that the entire, most interesting book is available for free download at https://archive.org/details/APatternLanguage

deployment work package WP3 where we will be using these patterns in order to derive structured guidance for prospective end users of the proposed cloud services. We intend to deliver and publish a full set of patterns by project end in D3.7 "Secure Cloud Usage for End Users" (M42, End of July, 2018).

In D3.2 "HCI Guidelines" we are present three HCI patterns[8] and intend to present a full set for design contentions in the PRISMACLOUD prototypes by the end of the project.

[8]HCI.P1 Digital Signature Visualisation, HCI.P2 Informed Consent, and HCI.P3 Stencil for Digital Document Redaction

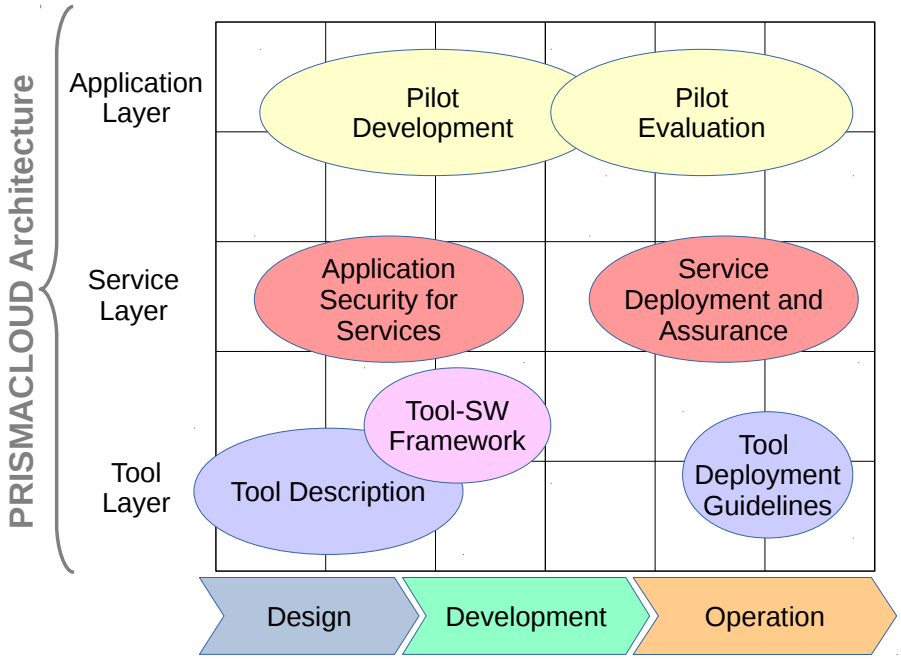


Figure 4: Secure service development artifacts.

3 Guidelines for Composing Secure Services

In this section we are detailing the development process envisaged to implement the methodology presented in the previous section. We are describing the various approaches taken on the different layers of the architecture and how they interface with each other. For that purpose we define a concrete set of documentation artifacts which is also used in the project to document the development of the PRISMACLOUD services.

3.1 Overview

Based on the architecture and the development methodology presented in Section 2, this chapter proposes recommendations for the development process. It serves as a guideline for the implementation of the proposed CryptSDLC methodology and defines necessary documentation artifacts to be generated during service development. For PRISMACLOUD in particular, the guidelines also serve as a project standard which must be used by all partner involved in service and application development.

This guideline covers multiple disciplines and intends to be compatible with relevant standards within their respective fields. Furthermore, it tries to establish a link between the different fields as well as interfaces. We are using accepted standards in cryptographic design on the primitives and tools layers, and security-by-design concepts on the service and application layers.

Besides considering CryptSDLC, the guidelines are following well known approaches from

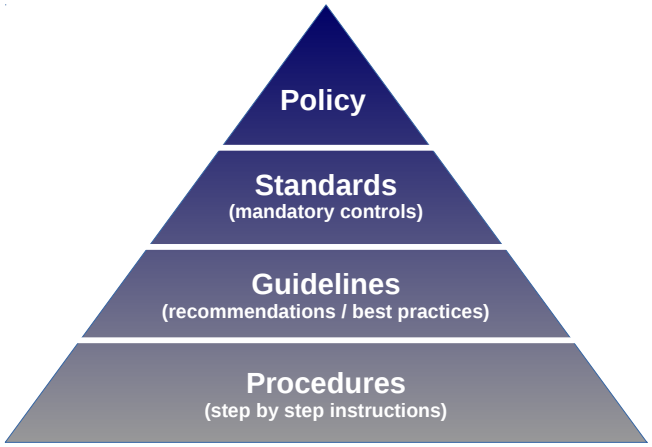


Figure 5: IT security document hierarchy.

software development life cycle management and security by design methodologies. Also recently established concepts like the CloudSDL and assurance framework developed in SECCRIT are also integrated. Finally, we adopted holistic approaches which not only considering the development phase but also includes operational aspects relevant for the production phase. The coverage of the guidelines is shown in Figure 4. The Circle mark the artifacts delivered in the project and the location on the x-axis mark the relevance for the corresponding stage in the lifecycle. Furthermore, the y-axis represents the layer the documents belong to as well as the coloring is aligned with the layer in the PRISMACLOUD architecture.

The final version of this report will contain full service documentation based on the guidelines presented. However, the relevant documentation for the tool layer will be covered within the WP5 deliverables. The current version of the guidelines defines the following key properties to be covered in the service's documentation:

- Description of the service
- An operational view and service model
- Description of benefits and design requirements
- Definition of the attack surface and completion of a risk assessment
- Definition of security controls to be implemented
- Description of operational considerations
- Definition of a monitoring and assurance profile

All steps together cover the complete life-cycle for service development and further integrate state of the art methodologies for application security and operational security. In the following we review the adopted methodologies and give recommendations how they should be applied in the service development process.

Within this document we are using definitions derived from common usage within IT security frameworks, in this case from the ISO27000 security standard for information

security management (ISMS). See Figure 5 for an overview of the commonly used elements in ISO27000:

- *Policies* are the top tier of formalized security documents. These high-level documents offer a general statement about the organization's assets and what level of protection they mandate. In PRISMACLOUD we are not dealing with this level.
- *Standards* are much more specific than policies and are tactical documents as they lay out specific steps or processes required to meet a certain requirement. In PRISMACLOUD we are setting a project-wide development standard which points to key procedures in our methodology documented in the service section of this document.
- A *guideline* is a recommendation or suggestion of how things should be done and point to certain statements in policies or procedures. It is meant to be flexible to allow for customization for individual situations. In this section we are defining the PRISMACLOUD guideline for secure service development, we recommend techniques for different levels of the architecture and show how they fit together and build the PRISMACLOUD development methodology.
- A *procedure* is the most specific of security documents. A procedure is a detailed, in-depth, step-by-step build specification. We develop some specific procedures to be applied in the development of the services. Especially the unified documentation of services and tools is based on procedures defined to be used by all partners throughout the project.

3.2 Requirements Engineering

The requirements engineering phase starts with the beginning of the project at the highest abstraction level which is the application level in our case. In PRISMACLOUD we analyzed applications and compiled a catalog of requirements associated with them as documented in D2.3.

From the requirements and the functionality described in use cases we extracted service functionalities which ideally resemble the generic idea behind the application and encapsulate the core functionality in a domain independent way. New requirements for the service have been derived from the application requirements specifically as a subset of the overall set.

The requirements which are relevant for the underlying cryptography have been extracted and translated into a more formal language which is used to model the environment in cryptography. Typical examples are the specification of the security properties, adversary models (e.g., computational or unbounded), network models (e.g., synchronous or asynchronous).

During the cryptographic design the models are further mapped to individual models for the different primitives and protocols used. Furthermore, it will be checked if solutions for the aspired functionality exists or if primitives have to be adapted to support the full set of functionality modeled at the tool layer. The overall process is part of the methodology shown in Figure 3.

Requirements engineering within Prismacloud. From our current experiences we recommend to adopt a process conducting the following steps:

- Decompose the application and derive a generic domain independent service with a well-defined set of associated requirements.
- Translate the requirements for cryptographic design to one of the more formal languages known from cryptography
- Map the requirements to specific primitives and protocols used to compose the tool.

3.3 From Primitives to Tools

As discussed in detail in Section 2, on a high level, *primitives* in PRISMACLOUD realize very basic cryptographic functionalities, while *tools* are higher-level concepts, which are solving problems on an abstract and generic, yet already practically relevant level. Due to the cryptographic nature of the tools developed in PRISMACLOUD, it is of prime importance to have a profound analysis and sound security proofs for them. In the following we will thus discuss the two main approaches being used for composing primitives to tools, and explain where they might be used within the project.

3.3.1 Universal Composability

Often when analyzing the security of cryptographic protocol one considers the analyzed protocol as a standalone application, sometimes even assuming that only a single instance of the protocol is executed at a time. However, this clearly does not properly model reality, where many instances of potentially different protocols are executed concurrently, potentially interacting with each other. Unfortunately, security properties proven in the standalone setting in general are not retained under protocol composition. Overcoming this problem and constructing security models where security is retained under protocol composition is the subject of various frameworks found in the literature, e.g., Canetti, Hirt and Maurer, Pfitzmann et al., and Küsters et al.[Can01, CCK+06, CDPW07, Küs, KT13, MR11, Mau11, PW00].

Even though being different in their details, all these frameworks follow the same high-level idea: In a first step, a protocol designer specifies the ideal behavior of the cryptographic task he wants to realize in terms of an *ideal functionality* $\mathcal{F}$. This ideal functionality takes inputs from all involved parties, performs some computations, and returns outputs to all parties. Apart from this, no communication between the protocol participants takes place, i.e., $\mathcal{F}$ can be thought of as a trusted party realizing the required functionality, which is secure by definition. A *real protocol* $\mathcal{P}$ now realizes (or: *emulates*) $\mathcal{F}$ if every attack $\mathcal{P}$ could also be mounted on $\mathcal{F}$, which cannot be possible because of the assumed security of $\mathcal{F}$. Slightly more formal, $\mathcal{P}$ emulates $\mathcal{F}$ if in any context any attack in the real world (i.e., on $\mathcal{P}$) can also be efficiently simulated in the ideal world (i.e., on $\mathcal{F}$).

The main results of all universal composability frameworks now is a very strong composition theorem. Namely, it says that if $\mathcal{P}$ emulates $\mathcal{F}$, then $\mathcal{F}$ can be replaced by $\mathcal{P}$ in

arbitrary higher-level protocols without affecting the security of the overall construction. That is, let $\mathcal{F}$ be an ideal (high-level) functionality which is emulated by $\mathcal{P}$, which itself uses another ideal (low-level) functionality $\mathcal{G}$ as a subroutine, e.g., for secure data transfer or encryption. Let furthermore $\mathcal{Q}$ be a protocol which emulates the low-level functionality $\mathcal{G}$. Then $\mathcal{P}^{\mathcal{Q}/\mathcal{G}}$ emulates $\mathcal{F}$, where $\mathcal{P}^{\mathcal{Q}/\mathcal{G}}$ is the protocol $\mathcal{P}$ where every invocation of the ideal functionality $\mathcal{G}$ is replaced by an invocation of its realization $\mathcal{Q}$.

Advantages. Constructing tools in universal composability frameworks yields a number of favorable properties.

First, we obtain very strong security guarantees as discussed before. In particular, for certain application areas meaningful security can hardly be proven in other ways. For instance, this is the case for cryptographic constructions involving human-memorizable passwords, as users tend to share passwords, use related passwords, or to leak information about their passwords, and thus such constructions must not be analyzed as standalone applications.

Furthermore, because of the composition theorem, universal composability frameworks allow for a modular security analysis of complex primitives and tools. This is because in the construction and security proofs of higher-level protocols, one can use ideal functionalities as subroutines, and later simply replace them by secure realizations. Furthermore, because of this modularity it is straightforward to replace certain parts of a protocol by, e.g., more efficient constructions.

Also related to this modularity, security proofs become less monolithic and more compact as well, which (at least in theory) might make it easier to verify the formal soundness of a proposed construction.

Drawbacks. Besides the obvious advantages, proving constructions secure in a UC framework unfortunately also introduces a number of drawbacks.

First, UC-security usually comes at a high computational price, rendering many constructions practically unusable, in particular when they are supposed to be executed on low-cost devices with low computational capacities.

Furthermore, even though universal composability has gained significant attention from the research community for more than a decade by now, many existing primitives have not (yet) been formulated in any of the existing frameworks. Therefore, designing UC-secure tools might require extensive effort also on the primitives level in order to get efficient building blocks that can be composed in a modular way.

Finally, virtually all of the existing frameworks are either to restrictive in their expressiveness to model all functionalities one needs, or they suffer from subtle technical flaws spoiling the high security guarantees. Even worse, those models that are considered to be formally sound and which are sufficiently generic to model all relevant functionalities do not provide sufficient support to protocol designers when defining ideal functionalities and real protocols. As a result, every protocol designer has to fix certain aspects of the frame-

work (e.g., the way that corruption of protocol participants works) himself, resulting in potentially incompatible primitives which cannot be used to jointly construct higher level tools. This issue is currently addressed by ongoing research of PRISMACLOUD [CEK+16].

Universal Composability within Prismacloud. We believe that certain benefits coming from UC modeling are often actually not required in practice. For instance, one usually chooses a fixed instantiation of a primitive and does not replace subcomponents of tools modularly later on. Because of this, and because of the computational overhead of UC-security, we will not design and prove all primitives and tools in UC-frameworks. However, we might use such a modeling for primitives and tools which are either used as building blocks in multiple higher-level constructions, or where the computational costs (and overhead) are acceptable in practice.

3.3.2 Direct Construction of High-Level Primitives

The other main approach for constructing complex primitives and tools and proving them secure are direct (or ad hoc) constructions. That is, one defines a set of experiments covering the security properties one wants to realize with the given functionality, e.g., unforgeability of signatures, confidentiality against a defined class of adversaries for encryption, etc. One then specifies concrete instantiations of algorithms and proves that those algorithms indeed satisfy those security definitions.

Advantages. The main advantage of ad hoc constructions is an increased efficiency compared to universally composable constructions. That is, when aiming for practically efficient schemes that can be deployed in the real world, it is often hard if not impossible to come up with UC-secure instantiations, while this is indeed possible for ad hoc schemes.

Drawbacks. Direct constructions based on security experiments often suffer from several severe drawbacks. One of the most fundamental drawbacks is that security is typically not retained under concurrent composition of the resulting protocols, often not even when the protocol is only composed with other instances of the same protocol, let alone other, potentially insecure, protocols.

A second drawback of UC secure constructions is that they often do not allow for protocols with very high modularity. That is, in general it is hard to replace certain sub-components of a tool without having to revise the entire security proof.

Finally, and related to the previous issue, security proofs tend to be monolithic and non-modular for ad-hoc constructions. This is different to UC-secure constructions, where lower-level primitives can be used as black-box building blocks by using their corresponding ideal functionality in the security proof.

Direct Constructions within Prismacloud. Within PRISMACLOUD, we will design most of our tools and primitives on an ad-hoc basis. This is mainly because PRISMACLOUD is aiming at developing practically usable schemes, and thus the higher efficiency of direct construction is required. However, we will try to mitigate the drawbacks of direct constructions wherever possible. For instance, we will at least partially mitigate the problem of security problems under protocol compositions by modeling, e.g., concurrent self composition in the definition of the security experiments by allowing the adversary to initiate arbitrarily interleaved invocations of the designed protocol. Also, we will address the reduced modularity by aiming for generic constructions wherever possible. That is, we will try to work with abstract definitions of, e.g., encryption or signature schemes in order to not nail down the concrete scheme to be used. This will increase the flexibility of our tools by making them re-usable in different contexts. As a positive side-effect, this will also make our security proofs a bit less monolithic, as sub-components do not need to be considered in the security proof but can be used in a similarly idealized fashion as in the UC approach.

Among others, we have already used this approach for constructing a framework for attribute based credentials (ABCs) [CKL+15], for designing an auditable distributed storage system as a fundamental part of the secure object storage tool [DKLT16], and for designing a data sharing platform with selective disclosure for confidential data [DKS16].

3.4 From Tools to Services

The tool abstraction we have introduced in PRISMACLOUD is a key concept which greatly simplifies the development of secure services. Tools are an abstract description and provide core functionality as well as additional features which can optionally be added to the system. Each feature come with sound cryptographic realizations underneath to be used in the implementation. Furthermore, the tools are designed in mathematical rigorous models and based on the ideas of provable security as described in the previous section.

When we speak of formal modeling or methods, we mean that we cryptographically model a scenario or a use-case in terms of well known theoretical tools from the domain of provable security. In particular, this means that we formally model it either using composability frameworks (such as the universal composability framework) or using a direct approach by carefully modeling the capabilities of an adversary. In both approaches, the provided constructions used by us come with a rigorous security analysis, i.e., a proof of security. This means that we come up with constructions that provably satisfy our posed security requirements and thus give us strong confidence in their security guarantees.

The tool layer is part of the PRISMACLOUD development methodology and was introduced to improve the results and speed in secure service development. It is also intended to maximizes the reuse of existing work on the cryptographic layer. According to the methodology, the service requirements are matched against the feature set provided by the tools to achieve the desired goals. In the ideal scenario, no tool development has to be triggered and service can be built right away from existing tools. However, if the respective tool does not provide the right features, the PRISMACLOUD methodology suggests trying

to add it at the lowest layer. This means that the development cycle going down to the primitives layer is triggered and if possible the feature is added to the tool by the best formal methods available. Once added, it can be reused for all further service designs and if the feature cannot be added, additional measures and security controls have to be used at the higher layers to fulfill the requirements.

Nevertheless, the process of generating a service out of a tool has to cover all additional steps not covered by the tools but needed for real world applications. In particular, the following steps are necessary to design a service out of a tool:

1. Specify a service and deployment plan as well as stakeholders
2. Identify major components in the service and sketch their main functionality
3. Embed the tool components within the service components
4. Map the requirements to the features provided by the tool
5. Generate a software architecture and specification
6. Implement software development lifecycles with integrated security
7. Propose operational guidelines like an assurance model to support production phase

Using the proposed methodology should lead to services which are secure by design and built on cryptographically sound composition of primitives and protocols, i.e., in a provable way in the best case. Especially, after embedding the components of the tool into the components of the service—the deployment step—and considering all additional guidelines specified by the tool, e.g. "communication between server and dealer must be private and authentic", we can reason about the service to be a secure instantiation of the tool. Compared to ad-hoc integration of cryptography into a service, the additional detour over the tools introduces some additional work for the first time, but later on supports wide reuse without the need for individual analysis of specific services.

To demonstrate this advantage we show the flexibility of the tool concept in Figure 6, where we design two different service concepts based on the same secure object storage tool. The two services—secure archiving and data sharing—are aimed at different use cases with different trust and business models while being based upon the same tool. They also use different, sometimes even contradictory, features from the tools.

Naturally, when building a piece of software there are additional aspects to consider apart from using correct algorithms, i.e., correct cryptography in our case. It faces all challenges known from secure software development and all state-of-the-art processes and methodologies for SDL shall also be applied during the PRISMACLOUD service development. Tools also offer an additional benefit: increased development speed and improved security through the secure and efficient software software implementations of core cryptographic functions provided with the tools.

Nevertheless, a complete service is comprised of many dedicated software components running within the cloud infrastructures. Even worse, operational aspects have to be considered and defined to fully support a cloud service life cycle thus mandating integration into operational processes. In summary, although the tool concept greatly facilitates the service development process, all SDL documentation must be heeded to foster quick

adoption of project results on the service level.

Service Development within Prismacloud.

In PRISMACLOUD we are developing a portfolio of 8 services out of the 5 tools developed in the project according to the presented methodology. In particular, the following documentation is generated for the PRISMACLOUD services:

- Overall idea and functionality of the service including:
  - A summary of key features with respect to novel security and privacy properties
  - A usage model and stakeholder analysis explaining the major roles involved
- Operational view and service model including:
  - Abstract interaction dynamics
  - Definition of provider/consumer scope of control
- Major functional requirements are described and mapped against the provided features of the tool
- Guidance for application development based on SDL
  - Results from the attack surface analysis
  - Results form the threat modeling process
  - Threat assessment and mitigation analysis
  - Validation plan
- Definition of an assurance model and security-monitoring profile

3.5 Design, Development and Deployment

A state-of-the-art software development processes for service development is essential for achieving secure services. Even if a tool's functionality is implemented correctly, its security is undermined if the enveloping software has vulnerabilities. In PRISMACLOUD we adopted the well established Software Development Lifecycle (SDL) from Microsoft which is compatible with the emerging ISO27034 series of standards for application security. Furthermore, we adopt parts of the cloud specific development life cycle extension developed by the SECCRIT project: the hierarchical and iterative security requirements elicitation process from CloudSDL as well as their assurance monitoring approach during operation. Finally, we talk about the relation to the security and privacy design methodology also used in the project and give some general technical deployment recommendations.

3.5.1 CloudSDL from SECCRIT

An overview of the CloudSDL is shown in Figure 7 and we are quickly reviewing the six stages defined for the secure software development phase of cloud based applications or services.

1. *High level security objectives analysis:* This preliminary step consolidates high level business objectives with security related standards, best practices and guidelines to set the initial security objectives for secure service design, development and deployment.

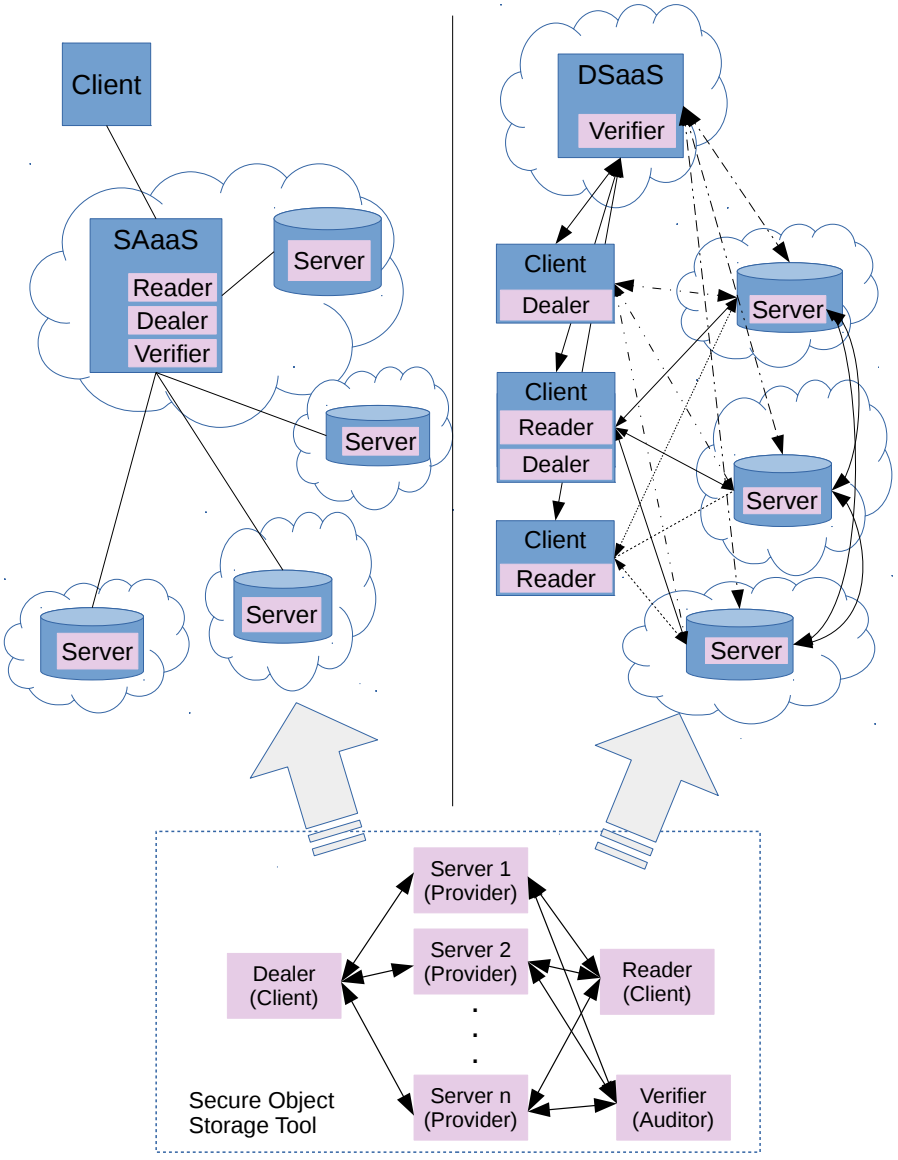


Figure 6: Illustration of different deployment models for storage tool.

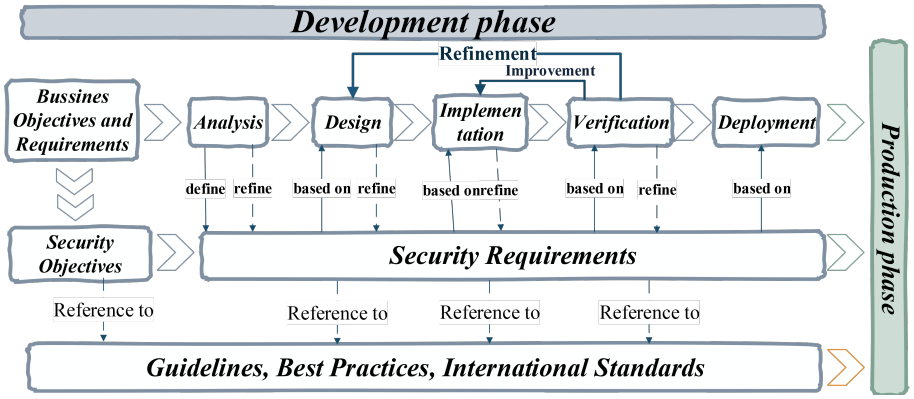


Figure 7: Cloud-service development life-cycle (CloudSDL.)

2. *Analysis:* During this step the service is analyzed for compatibility with requirement coming from cloud environments. Furthermore, the initial set of security requirements is specified and potential threats to the particular use case are identified. Ideally, if security requirements for the IT service are predefined, they are taken into account and, if needed, adjusted to the circumstances occurring in each subsequent stage.
3. *Design:* In the design step, the software architecture for the developed IT service is designed in line with the security requirements specified in the analysis step. If required, refinements of the security requirements are performed to align the security requirements towards the particular use case.
4. *Implementation:* During implementation standard software development methodologies are applied. It is important, that the used methodologies support the security by design philosophy and therefore make use the developed requirements.
5. *Verification:* In this step, the software is tested against a predefined set of security requirements before being deployed or migrated in to the cloud. Additionally, before the application is deployed in the cloud the readiness of the organization shall be verified (e.g. special disaster recovery strategies, trainings, or revisions of SLAs might be required).
6. *Deployment:* In the final step of the development phase the IT service is deployed to the cloud environment by taking into account the security requirements related to platform configuration.

This approach considers security from the very fist step and integrates security requirements engineering throughout all phases. Additionally, these requirements can also be used to support the selection of suitable cloud providers or to extract processes and monitoring information for the production phase.

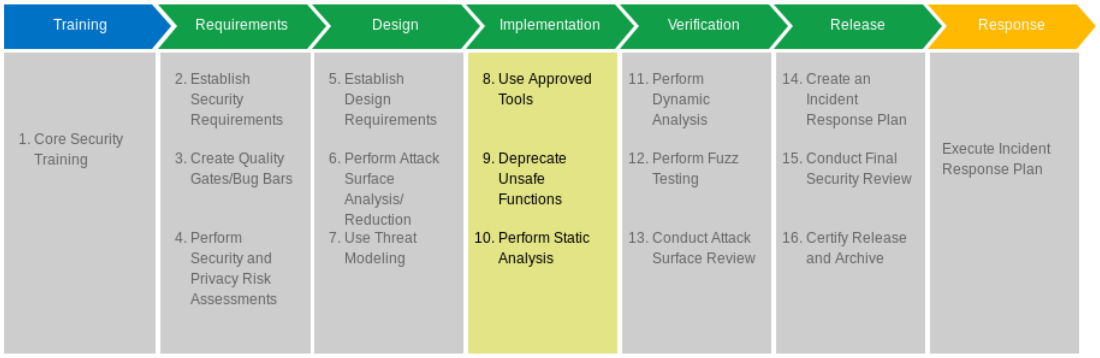
CloudSDL within Prismacloud. In PRISMACLOUD we follow the process as good as possible but without considering the full spectrum of processes, i.e., we omit the steps dealing with organizational aspects as well as legal and compliance issues. However, we adopt the iterative and hierarchical security requirements engineering process defined and provide following information:

- Application level security analysis including business objectives
- Derivation of service level security requirements
- Provide design level support for security based on cryptography if possible
- Base the implementation and verification on standard Microsoft SDL process
- Support service deployment and usage with additional security capabilities and advertise them via service description languages

3.5.2 Microsoft SDL

The cost of mitigating software vulnerabilities is directly proportionate with their detection point of time. The earlier vulnerabilities are recognized and mitigation procedures initiated the lower the overall impact upon project costs. Neglecting security during early development stages, such as Requirements Analysis or Design, yields products with systematic security problems, e.g., security problems based upon badly chosen architecture decisions.

To prevent costly corrective measures Microsoft's Security Development Life-cycle focuses upon early detection of potential security problems.



PRISMACLOUD is a research project consisting of multiple partners. Produced artefacts include multiple software prototypes fulfilling partner-specific use-cases. This special constellation of multiple partners working together on a final prototype lead to some modifications of the original SDL:

Phase	Practise	Not applicable because
Training	Core Security Training	All personnel is provided by the different partners and is already educated in secure programming
Release	Create Incident Response Plan	Research ends with prototype implementation, no long-term system management is included
	Certify Release and Archive	Our software is delivered "as-a-service". While it has to fulfil our internal test-cases no public software releases are created thus no release can be certified.
Response	Execute Incident Response Plan	Research ends with prototype implementation, no long-term system management is included

Other parts of the SDL have already been created by prior deliverables, including the security requirements needed for finishing the requirements phase. Traditional software projects perform an initial security and privacy assessment in order to determine the level of privacy and security actually needed by the software project. Based upon this selection, fitting quality gates or bug bars—bugs that are classified as "show-stoppers"—are defined. As we are a security- and privacy-focused research project, all our security and privacy requirements are deemed to be show-stoppers thus removing the need for further classification.

The project engineers heed the best practices detailed within the Design, Implementation and Verification Phases.

During the Design-Phase the gathered security requirements are applied to a planned service implementation. Not all high-level security requirements might be fitting, thus in an initial step design requirements for the concrete service are elected. A common problem with services are overarching interfaces that provide more functionality that is actually needed. This additional functionality leads to an increased attack surface and increases the chances for security vulnerabilities and long-term maintenance costs. To prevent this, all exported interfaces are subject to an Attack Surface Analysis and Reduction. The generated minimal-interface and design requirements are input for threat modeling utilizing the STRIDE methodology. The output of this modeling are concrete security threats for our planed concrete service.

During the Attack Surface analysis all paths for data/commands into or out of the application well as all valuable data used within the application (including keys, personal data, PII, intellectual properties) are documented. This includes protective measures and codes that are placed upon that path—firewalls, authentication, authorization, etc.

STRIDE aids the threat modeling process by providing an usable categorization of potential threats into the following classes. **Spoofing identity** allows an attacker illegally accesses and utilize identity information to gain access to other services. **Tampering with data** includes malicious altering of data. **Repudiation** threats are associated with users denying performing malicious actions. **Information Disclosure** includes the board

field of data leaks. **Denial of Service** threats describes any means possible for making an service inaccessible. **Elevation of Privilege** allows an attacker to execute operations with higher user/access rights than her authorization originally entailed.

During a typical threat-analysis meeting multiple dozen of threats can be identified and grouped according to their membership of the different classes.

During the Implementation Phase the tools (akin to algorithms or building blocks) are combined into services. The SDL focuses upon providing a list of known good development practises or tools (not to be confused with tools in the sense of algorithm) as well as on static verification of the written source code. All project partners are allowed their own tool selection. At AIT we have selected the Java Programming language and the pre-selection contains Language Feature Level (Java 8.0) as well as used libraries (application servers as well as client libraries). This limitation was deemed necessary as "using components with known vulnerabilities" was recently added to the OWASP Top 10 Vulnerabilities. In addition, some programming language functionalities have been deemed as inherently insecure and not allowed for development: this includes native method invocation (as it circumvents Java's memory-safety guarantees) and direct database-access without using an database mapper. Again, this selection was used to prevent common security vulnerabilities – Injection Attacks (prevented by the Database Mapper) are currently Number 1 at the OWASP Top 10. All written source code is subject to static doe analysis. For this, the "findbugs" program has been integrated into the project's build chain and is automatically performed during every build.

After the implementation phase comes verification. No source code is ever released into the prototype testbed without prior verification. In contrast to the code validation all code is tested against running systems. All network code is subject to network vulnerability scanners (AIT is using a combination of nmap, openvas, sqlmap and arachni). No security finding of level "Medium" or higher shall be allowed. In addition Fuzz testing is performed against all network interfaces. As external tests are inherently black-box based the result of the tests is not only a list of potential vulnerabilities but also an external view of exported interfaces. This external view describes the actual attack surface of a service and is thus compared to the initial attack surface analysis of the design phase. The tested attack surface must be the same or smaller than the designed attack surface.

One best-practice normally performed during the release-phase is still performed during our secure software development life cycle: a final security review is performed after all sub-components and services have been deployed within our prototype testbed. Together with functional tests this is the final verification outcome of the PRISMACLOUD research project.

SDL within Prismacloud. In PRISMACLOUD we apply SDL and provide full documentation of following steps:

- Attack Surface Analyis
- Threat modeling with STRIDE
- Risk assessment
- Risk mitigation plan

3.5.3 SECCRIT Assurance Monitoring

SECCRIT proposes a two-phase secure cloud service life-cycle that integrates requirements engineering and iterative refinement with respect to security, through each stage of both phases. The first phase is called *Development phase* and covers the sequential set of steps where a service is being designed and developed. Secondly, the *Production phase* is where a deployed service is validated against those security requirements that have been defined in development phase. As mentioned before, consistent integration of security concerns throughout each step of both phases is vital for designing and operating secure systems and services. Therefore, in each phase of SECCRIT's proposed life-cycle the security requirements engineering process is aligned to the particular step (design, development, maintenance, assessment or monitoring), standards, best practices, or guidelines [fSC05, NA12].

We already discussed the CloudSDL approach for the service development phase in Section 3.5.1. In the following we will discuss assurance monitoring which is used in the production phase. An overview of the approach developed in SECCRIT is shown in Figure 8.

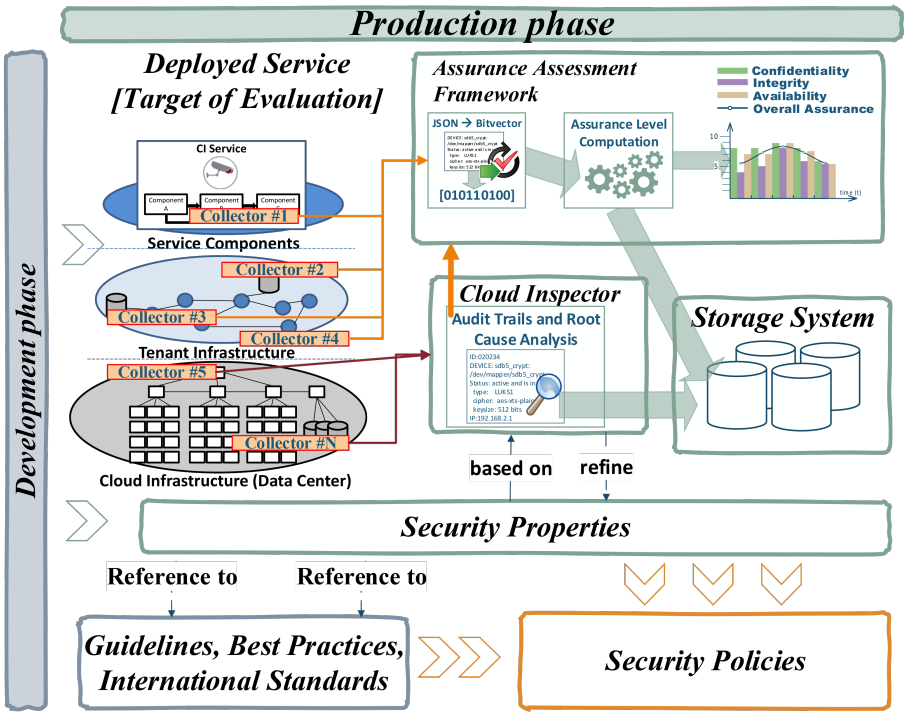


Figure 8: Production phase of secure cloud-service development life-cycle

To perform consistent security assessment the framework [HTL+14] requires a predefined set of security properties, used to validate security across individual components of interest. These security properties need to be concisely defined as the validation is based upon their condition. Therefore, we use the security requirements from the development phase to

engineer the security properties for our validation process.

To acquire security related information across the infrastructure we use *Collectors* that harvest information, deliver it to assurance framework to compute the assurance level and classified across three assurance classes (confidentiality, integrity, availability). Additionally, both security requirements and properties are used to define policies for maintaining or assessing security.

CloudSDL and Assurance Monitoring within Prismacloud. In PRISMACLOUD we try to apply the assurance monitoring approach from SECCRIT where appropriate. In particular, we analyze the secure archiving service in more detail to explain the methodology. However, because a full analysis of all service is out of scope, for the remainder of the services we give general operational guidance where needed. Furthermore, we will give additional guidance on operational aspects of the services in WP8, where we will report lessons learned from pilot evaluation and verification.

3.5.4 Secure deployment

Nowadays, applications and services entail vast amount of dependencies (e.g., libraries, packages, runtime environments) that have to be in place for both deployment and production environments. This is currently the main obstacle when it comes to portability. To enforce a higher degree of portability and interoperability we need a solution that will offer lightweight distribution of packaged applications. Currently, this portability is achieved through virtual machines that combine the applications or services together with their dependencies. Unfortunately, virtual machines introduce a high performance, complexity and performance overhead. A possible solution to this are containers that run on the top of the host operating system. Containers are virtualization technology, deployed on top of a shared operating system environment, used to isolate processes by controlling available resources and namespaces on which a service or a process is being provisioned. Basically, containers offer the ability to create customized isolated environments that are completely independent in terms of namespace from the host OS. Therefore, containers are quite similar to the standard visualization concepts being used nowadays such as virtual machines (VM). Although, containers and VMs are in essence similar, containers are designed to rapidly provision and scale software on on demand by minimizing resources consumption and offering high portability.

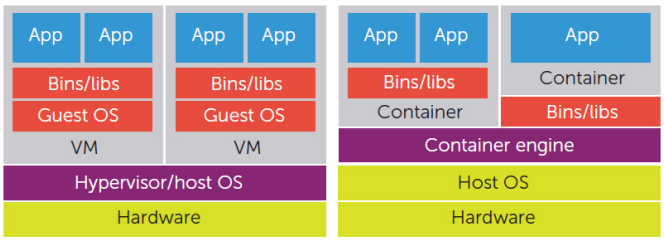


Figure 9: Virtualization architectures [Pah15]

The foundation for achieving fast deployment, portability and scalability lies in the lightweight characteristic of the container technology, namely the main difference with respect to the virtual machine concept. Figure 9, outlines the main differentiation between the virtual machines and containers. As we can see form the Figure 9, the container virtualization concept unlike the hypervisor virtualization does not require to boot the operating system, and even the binaries and libraries can be left out in some cases. The isolation of namespaces within containers offers us the ability to configure and install dependencies in a closed environment independently from the host OS. Therefore individual processes and services that are deployed in the container environment are migrated together with the container itself as a coherent service package. In addition, the isolation of services or its components results as increase of security by reducing the manoeuvring capabilities of a service. Within a single container diverse application, service, supporting libraries and dependencies, as we can see from the Figure 10, can be deployed. Therefore, containers ease the development, deployment and maintenance of applications and their dependencies, offering at the same time high reusability and portability.

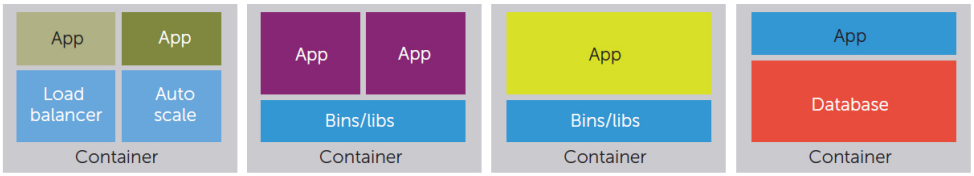


Figure 10: Virtualization architectures using Containers [Pah15]

Moreover, the flexibility offered by container environments gives us the ability to design and deploy composite service architectures; and gain more control of service components in terms of security by limiting the capability of a service to a single container. The current approaches are focused on defining policies, such as host-hardening standards, versioning and configuration management of applications and services hosted within a container[9]. When taking into the consideration security for designing composite services, we have to focus on following challenges:

1. workflow of both information and services
2. internal and external privileges and access control model
3. deployment environment and circumstances
4. containerization technology
5. support for common security controls
6. operations management and configuration governance

Secure deployment within Prismacloud. Within PRISMACLOUD we recommend the use of secure containerization technologies for service deployment, preferable based on Open Source Software with broad provider support, e.g., Docker or LXC/LXD-based

[9]Container security: Twistlock - https://www.twistlock.com/,
Docker - https://docs.docker.com/engine/security/security/,
Veritas - NetBackup CloudStore Service Container - https://www.veritas.com/support/en_US/article.000078835

solutions that allow for best interoperability, scalability and portability with least possible memory and storage overhead.

3.6 Standard Identity Provisioning and Management

Identity provisioning and management is an important task in distributed systems—which our pilot applications will definitely be. Therefore, we foresee the possibility to use a central identity service which is also capable to provide hard electronic identities in a standardized and interoperable way. Connecting the PRISMACLOUD pilot to the European eID system demonstrates the seamless integration of our services with efforts going on in the area of electronic identification and trust services for electronic transactions (eIDAS) and the digital single market (DSM).

Especially for cloud services, identity management is becoming more and more important. The challenge is to manage accesses to applications and data from many different locations and devices while still ensuring security. For this reason, partner XiTrust will provide an identity management service that equips the users with hard electronic identities. These are especially made for identification and authentication at a high assurance level and are based on qualified certificates. The process of getting such an identity looks like this:

1. A user who wants such an identity, visits a registration officer. Furthermore, the user needs to have a valid passport and a mobile phone for the registration process.
2. During this process the registration officer enters personal information of the user, such as mobile phone number, birthday and other information from the passport.
3. The user chooses a revocation password if he/she wants to revoke his/her certificate of the hard eID and a password that he/she needs whenever the person wants to sign a document or authenticate him-/herself.
4. In the next step the registration officer guarantees with his/her hard electronic identity that the registering person is really the person he/she pretends to be and that all the entered information is correct.
5. The registering person now owns a valid identity and can sign documents and authenticate him-/herself with his/her hard electronic identity.

The partner XiTrust will be responsible for the identity management and employs the so called registration officers (RO) and central registration officers (cRO). These central registration officers are special ROs, who have been certified to educate persons as ROs. This partner is therefore able to equip all partners with hard eIDs or, if wanted, can educate people from other partners as ROs.

Use Case. If a person has gotten a hard electronic identity and he/she wants to use it now, he/she just needs to open the login page of an application and choose the mobile phone login option. The user is then asked to enter a mobile phone number and the corresponding password which looks like as illustrated in figure 11. In a next step the user receives a one-time password on his/her mobile phone which needs to be entered into the login mask as well. This login mask is shown in Figure 12. Alternatively, the user can handle the login with a QR-code instead of a one-time password. In this case a QR-code

will be displayed on the screen of the user and he/she needs to scan this QR-code with the appropriate Application on his/her mobile phone. The user is now authenticated; the process for digitally signing a document is identical. The signatures that can be created with this hard electronic identity are so called qualified electronic signatures and are therefore legally equal to handwritten signatures.

Figure 11: Mobile phone number and password request for the login

Figure 12: One-time password request for the login

eIDAS Regulation. The hard electronic identities from above are compliant with the eIDAS regulation and are therefore valid in the whole of Europe. The Regulation (EU) No 910/2014 on electronic identification and trust services for electronic transactions in the internal market (eIDAS Regulation) was published on July 23rd, 2014 and shall apply as of July 1st, 2016. This regulation establishes a framework ensuring that a user's national electronic identity issued in one Member State is also valid in another EU Member State and is the reason why the electronic identities from above are valid in whole Europe. Additionally, an interoperability framework will be established to simplify electronic interactions between businesses, citizens and public authorities from different countries. Especially the electronic seal is very interesting for companies as they can authenticate all their electronic assets with it. Until now it was necessary that a natural person, representing the legal person, signed the electronic assets to guarantee authenticity. The goal of this regulation is to enhance the trust in electronic transactions throughout the European market and to eliminate obstacles when using electronic identification means across borders. The regulation covers the creation, verification and validation of the following trust services:

- electronic identification of natural and legal persons
- electronic signature (as well as advanced and qualified electronic signatures)
- certificate for electronic signatures and seals (as well as qualified certificates for electronic signatures and seals)
- electronic seal (as well as advanced and qualified electronic seals)
- electronic time stamp (as well as qualified electronic time stamp)
- electronic registered delivery services

4 Prismacloud Services

This section provides basic documentation of the PRISMACLOUD services according to the guidelines defined in the previous section. It is intended as a major part of the service documentation and serves as a **comprehensive reference manual for service-level adopters** of the project's results. The current documentation was also a first successful test for implementation of *CryptSDL* in the project consortium and showed the potential of documentation standards in research projects.

4.1 Secure Archiving (SAaaS)

4.1.1 Overview

The secure archiving service (SA or SAaaS) is tailored for creating reliable and trustworthy backups. Its functional key requirements are motivated and derived from our e-Government use-case but have been designed to be compatible with commonly encountered backup scenarios. Within this section we initially introduce general backup ideas and key-features, explain the proposed service view, discuss functional requirements in detail and finally focus upon the security impact upon our user-case prototype.

The general idea of the secure archiving service is to provide an extremely reliable secure storage back end for backup and archiving purposes. The former places high importance on transparent, easy and quick access to data while providing excellent integrity guarantees. The archiving use-case focuses upon data retention over long time periods. Commercial solutions offer additional features such as searching over backup data which are diametrical to data confidentiality and privacy. Within our prototype we focus upon confidentiality and assume that commercial solutions can incorporate our component as core storage layer and add additional features such as searching on top of it.

Key design concept was to enable hybrid cloud storage scenarios, i.e., the system can be scaled-out to integrate public cloud offerings as with existing interfaces. The Simple Storage System (S3) standard from Amazon emerged as a de-facto industry standard API in the cloud storage domain. Therefore we are designing the service to be compatible with this standard on both sides, the back end storage level as well as to the client.

4.1.2 Key Features

In the following we present the major advantages of the service. The following list also highlights the relevance of different benefits, whether they are more relevant for typical archiving (Arch) or backup scenarios (Back).

- Increased *data privacy and availability* (Back/Arch): The usage of threshold secret sharing schemes for data sharing improves both data privacy and availability. In a multi-cloud configuration, a single provider only holds a data fragment and is not able to read or tamper plaintext data. When using computational secure shar-

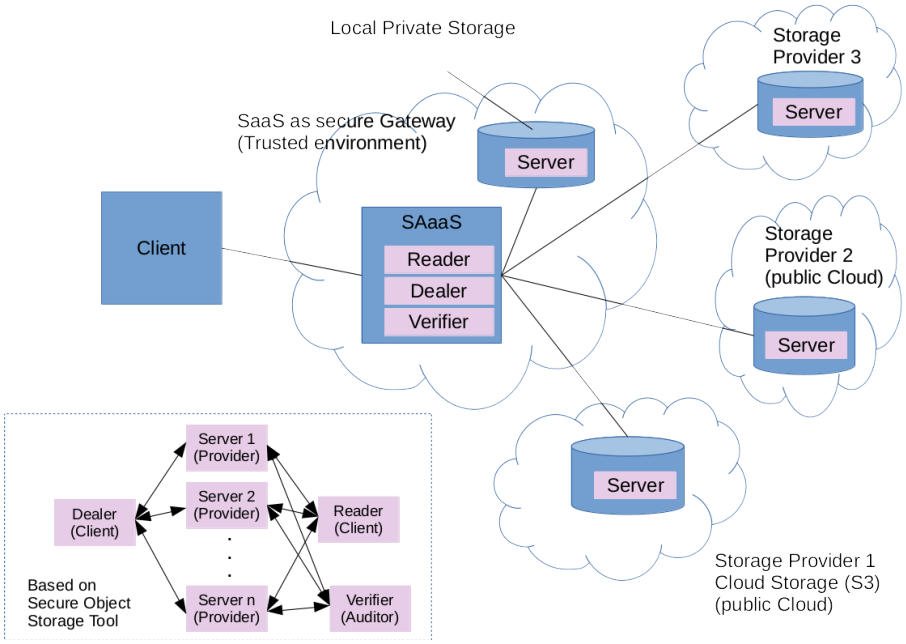


Figure 13: Secure Archiving (SA) overview.

ing (CSS) the resulting system delivers the strengh of replicated encrypted backup solutions while improving efficiency and flexibility, i.e., it helps with integrated business continuity features. Availability is improved as no single instance has a crucial impact to the overall system performance.

- *Prevent from vendor lock-in* (Back/Arch): Using a multi-cloud capable distributed approach for data storage prevents provider lock-in. When data is dispersed over multiple zones or providers, interoperability and portability is included by design. If new offerings are more appealing or existing cloud provider change their policies, customers can switch to new providers without service interruption due to format operations and interface compatibility problems.
- *Keyless (credential based) operation* (Back): The major difference to encrypted solutions is the keyless nature of secret sharing based systems. Their security is based upon the non-collusion assumption of storage providers: this assumption can be realized by the usage of different administrative zones, different data centers or the use of different cloud providers. The keyless operation has advantages over encrypted backups: on one hand, encrypted backups or archives require a secret key to be maintained over the entire lifetime of the data. The whole backup redundancy becomes obsolete if the respective keys are lost, i.e., key management thus becomes a critical element which needs to be done on site. On the other hand, selective sharing of the data with different stakeholders is not easily possible for encrypted data because secret keys have to be shared or public key infrastructures have to be set up.
  However, for secret shared data an access control system is inevitable to control access to data fragments. This would not be necessary for encrypted data, but is

typically also used there to prevent ciphertexts being exposed to untrusted entities. In the end, the most important difference lies in the security model, i.e., secret key vs. non-collusion based security, which require careful consideration in for the given use case and type of data. Ideally both options are combined the get the best of both worlds, which is what we also support with our PRISMACLOUD secure archiving service.

- *No data remanence (Back/Arch)*: Because the single storage servers do not have access to plaintext data, specific destruction techniques are not necessary for timed data deletion. In general it is not possible to securely delete data on cloud infrastructures. If data confidenciality is of importance, i.e., due to sensitive data, encryption has to be used. This encryption has to be performed on the client: if server-side encryption is employed, keys are at some point in time present at the remote system and could eventually be recovered by forensic means.
- Support for *long-term security* (Arch): When security is more important than storage efficiency, data can be encoded with perfect secret sharing (PSS). Such a scheme delivers perfect or unconditional security which is provably unbreakable. Using a PSS, one can protect data that needs to be confidential even over decades. However, it is important to note, that the security is governed by the non-collusion assumption which is fundamentally different from standards secret and public key cryptography. Nevertheless, because there do not exist any other perfectly secure scheme for data storage, this is the best what can be done with this respect. For better collusion resistance we recommend to combine the system with secret or public key encryption, i.e., to get the best of both worlds.
- Support for *multimodal encryption* (Back/Arch): To get the best of both worlds, the services supports both modes of encoding, efficient CSS and long-term secure PSS. The modes can be configured on a bucket or folder level to enable storage of bulk data along with high efficiency (CSS) and with classified or sensitive information with highest security (PSS) in the same system. Additionally, the encryption support is also integrated as a means for collusion resistance and increased security.
- Support for *remote system auditing*: To enable the SAaaS provider to regularly check the consistency of the data stored on the storage nodes we provide an efficient protocol with minimal network overhead to audit the state of the system. If the system passes the audit procedure, it is guaranteed that all storage nodes are still in possession of all stored fragments. If an error is detected by the auditing mechanism, a repair procedure is triggered to synchronize the corrupted node(s) with the system state. As long as the preconfigured minimum number of nodes is still available, the systems stays fully operational. If the SAaaS provider is serving multiple customers, we offer the opportunity of bulk auditing data of multiple customers at once to gain high-efficiency audting. This would not be efficiently possible with other remote data checking technologies.
- Can leverage *public cloud storage* (Back): All components of the PRISMACLOUD SAaaS service are designed for deployment on public clouds, i.e., they do not use any specific low-level functionality or infrastructure-dependent internal interfaces. The whole distributed storage layer can be deployed as an overlay architecture on

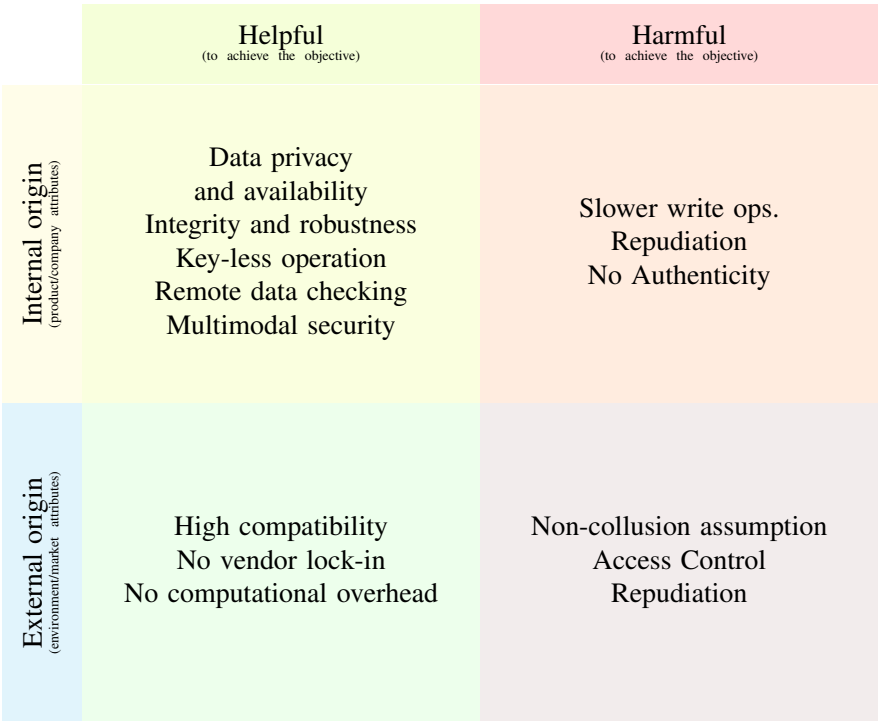


Figure 14: SWOT analysis of Secure Archiving Service (SAaaS).

top of existing industry standard public cloud offerings. The SAaaS gateway service is a Java based application which can be deployed on any technology, e.g., in a dedicated virtual machine or an event driven framework. The gateway is intended to be deployed by a CSP which is offering the service to customers, i.e, it is assumed that he has full control over the software stack. However, because the gateway is a Java application it is agnostic to the underlying technology. The storage nodes used to store the fragments can be any storage compatible to the Amazon S3 (Simple Storage Service) protocol, the de-facto industry standard for object storage. Additionally to the basic S3 compatibility, for the auditing feature to work the cloud should also support compute capabilities like virtual machines or containers. They are necessary to support the auditing capabilities, do activity monitoring and provide proactive security for long-term archiving.

- *Integration with Legacy Systems*: Finally, to enable efficient integration, the user-side interface to the storage architecture is also S3 compatible. It enables customers to transparently run highly reliable and secure distributed S3 storages without special software. After setup, the service can be used as a direct replacement for S3 based backup solutions.

In Figure 14 we show a SWOT analysis of the given service.

4.1.3 Usage model and stakeholders

As shown in Figure 13 the scenario assumes the concept of a trusted gateway managed by a trusted provider entity. Three different classes of stakeholders are involved:

- The *Cloud Customer*, also called client, gets access to a secure multi-cloud storage solution through an accessible industry standard interface (Amazon S3). He is able to leverage the benefits of such a system without complex management of a distributed system or dealing with installation and management of the PRISMACLOUD SAaaS software. However, as the SAaaS gatway operator is fully trusted, the consumer needs a good trust relation to that operator.
- The *SAaaS Provider* is running and maintaining the SAaaS gateway. He is managing the storage providers in the backend as well as the software running there—they are managing the whole overlay architecture on behalf of the customer. The SAaaS Provider could also be a storage provider. In this case, he might be using the secure cloud storage as scale-out storage option to achieve higher availability and security guarantees.
- The *Storage Provider* is holding data fragments of user data and serves as a so called zero-knowledge storage provider, i.e., he has no access to plaintext data. No special requirements are necessary except for the possibility to also support an active compute instance. Furthermore, the availability guarantees required from the storage provider are minimal and they do not need to provide internal replication. The good availability comes from the system architecture and build in redundancy and not highly reliable SLA for individual provider level.

4.1.4 Service Model and Interaction Dynamics

The high level component diagram shown in Figure 15 shows the main concept plus required software components which are interacting in the system:

- *Client:* This is the customer application using the storage service. Because SAaaS is providing a S3 compatible interface the requirements on the client side are minimal. Basically all S3 compatible software solutions can be used together with the PRISMACLOUD SAaaS solution. For configuration a special admin interface must be provided to the customer which enables access to the specific settings available in our system, however, this is specific to the implementation and can be made available over a different web service.
- *Secure Archiving Service:* This is the main component of the system and serves as a gateway to the clients run by the SAaaS provider. The archiving service comprises two main components, the storage gateway and the auditing module.
  - The *storage gateway* is doing all the fragmentation (encoding) and defragmentaton (decoding) of data and manges the storage nodes. It exposes an S3 compatible REST interface to the client and uses S3 storage services in the backend, i.e., it intends to be compatible with industry standard and serves as a drop in replacement for using plain S3 storage solution with a single provider.

  - The *auditing module* is responsible for system monitoring and data checking. It gathers relevant information to assess the system state, checks SLA compliance on both sides, to the customer and of the storage providers. Additionally he also regularly triggers a storage audit which checks the integrity and availability of the data in the overall system.

  As mentioned in the client description, an additional administrative interface will also be required to manage the system settings. Because this is very specific to the scenario and not relevant to the overall system it is not specified in the service architecture.
- *Storage Node:* This is the component providing storage for fragments. It represents a node in the storage network and comprises following two components.
  - *Object Storage* This is the standard IaaS passive object storage used to hold the fragments. No more than a minimum set of standard S3 IaaS storage capabilities is expected to be compatible with PRISMACLOUD SAaaS. Provisioning and management of resources is done by the secure archiving service.
  - *Security Monitor* This element is required to support additional functionalities provided by the archiving service. It is a software component which has to be run on the storage provider side and needs access to the local data. It is controlled by the auditing module of the SAaaS and executes the auditing procedure if triggered. It also collects monitoring information and preprocesses log trails if requested.

4.1.5 Provider/Consumer Scope of Control

The deployment of the secure archiving service is involving more parties than a typical archiving service. This is due to the nature of the service which is intended as a virtual storage service built on top of multiple individual storage offerings. For the secure archiving service we introduced the SAaaS provider which is offering and managing the secure gateway functionality as a service. He or she is in full control of the PRISMACLOUD SAaaS middleware Archistar. The scope of control in the software stack is shown in Figure 16 and discussed in the following.

- *Cloud Consumer:* To foster quick adoption of the service, the cloud consumer scope of control is reduced to an absolute minimum. It should resemble the standard usage experience of S3 cloud storage with only minimal additional controls exposed to the user. The consumer has no specific software components to operate and manage apart from the S3 client software used to access the storage system for data up- and download.
- *SAaaS Provider:* The SAaaS provider is responsible operating the PRISMACLOUD Archistar middleware software solution and also for hosting the SAaaS service. To host the archiving service they have to run the storage gateway software and also the assurance module. Additionally to the local deployment the SAaaS provider is also responsible to manage the object storage of the various storage nodes and to operate and manage the security monitor component.
- *Storage Provider:* This is the IaaS provider provisioning the real storage space which

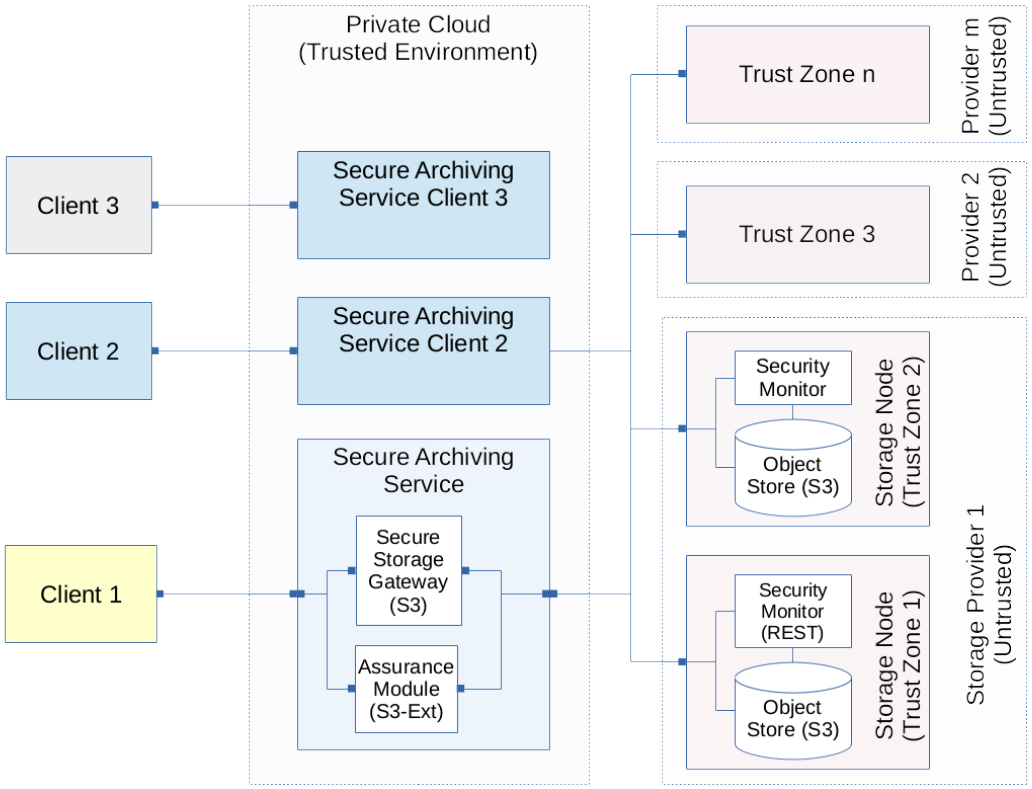


Figure 15: Interaction dynamics for secure archiving service.

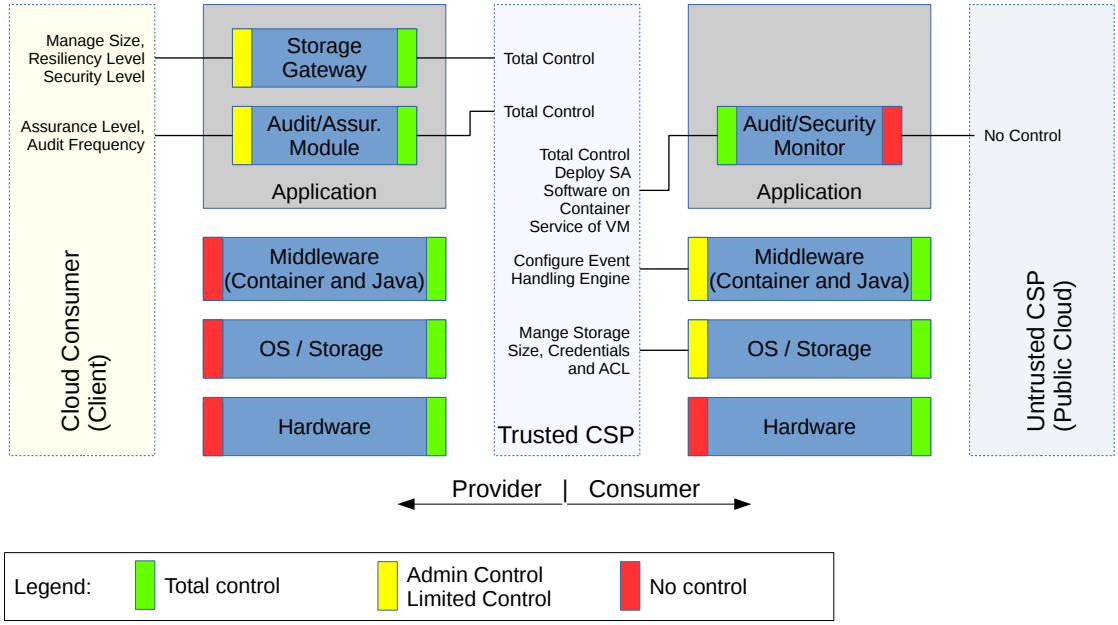


Figure 16: Scope of contol secure archiving service deployment.

hold the data fragments. He full control over his own infrastructure and is able to access all data stored. Furthermore, he is also hosting the security monitor in his compute infrastructure but has no obligations nor access to it. However, because the security monitor is running in his infrastructure he cannot be prevented from inspection of the security monitor component. However, the security monitor has no critical information and is just executing commands from the assurance modules and not containing any sensitive information.

4.1.6 Parameters

When secret sharing or information dispersal is applied to increase security and availability in data storage, the threshold configuration parameter k of the sharing algorithm is essential to set the level of security and the redundancy $n - k$ is the main parameter to tune the reliability and availability of the overall virtual storage system. Detailed models to calculate availability and corresponding overhead considerations are discussed in the SECOSTOR tool specification in deliverable D5.3 and more advanced decision support models have been developed and analysed in D7.3 and D7.4. The capability models of D7.3 also present detailed solutions also considering geo-location constraints.

4.1.7 Application Development

Attack Surface Analysis

Within the Backup use-case the user-facing service is an Amazon-S3 compatible Backup-Service where users can upload, list and download plaintext backup data. The service itself performs secret sharing (which splits up plaintext into multiple shares, a subset of which is needed to reconstruct the original data). The created shares are distributed upon multiple storage locations. All this functionality is provided by the backup service which is fully trusted.

The network-based attack surface is based upon the Amazon REST interface provided (front-end) and consumed (back-end) by the Backup Service. We only provide a subset of data operations, thus further limiting the attack surface. On the front-end, the following operations are provided:

- list data buckets
- list files within data bucket
- add file to data bucket
- retrieve file from data bucket
- delete file from data bucket
- copy file between data buckets

The back-end utilizes an even smaller subset of operations:

- add file to data bucket
- delete file from data bucket

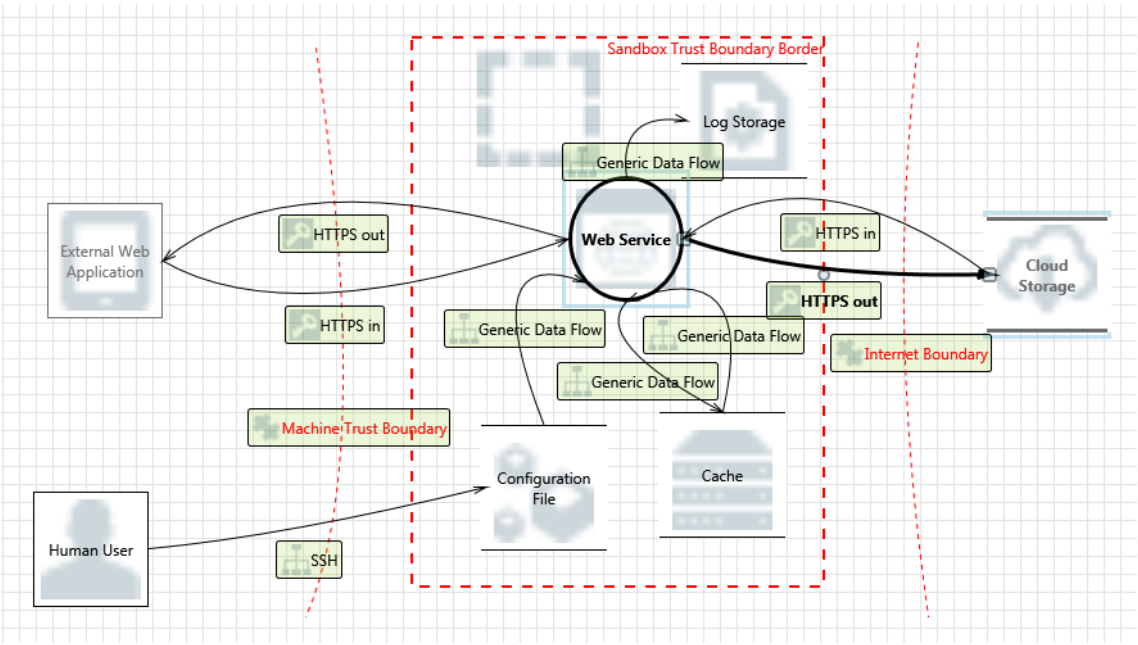


Figure 17: Attack surface model for secure archiving service.

As this is a new project, we are not encumbered by existing legacy protocols that must be integrated for product integration. This allows for a slim service interface without duplicate operations.

We deploy the application server directly, i.e., without an additional web-application reverse proxy or web-application firewall. There is no distinction between user groups for those operations, i.e., this further reduces the attack surface.

When it comes to file-based attack-vectors, we separate configuration data from user data. User-data is processed by secret sharing by the storage gateway. When using perfect secret sharing this removes all outgoing data from the attack surface. If using computational secret sharing, the security of outgoing data is exactly the security of the symmetric encryption scheme (e.g., AES with a suitable mode of operation) and can thus be removed from the attack surface. We do not cache unencrypted user-data on-disk within the backup gateway. In sum, no user-data is part of the attack surface.

All front-end bucket configuration (including user accounts, credentials and access rights) is stored within the storage gateway. Access location and credentials for back-end storage is also stored within the storage gateway. Both elements are sensitive and thus part of the attack surface.

Figure 17 summarizes our potential attack surface.

Gathering Threats with STRIDE

We utilized the Microsoft Threat Modelling Tool to arrive at Figure 17. This model was base for automatic threat geneation, the a representation of the results can be found in

the appendix in Section B.1.

All threats were further analyzed, i.e., verified that the threat is applicable and not mitigated through our chosen architecture. The following table gives a rough overview of the resulting threats:

STRIDE Category	Original	Not-Applicable	Mitigated/Design	Resulting
Spoofing	9	4	1	4
Tampering	6	1	2	3
Repudiation	13	1	1	11
Information Disclosure	7	0	1	6
Denial of Service	13	4	4	7
Elevation of Privilege	5	0	0	5
Total	53	10	9	34

Non-applicable threats are explained by our trust assumptions: the whole backup gateway is assumed to be fully trusted thus reducing the threat arising from communication between internal components such as caches or configuration files.

Threats mitigated by design can be explained by our reliance upon secret shared data within the cloud. As no plaintext data is stored within the cloud and no single cloud provider has total control over data, two areas are greatly reduced: denial of service and tampering. It can be argued, that these are the areas that benefit from our security by design approach.

Please note, that we only added a single cloud provider within our STRIDE diagram. If we would add a realistic amount of cloud providers (e.g., five) the number of threats mitigated by design would also rise. The 36% threat reduction seen in this example are thus the minimal threat reduction experienced.

Analyzing and Mitigating remaining Threats

Threats were rated according to ISO 27005 in two dimensions: business impact and likelihood of occurrence. Both rankings ranged from 0 ("minimal impact") to 4 ("critical impact").

For Business Impact we have chosen the maximum ranking of 4 for all threats that either allow for unrestrained data modification or user impersonation. Loss of user data would imply a critical business impact. If the business-provided service's availability is compromised, a base ranking of 3 was given. If attacks only impacted a small user amount (i.e., only users adjacent to a network path) the impact was lowered by one.

The ranking for Likelihood was influenced by the technical complexity of an attack as well as by current trends within web exploitation techniques (as given by the OWASP Top 10).

ID	Threat	Business Impact	Likelihood	Result
15	Potential Process Crash or Stop for Web Service	3	3	6
16	Data Flow HTTPS Is Potentially Interrupted (between Backup Service and Web Application)	2	2	4
17	Web Service May be Subject to Elevation of Privilege Using Remote Code Execution	4	3	7
18	Elevation by Changing the Execution Flow in Web Service	4	3	7
25	Insufficient Auditing (outgoing traffic)	3	1	4
26	Potential Weak Protections for Audit Data	2	1	3
27	Weak Credential Storage	4	2	6
33	Potential Excessive Resource Consumption for Web Service or Cache	1	2	3
41	Access to Configuration Files is potentially interrupted	2	2	4
44	Spoofing of the External Web Application External Destination Entity	3	3	6
45	External Entity External Web Application Potentially Denies Receiving Data	2	2	4
48	Risks from Logging	1	2	3

In this example, the risk ranking indicate that Elevation of Privilege attacks (ID 17 and 18) are of the highest concern, followed by multiple Denial-of-Service attacks (ID 15, 27 and 44). Due to our secret sharing approach, there are limited risks arising from the remote cloud storage providers.

STRIDE provides a catalogue with countermeasures for detected threats. As part of our secure development life-cycle we have selected key countermeasures that must be implemented within our prototype to further mitigate resulting threats. The Table in 18 gives examples of high-level mitigation techniques.

The archiving scenario allows us to mandate strong authentication and authorization between the archiving client and our backup storage service. This reduces threats from the Spoofing identity, Tampering with data and Denial of Service categories by-design. The software architecture was designed to run with minimal user rights thus reducing the likelihood of Elevation of Privilege attacks.

The usage of secret sharing for securing external data storage (within the external cloud data providers) further mitigates threats as this implicitly reduces the categories Spoofing Identity, Tampering with Data, Denial of Service and Information Disclosure. The backup service is fully trusted by design, this mitigates Repudiation as this service can be used for creating trusted audit trails.

To fulfill a threat, an attacker exploits vulnerabilities. The OWASP Foundation releases

Threat Type	Mitigation Techniques	Concrete Examples
Spoofing Identity	Appropriate authentication	All connections authenticated
	Protect secret data	verify that cloud credentials are distinct
	Don't store secrets	not applicable
Tampering with data	Appropriate authorization	All connections authenticated
	hashes	Usage of RSS/VSS
	MACs	Usage of RSS/VSS
	Digital signatures	Usage of RSS/VSS
	Tamper resistant protocols	Implicitly through Secret-Sharing
Repudiation	Digital Signatures	Logs will be signed
	Timestamps	Backup Service provides trusted timestamps
	Audit trails	Backup Service provides trusted audit trails
Information Disclosure	Authorization	All connections authenticated
	Privacy-enhanced protocols	done by design
	Encryption	Provided by Secret-Sharing
	Protect secrets	Only stored within Trusted Backup Service
	Don't store secrets	not-applicable
Denial of Service	Appropriate authentication	All connections authenticated
	Appropriate authorization	perform authorization against trusted backup service
	Filtering	not-implemented (yet)
	Throttling	rate-limits per user implemented
	Quality of Service	not implemented (yet)
Elevation of Privilege	Run with least privilege	backup service runs with normal user privileges

Figure 18: STRIDE Threat & Mitigate Technique list (from OWASP)

bi-yearly lists of the most common vulnerabilities for web applications. The following table overviews how the chosen architecture mitigates some of those vulnerabilities.

Vulnerability	Counter-measure
XSS Injection	API-only interface, no HTML or JavaScript code will be executed
Injection	all data-access must be sanitized, i.e., use sanitation libraries as well as ORM
Broken Authentication and Session Management	usage of framework-provided session framework to prevent home-grown vulnerabilities; all access must be validated
Insecure Direct Object References	see Broken Authentication and Session Management
Missing Function Level Access Control	see Broken Authentication and Session Management
Using Components with Known Vulnerabilities	indirectly solved by secure component selection
Unvalidated Redirects and Forwards	not-applicable to API-only application

Validation

Validation is an important step during the secure software development life-cycle and highly depends upon prior generated artefacts:

- a written-down attack surface description allows validation of minimalism
- during testing the implemented attack surface, i.e. API, is compared to the designed attack surface and must be equal or smaller
- during validation, countermeasure against all identified vulnerabilities are validated and documented
- additionally identified vulnerabilities are used as input for the next iteration of security engineering as well as checked-against within other scenarios. This feedback loop allows for continuous improvement of our scenario's prototype security.

4.1.8 Operational Aspects

In this section we present a preliminary security assurance model for the secure archiving service according to the approach proposed in [HTL+14]. The basic idea is to derive a monitoring strategy for a security monitor component, which is intended to continuously assess the system status.

We did this step for the secure archiving service, however, for the remainder of the services this is out of the scope and we do not give such detailed modeling. Instead we discuss general important operational aspects where necessary and report lessons learned from the evaluation of the pilots in WP8 reports.

In the following, we will illustrate the assessment of the secure archiving service that we simplify by only focusing at its core part, depicted in Figure 19. The security requirements are monitored by validating infrastructure security conditions via security properties. We outline an exemplary set of security properties clustered across security classes (confidentiality, integrity and availability) in Table 4.

Table 4: Assurance security properties and corresponding classes analyzed and developed under the scope of EU FP7 research project SECCRIT.

Assurance Class	Security Property
Confidentiality	CSP_1 - Concurrent session control CSP_2 - Password Rotation CSP_3 - Strong Password CSP_4 - Maximum shares per zone CSP_5 - Encryption
Integrity	ISP_1 - System/Service Integrity ISP_2 - Information (Data) Consistency ISP_3 - Alteration Detection ISP_4 - Error Correction
Availability	ASP_1 - Geo-location ASP_2 - Service Availability ASP_3 - Service Isolation

Furthermore, we illustrate per each assurance class from Table 4 a single security property policy, Table 5, and detail how it is being validated/monitored across our architecture model Figure 19.

For illustration purposes we propose from Table 4 a security property evaluation set (e.g., ES = $\{CSP_1, CSP_2, CSP_3, CSP_4, CSP_5, ISP_1, ISP_2, ISP_3, ASP_1\}$) that is used during our security assurance assessment process. The evaluation set is used consistently at client, tenant and infrastructure level to validate security condition of each entity. This is necessary to show that the infrastructure, i.e., trusted zones, where the data shares are deployed is reliable with regards to our evaluation criteria defined by the ES. Therefore, each individual trust zone, across all availability zones, must be validate against ES. The above mentioned evaluation set ES in the assurance assessment framework is used in a form of a bitvector, where each bit corresponds to a particular security property in the exact ordering as defined in the ES. Next, each property is validated per an individual component of our secure archiving service depicted in Figure 19. To derive the overall security assurance results we have to aggregate the results towards the root of our service, in this case client service. The aggregation process is conducted by performing horizontal and vertical security assurance aggregation. Horizontal security aggregation process ag-

Table 5: Assurance Class Integrity - Security Property System/Service Integrity

Layer	Monitoring Points
	ASP - Service Availability
Infrastructure	Cloud Infrastructure availability is validated.
Tennant	Availability of VM and storage (e.g., S3 bucket) that holds the shares is validated.
	The auditor runs the Remote Data Checking (RDC) protocol to verify the correctness of stored shares.
	Trusted gateway is availability is measured.
Client	Secure Archiving service configuration files consistency is validated to detect any changes from predefined configuration setup.
	ISP - System/Service Integrity
Infrastructure	None.
Tennant	The authenticity of software running on the trusted gateway is validated to detect any changes.
	The correctness of VM and storage configuration is validated to detect any changes from predefined configuration setup.
Client	Secure Archiving service configuration files authenticity checked.
	ASP - Maximal Shares
Infrastructure	Number of shares across all availability zones per Cloud Service Provider must be lower than threshold.
Tennant	Storage containers are validated against maximum number of predefined shares.
	Distribution process of share across individual trust zone is validated.
Client	None.

gregates security results in a horizontal manner across components that reside under the same parent element, i.e., trusted zone components. Vertical security aggregation process aggregates security results in a vertical manner by aggregating either the results of horizontal aggregation or a single child parent relation. In case of our secure archiving service we first perform vertical aggregation of storage service towards the VMs containing our storage service, and the perform horizontal aggregation first of the results at individual trust zone and then between the trust zones. Afterwards the results are vertically aggregated towards the root against the trusted gateway and finally with the root client service itself. The auditor is in this particular case used only as a supporting tool for our security assurance process.

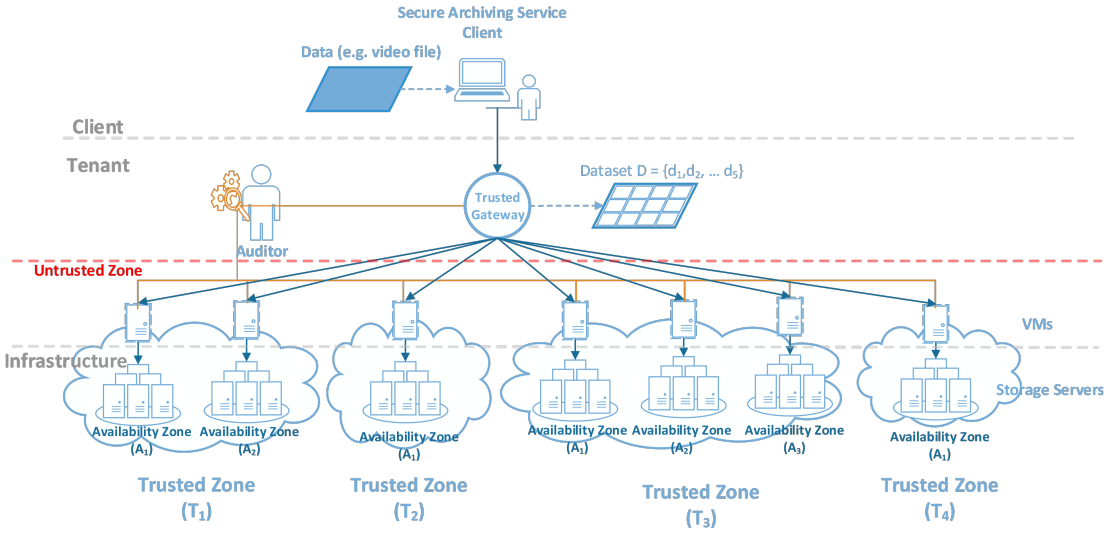


Figure 19: Secure archiving assurance use case

The Table 6 outlines how does our security assurance framework address individual design requirement. The protection of confidentiality and availability is supported by assurance framework with security properties (CSP_1,CSP_2,CSP_3,CSP_4,CSP_5, ASP_1,ASP_2,ASP_3) from confidentiality and availability classes (SA.Req.01). The audtiability is a process that is integrated as a part of the assurance framework as a standard process that establishes the audit trail of the acquired security information (SA.Req.02). The deployment of the assurance framework can be outsourced to a third party, or even hosted locally, in such a form that the processing and transformation of essential security sensitive information remains on site, and the auditing part is place on the third party infrastructure (SA.Req.03, SA.Req.04, SA.Req.05). Furthermore, the audit established by the assurance framework is used to offer guarantee that across a certain period of time security requirements are set in place (SA.Req.06). The assurance framework is implemented in such a fashion to address the processing of large scale data (SA.Req.08).

Table 6: Mapping requirements of the secure archiving service towards the assurance security properties.

Requirement	Mapping
SA.Req.01	The Assurance model supports confidentiality and integrity by validating the following security properties: CSP_1,CSP_2,CSP_3,CSP_4, ASP_1,ASP_2,ASP_3
SA.Req.02	The Assurance module checks if remote data checking protocol is supported and run on a regular basis.
SA.Req.03	The Assurance module can be hosted both on- and off-site, because it has no access to plaintext information.
SA.Req.04	The Assurance module can trigger privacy preserving audit runs various remote server and do a quorum for on the result.
SA.Req.05	The Assurance module checks if ITS secret sharing is used for data.
SA.Req.06	The Assurance module validates that proactive security measures are enabled and run on a regular basis.
SA.Req.07	The Assurance modules checks the state of the BFT modules at the different server nodes and compare them.
SA.Req.08	The Assurance model checks if batch auditing is supported and if it could be applied over data of multiple users.
SA.Req.09	The Assurance model validates the following security properties: ISP_1,ISP_2,ISP_3,ISP_4 and triggers rebuild operations if necessary.

4.2 Data Sharing (DSaaS)

4.2.1 Overview

The general idea of the *data sharing service* (DS or DSaaS) is to support the creation of cloud-based collaboration platforms similar to well-known cloud-based consumer solutions while heeding stronger privacy and availability guarantees. The data sharing service allows multiple parties to securely store data in a multi-cloud network without giving single providers the ability to read the data. Although the data is protected from all kind of provider related threats it can still be shared with other users of the system in dynamic groups. To fulfill this idea we are using the *Secure Object Storage Tool* as base.

In contrast to the secure archiving service, no single point of trust or failure shall exist in the data sharing service and all encryption and encoding is entirely handled at the client side. This configuration gives the highest level of privacy but introduces complex configuration and system management. Also contrasting, we assume the archiving use-case to utilize short-time storage, thus having reduced storage and performance requirements. This fits very well with the evidence gathering and sharing requirements of the law enforcement use-case.

An overview of the overall service and deployment model for the data sharing service is given in Figure 20. Although the data sharing service is based on the very same secure object storage tool like the secure archiving service, it targets a completely different application domain and often uses complimentary features. The trust models underlying those two use cases are different. No trustworthy provider is assumed nor is a single user scenario enough to cover the requirements. Furthermore, most of the clients have to be considered

untrusted and their operations should be verified and logged as good as possible.

4.2.2 Key Features.

The core benefits of the data sharing system are quite similar to the one of the secure archiving service described at Section 4.1, but due to the different focus of the data sharing service additional features are supported and others left away, e.g, long-term security is not in the focus. In particular, the data sharing service provides the following key features:

- Increased *data privacy and availability*: The same as in SAaaS, see Section 20 for details. These feature are given by the architectural design and the selection of used algorithms.
- *No provider lock-in*: The same as in SAaaS, see Section 4.1 for details.
- *Keyless operation*: The same as in SAaaS, see Section 4.1 for details. Especially, the benefits for selective sharing of data in dynamic groups without complex key management can be considered a major benefit of the system compared to systems using plain encryption technologies.
- *No data remanence*: Secure deletion is supported, like in the SAaaS, see Section 4.1 for details.
- *Leverage public cloud storage*: The same as in SAaaS, see Section 4.1 for details.
- *Support multiple users and collaboration*: The system supports multiple user at the same time and can therefore be used to provide a storage service to organizations and work groups. Furthermore, due to the keyless feature of the system collaboration can be achieved easily and in a very dynamic fashion. Users can be added or removed from groups and roles without the need for key revocation or re-encryption of stored data.
- *Concurrent access* from multiple clients of multiple users is supported. The system provides strong consistency guarantees and synchronizes the access in a robust way.
- *Malicious client detection*: In systems which apply secure information dispersal, the consistency of stored data cannot easily be checked any more without revealing the plaintext at some servers. This is a major problem in systems without single point of trust and requires additional verifiability means. The PRISMACLOUD architecture provides efficient verifiability protocols which enable the system to check if the clients stored consistent data, which is important if the clients can be malicious or have self-interest to store corrupt data without being noticed.

In Figure 21 we show a SWOT analysis of the given service.

4.2.3 Usage Model and Stakeholders

As shown in Figure 20 we distinguish three basic types of stakeholders:

- The *Cloud Customer* is running the clients and should be a group of people, e.g., and organization or network of organizations, who want to collaboratively manage information with high levels of security, privacy and assurance. They utilize a fully

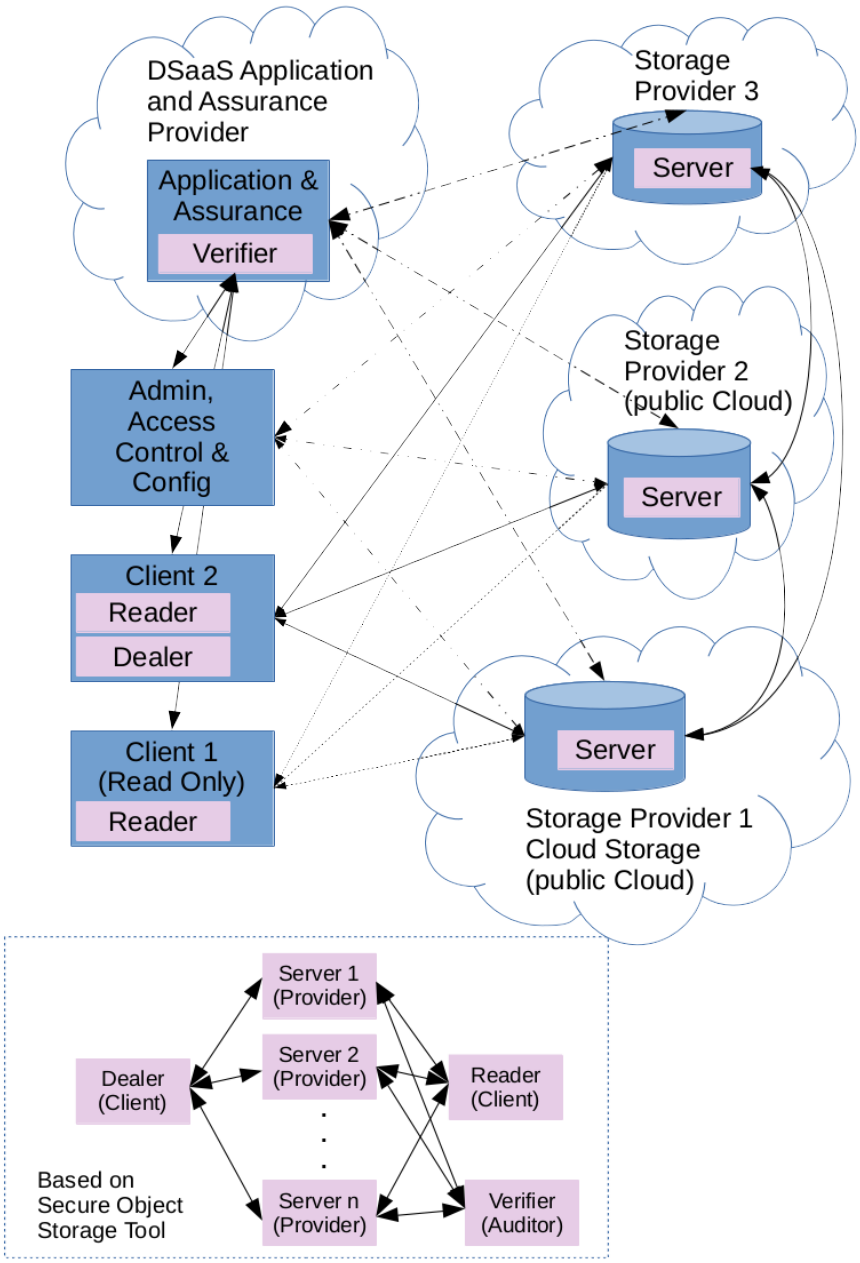


Figure 20: Data sharing (DS) overview.

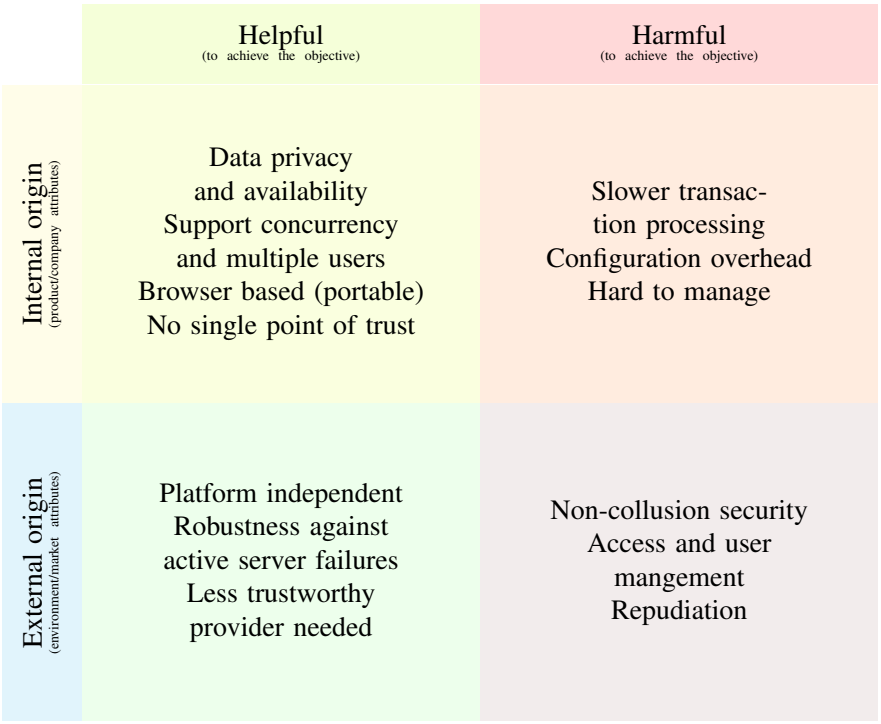


Figure 21: SWOT of Data Sharing as a Service (DSaaS).

client-side encryption solution that securely disperses data over multiple data center. There is no single point of trust or failure within the system. From the DSaaS provider he gets trustworthy assurance reports about data access and malicious client behavior.

- The *DSaaS Provider* hosts the web portal, maintains the software and runs the assurance monitoring for the storage system. The DSaaS provider is responsible for storing basic configuration and user data as well as managing access rights. However, the DSaaS provider do not have any access to plaintext data and is therefore acting as a zero-knowledge provider—the system does not need to be fully trusted and therefore resembles a security and privacy friendly data management platform. The assurance monitoring regularly collects access reports on all servers and runs audit and verification protocol to check for corrupt nodes or malicious client behavior. The assurance monitoring is done in a privacy preserving manner that the DSaaS provider does not learn anything about the evidence data.
- The *Storage Provider* is holding the data fragments of user data and serves as an so called zero-knowledge storage provider, i.e., he has no access to plaintext data. No special requirements are necessary except for the possibility to also support compute instances. A PRISMACLOUD storage node always requires some remote logic to be executed on the remote storage nodes to handle advanced features like verifiablity, auditing or multiuser concurrency. Beside that, the availability guarantees required

from the storage provider can be minimal and no provider internal replication is needed as availability is provided through the system's architecture with built in redundancy.

4.2.4 Service Model and Interaction Dynamics

The high level interaction diagram shown in Figure 22 shows the main components of the system and their distribution to stakeholders.

- *Client:* This is the customer application enabling users to access the storage system. The application will be fully web based and run as browser based application. It is based on modern technologies like JavaScript, HTML5, CSS and WebSockets. It performs all encoding and communication with storage nodes at the client side. This way, no other component, not even the DSaaS provider, will ever have access to plaintext information. Outsourcing the fragmentation of data to the client is necessary to fulfil the requirement of having no single point of trust (but the clietn) in the system.
- *Datat Sharing Service:* The data sharing service is hosted by the DSaaS provider and comprises two main components, the portal server and the admin and assurance module.
  - The *Portal and Application Server* is the central connection point for both, end users as well as admins. It hosts the main application software which is served to the clients upon connection. It also stores encrypted configuration files to the clients to facilitate high portability of the applications.
  - The *Admin and Assurance:* The main server assists in administrative tasks like user management and system monitoring. However, the data sharing service does not have enough permissions to access plaintext data of users or similar superuser capabilities known from local system administration. We assume a honest but curious provider for a DSaaS provider. Thanks to the assurance module, the DS service provider can check if the clients are acting maliciously, e.g., by storing inconsistent and therefore corrupt data—not even the clients need to be fully trusted to keep the system availble.
- *Storage Node:* Like with SAaaS, this component is responsible for storing fragments. Unlike SAaaS, it is using a different architecture, interface concept, as well as providign additional functionalities not present in the archiving approach. The storage node consists of two main components, a server component and the object store.
  - *Server Component:* This module is an active compute component which manages all communication with clients, data sharing service, and other storage nodes. It runs the various protocols required for the data sharing service ranging from read/write/admin access of clients, the concurrency layer in conjunction with other storage nodes, as well as admin and assurance monitoring with the central sharing service module. For communication with other server components it uses secure connections based on TLS and for communication with clients it supports WebSocket technologies.
  - *Object Store:* general object storage based on Amazon S3 technology. Its inter-

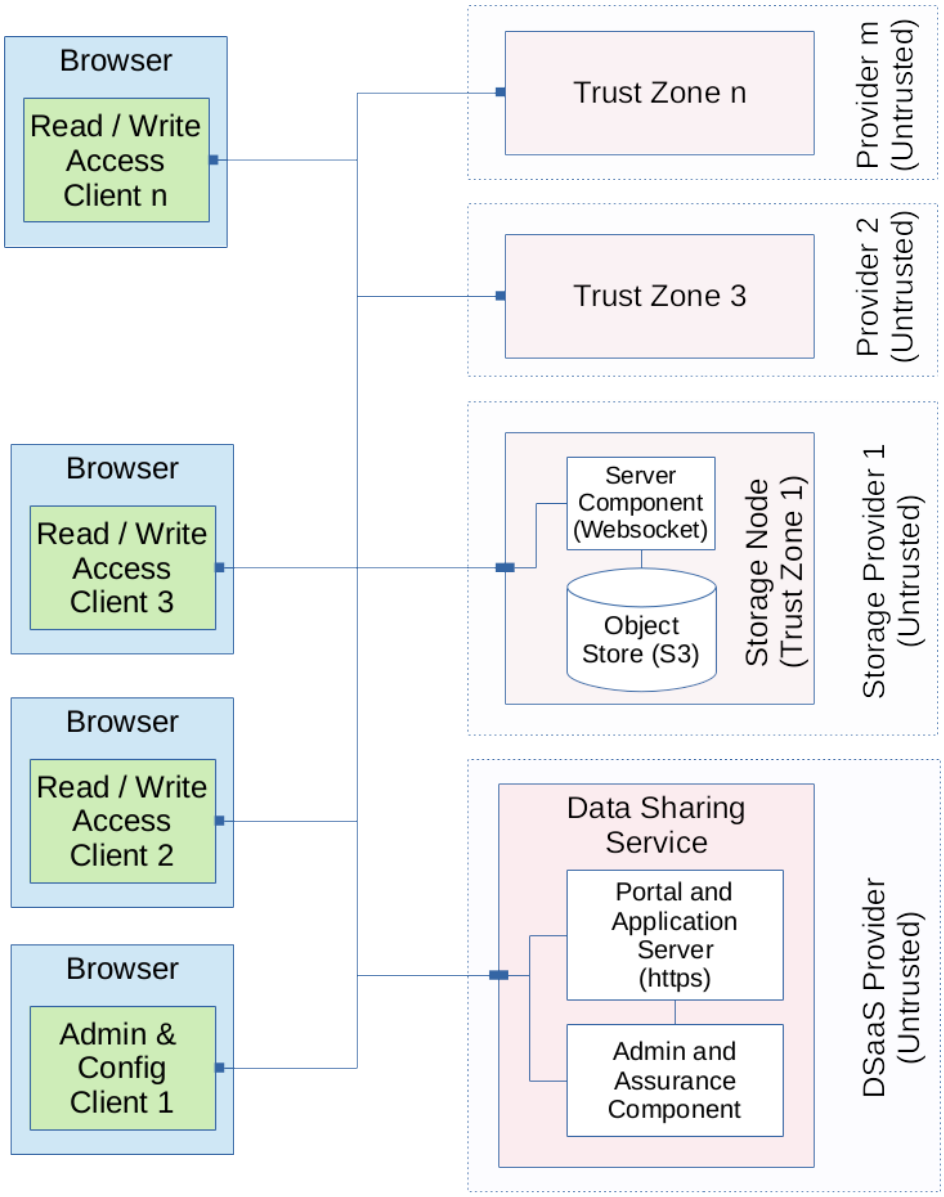


Figure 22: Interaction dynamics for data sharing service.

face is only exposed to the server component and all communication is routed over it. In that sense it only serves as the internal persistent storage.

4.2.5 Provider/Consumer Scope of Control

For our analysis we assume the Data Sharing service being a SaaS level service, quite similar to well known services like Dropbox or the like. It would be also possible for the client to deploy the software on his own and only leverage PaaS providers to run the modules, however, this does not really impact our analysis except that the provider side control is even more reduced. In general, the idea of the service is to maintain a decentralized storage network without single point of trust or failure. The responsibilities for the different layers in the cloud stack is shown in Figure 23.

- The *Cloud Consumer* or cloud service client is the user of the storage service and should be able to upload, mange and share his data in a secure and privacy friendly way. For best security the service is designed such that all crucial steps are done locally in the web browser of the client, i.e., especially the encoding and dispersal of data. The JavaScript code to be executed is delivered by the application server of the DSaaS provider. Additionally, to work with the system credentials and some other configuration data and states have to be stored on the client side. Alternatively, to avoid local storage, important management data can be stored in encrypted form on the DSaaS server or in another wallet. All non-sensitive information about the user configuration of the storage network used is held on the DSaaS side and can be configured there. If the service is offered for a group of people, a dedicated admin will be able to manage the storage configuration for the whole group, e.g., a company using the serivce. The client also controls additional features if made available by the provider, like monitoring and auditing features.
  Additionally, to control options at the application server, certain options have to be set on the BFT-nodes individually. This is necessary for security and privacy reasons, i.e., to avoid the application gateway have full access to plaintext user data. In particular, the client directly manages access control lists for his data and users in his responsibility, if any. He can also define audit trail settings and data remanence times for secure deletion.
- The *DSaaS provider* is the application service provider. He is responsible for hosting and maintaining the main application. Depending on the particular way the service is implemented, he can additionally have the role of brokerage of storage nodes and control according databases and peer to peer services. The DSaaS provider can also offer additional monitoring services like remote data checking or maintenance control. However, no matter how the service is designed in detail, he should have not enough access rights to access data on storage nodes without user consent nor be able to reconstruct plaintext data. The responsibility of the DSaaS provider is on the brokerage, provisioning and procurement of BFT-nodes and the BFT network.
- The *Storage Provider* are responsible for providing the storage space and running the BFT-nodes, which have to be made available to the clients. No special requirements are needed, except for the establishment of some trust relationship with the

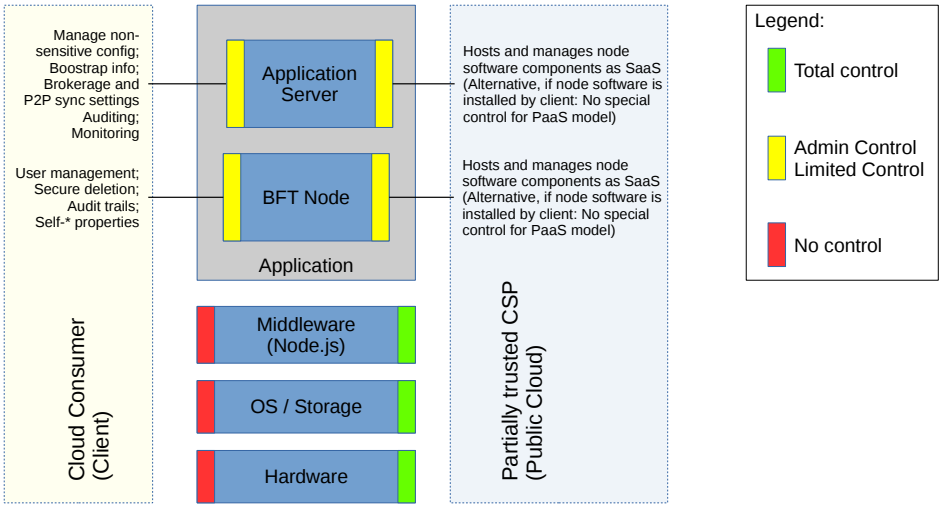


Figure 23: Overview of control scope for main type of stakeholders.

client side additionally to the DSaaS provider. Alternatively, if they are just PaaS providers, the node software can be deployed by the client, but then he has also to do the whole configuration and maintenance of the node software.

4.2.6 Parameters

When secret sharing or information dispersal is applied to increase security and availability in data storage, the threshold configuration parameter k of the sharing algorithm is essential to set the level of security and the redundancy $n - k$ is the main parameter to tune the reliability and availability of the overall virtual storage system. Detailed models to calculate availability and corresponding overhead considerations are discussed in the SECOSTOR tool specification in deliverable D5.3 and more advanced decision support models have been developed and analysed in D7.3 and D7.4. The capability models of D7.3 also present detailed solutions also considering geo-location constraints.

4.2.7 Application Development

Attack Surface

Within the Data Sharing use-case the data-sharing tool is implemented directly within the client. After the client performs the secret-sharing operation, it distributes the generated shares to the different active backend storage servers through an WebSocket interface. Each provider in term is connected to backend storage—this can be directly attached storage or remote storage such as S3. In the latter case credentials for the storage access

will have to be stored at the storage provider.

As this is a new project, we are not encumbered by existing legacy protocols that must be integrated for product integration. This allows for a slim service interface without duplicate operations.

We deploy the application server directly, i.e., without an additional web-application reverse proxy or web-application firewall. There is no distinction between user groups for those operations, i.e., this further reduces the attack surface.

Operations provided between the user client and the storage service are very similar to the backup scenario. The transport mechanism is different though: while Amazon S3 is a HTTP-based protocol, WebSockets are stream-based. In general we assume the following operations to be provided:

- sign in user
- sign out user
- list data buckets
- list files within data bucket
- add file to data bucket
- retrieve file from data bucket
- delete file from data bucket

The communication between the storage service and the storage itself utilizes an even smaller subset of operations (which will be directly performed upon the file-system):

- add file to data bucket
- delete file from data bucket

When it comes to file-based attack-vectors, we have to analyze the client application as well as the storage service. Within the Client credentials for accessing the storage services have to be stored. In our envisioned browser-based setup we will use the browser session storage or HTML5 storage for all credential data. In addition a browser cache might be utilized to improve performance.

On the storage servers we also might have access credentials for the utilized backend storage. As we will employ a BFT-protocol for providing high-availability each storage service needs access credentials for each other storage service.

User-data is processed by secret-sharing by the user client. No plaintext user data ever leaves the original client—this reduces the attack surface against sensitive data tremendously. Figure 24 summarizes our potential attack surface.

Gathering Threats with STRIDE

We utilized the Microsoft Threat Modelling Tool to arrive at Figure 24. The generated graph was the base for automatically generated threats—a first preliminary examination of those resulted in the following table:

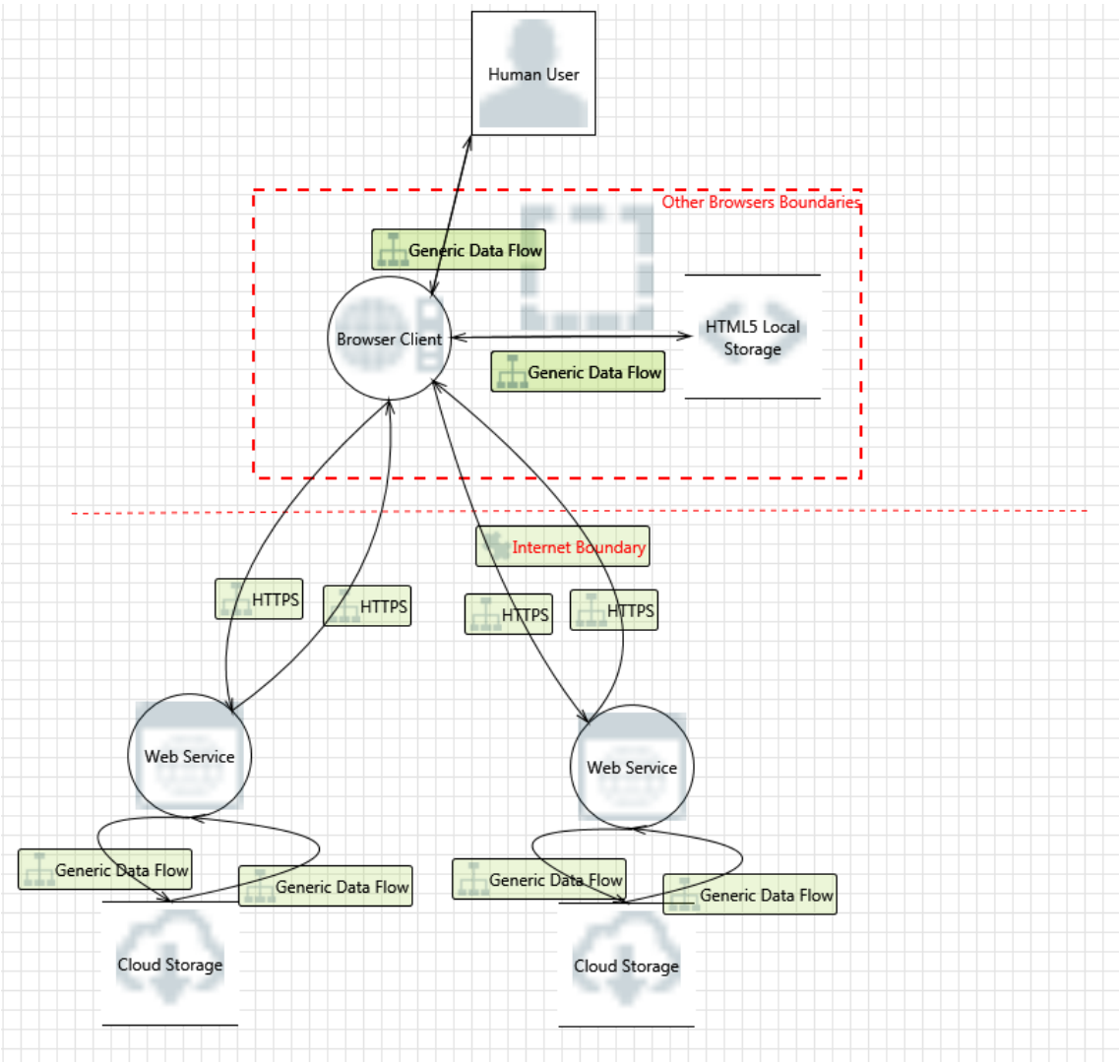


Figure 24: Attack surface model for the data sharing service.

STRIDE Category	Original	Not-Applicable	Mitigated/Design	Resulting
Spoofing	13	5	0	8
Tampering	6	0	0	6
Repudiation	6	0	0	6
Information Disclosure	4	0	0	4
Denial of Service	14	5	0	9
Elevation of Privilege	16	0	0	16
Total	59	10	0	49

When compared to the Secure Archive Scenario the main distinction is the placement of the secret sharing functionality. In this scenario all secret sharing is performed within the client (web-browser). This reduces the risk of data exposure as no unencrypted data ever leaves the client but introduces a potential repudiation problem. With the Secure Backup scenario the Backup Service was a trusted component that was utilized for creating audit logs—in the Secure Sharing scenario we do not have this central component. In contrast we utilize a byzantine fault-tolerant concurrency protocol to achieve distributed auditing capabilities: as all audit data is stored on all connected storage servers we can audit logs though simple majority voting.

Threats focusing around Denial-of-Service attack vectors were implicitly mitigated as there is no central backup/sharing component that could become overloaded. To create a denial-of-service situation, a malicious client would have to overload connected cloud storage servers—their resource consumption will be directly billed to the originating client thus creating a monetary defense against denial-of-service attacks.

This scenario is currently missing the administrative interface.

Analyzing and Mitigating remaining Threats

According to the STRIDE-Analysis Spoofing and Elevation of Privilege are the two largest attack areas. To mitigate Spoofing all network connections will have to be authenticated on the serverside. As the client is communicating directly with the storage servers we will place high importance upon protective measures such as HSTS and proper SSL Validation.

To prevent elevation of privilege based attacks we will harden our client platform through selection of suitable frameworks and libraries. All input data will be validated for malicious code, all data output will be sanitized through central components.

4.2.8 Operational Aspects

The same ideas as for the secure archiving service apply. Because the security relies on the non-collusion assumption — in the pure sharing mode — isolation of fragments have to be maintained as good as possible. Only authorized users should have access to data and no single cloud provider should have enough access rights to recover plaintext information of users.

4.3 Selective Authentic Exchange (SAEaaS)

4.3.1 Overview

The service that allows for Selective Authentic Exchange is build upon the cryptographic primitive of Redactable Signature Schemes (RSS). An RSS allows the signer to specify on signature generation which parts are authorised to be redacted (content removed) subsequently without invalidating the digital signature on the remaining (not redacted) content. This is enabled in the PRISMACLOUD tool called FLEXAUTH, as well as in a hardware security module (HSM) implemented in a field programmable gate array (FPGA) during the course of PRISMACLOUD. Additionally to making use of the tool the service offers additional features regarding the messages it signs, produces and verifies. The FLEXAUTH tool works on messages composed of strings and would not be easily integrated into the message flow of existing services. PRISMACLOUD enhanced the tool in this service to handle (take as input to the sign, redact and verify operations) XML files. As the RSS is based on asymmetric cryptography the service is storing and using cryptographic keys. This is done inside the FLEXAUTH tool as well as inside the HSM. The secret key – which needs to be guarded against unauthorised use – is only needed to generate signatures. It is not needed for redactions, which with the chosen RSS from the FLEXAUTH tool is a public operation. Of course the verification, like with standard signature schemes, would require trusted, but public, signature verification keys, e.g. certified to be linked to a natural or legal person by a public key certificate from a trusted certificate authority (CA). Note, that the secret signature generation key is not supplied to the SAEaaS directly when calling the sign functionality of the SAE part. Instead it is referenced by `UserID` given as a parameter. The part called MOXIS in figure 27 is the FLEXAUTH library, also indicated as the blue box in figure 25. As you can see in the figure, the SAE-service can call either the hardware based implementation or the software library. SAEaaS has the capability to generate, verify and handle redactable signatures for XML files. The three functionalities are sign, redact and verify. An RSS-signed XML document protects the integrity of the selected parts of the XML document. There are two different integrity protections for those parts that are protected by the service:

- If a part that is fixed and can not be subsequently changed in any semantic way without breaking/invalidating the signature. Its position (identified by XPath expression) and its content (the result of the XPath expression). Then this is called a fixed part.
- If a part is admissible then it can be subsequently redacted while preserving a verifying signature that if valid vouches for the integrity and authenticity of origin for the remaining parts that are signed.

The integrity protected parts are selected from the XML document by passing XPath references, similar to the way XMLDSIG [SG07, ERS02] does it. In more detail, what parts of the XML document that are to become protected by the signature are selected by two lists of XPath expressions. In the following example we give the XPath statements, which come from the eHealth pilot; they describe that the personal data related to the diagnosis, the patients information and the doctors name from the XML can be redacted,

while the random document identifier will be protected agains any subsequent modification in the same way a classical digital signature would protect the integrity:

- fixed: `["#xpointer('/ehr/identifier/id')"]`
- redactable: `["#xpointer('/ehr/anagraphic')","#xpointer('/ehr/diagnosis')", "#xpointer('/ehr/sickLeave')", "#xpointer('/ehr/dischargingDoctor')"]`

Different to standard digital signatures, there are two protection classes, thus there are two different lists of XPath expressions.

SAEaaS will return XML messages with an redactable or sanitizable signature embedded into XML, thus the XML becomes a signed XML with an enveloped signature. This is shown in Listing 1.

Listing 1: Example: XML representation of a redactable signature embedded into the XML that is protected

```
<ehrSigned>
  <ehr>
    <identifier>
      <id>fd4426bf-dcb6-42d8-81c6-e7e84b4572d2</id>
      .....
    </identifier>
    <anagraphic>.....</anagraphic>
    <dischargingDoctor>.....</dischargingDoctor>
    <diagnosis>.....</diagnosis>
    <sickLeave>.....</sickLeave>
  </ehr>
  <signature>
    <reference position="1" redactable="false" URI="#xpointer
      ('/ehr/identifier/id')"></reference>
    <reference position="2" redactable="true" URI="#xpointer
      ('/ehr/anagraphic')"></reference>
    <reference position="3" redactable="true" URI="#xpointer
      ('/ehr/diagnosis')"></reference>
    <reference position="4" redactable="true" URI="#xpointer
      ('/ehr/dischargingDoctor')"></reference>
      <signatureValue>
        .....724 characters Base64 encoded......
      </signatureValue>
      <publicParameterId>
        3e8b6b5784ff468faf286f98d69036a9
      </publicParameterId>
      <certificate>
        .....1376 characters Base64 encoded.....
      </certificate>
    </signature>
<ehrSigned>
```

Note, that after the successful execution of the redact operation the information that some part got removed remains, i.e. the scheme does not offer cryptographic transparency [BFF+09], but public non-interactive accountability [BPS13]. Please see Deliverable D4.4 [DDH+16] for more details. However, the content is completely removed and can not be recovered from the remaining parts and the adjusted – still valid – signature. This holds also true for the position inside the XML, i.e. the XPath expression is also deleted.

4.3.2 Key Features

The main advantages of this service are:

- *Increased data privacy:* The patient controls the data that is forwarded. If the patient doesn't want to share all the information of the health record, those parts can be redacted and the third person will not get the information. You can see this reduction of confidentiality protection requirement in the figure 25 when comparing the needed security level for confidentiality between the storage provider in Cloud B to that in Cloud D; the latter receives redacted data which does not contain the information that was redacted and thus might require less protection.

- *Available as SaaS:* This increases the availability for the user. Further, the service is easily accessible from everywhere and anytime. The user only needs internet access and a mobile phone. Further, no additional software on the client devices is needed to sign, redact or verify.

- *Certificate based signatures:* The web application MOXIS is used for signing electronic health records with malleable signatures. For this signature, a certificate containing the secret signature generation key is used and securely stored in a secure certificate store (certStore). This certStore is located on a different server and this server is only working in the background. So, if MOXIS gets attacked the private keys are still secure. Which means that the computed signature is a so called advanced signature and legally valid [PH11, vGPFH15, AFHP+15].

- *Strong authentication:* In order to guarantee that the person is really the person he/she pretends to be a strong authentication is used. For accessing the signature platform and the trusted cloud platform where the patient can verify signatures and redact documents. So, the patient can be sure it was really this doctor who signed the document and if the patient forwards the document the receiving person can be sure that the document was only modified by the patient.

- *Concurrent access:* Many doctors can sign at the same time and many patients can redact and verify documents also simultaneously.

- *Decoupling of signing and redacting and verifying:* The patient can collect electronic health information that is authenticity and integrity protected by the doctors' sig-

natures from many hospital information systems. The signatures can be verified on original or correctly redacted documents by any third party with a trusted signature verification key of the doctor. This removes the need to trust the storing an handling cloud service provider for the integrity protection, any violation of integrity can be detected by a failing signature verification. More crucial for scalability and privacy is that the patient can, without any interaction with the doctor or the hospital information system (HIS), carry out the redaction (non-interactive). This means many patients can redact and verify documents without the HIS being involved.

- *Decoupling of functional steps into micro-services:* As the three main functionalities require different set of cryptographic keys, we developed them as micro-services. For example the redact service can operate without the need of even public keys. Also the linearisation does depend on the data structure to be protected, not on the algorithm. This makes many of the individual micro-services adaptable for use in multiple contexts. Also they can be exchanged and developed independently[10].

But the service faces also some weaknesses:

- *Many parties involved:* PRISMACLOUD decided to realise this service as a set of micro-services. Hence, until the receiving person gets the document, the document passes many services. First the hospital proxy which picks up the XML document from the hospital information system (HIS) and sends it to the SAE-service. The SAE-service linearizes the document and forwards it to MOXIS. The document gets signed and then it is send back to the SAE-service. The SAE-service Integrates the signature value to the xml-file and sends it to the trusted cloud platform where the user can verify, redact and forward the document to the person who should get the document. This distribution causes a lot of risks which can be seen in the amount of threats that were discovered by the threat modelling tool.

- *Denial of Service:* As so many components are involved a denial of service attack would interrupt the whole workflow.

In Figure 26 we show a SWOT analysis of the given service.

4.3.3 Usage Model and Stakeholders

In Figure 25 we show the foreseen deployment of the different functionalities of the service and the entities interacting with them.

4.3.4 Service Model and Interaction Dynamics

This subchapter explains the main components of this service and how they interact.

- *Client:* There are no special needs for the client as this service is a web application

[10]This worked well; SAEaaS involved the interaction of FCSR, ATOS, XiTRUST and UNI PASSAU.

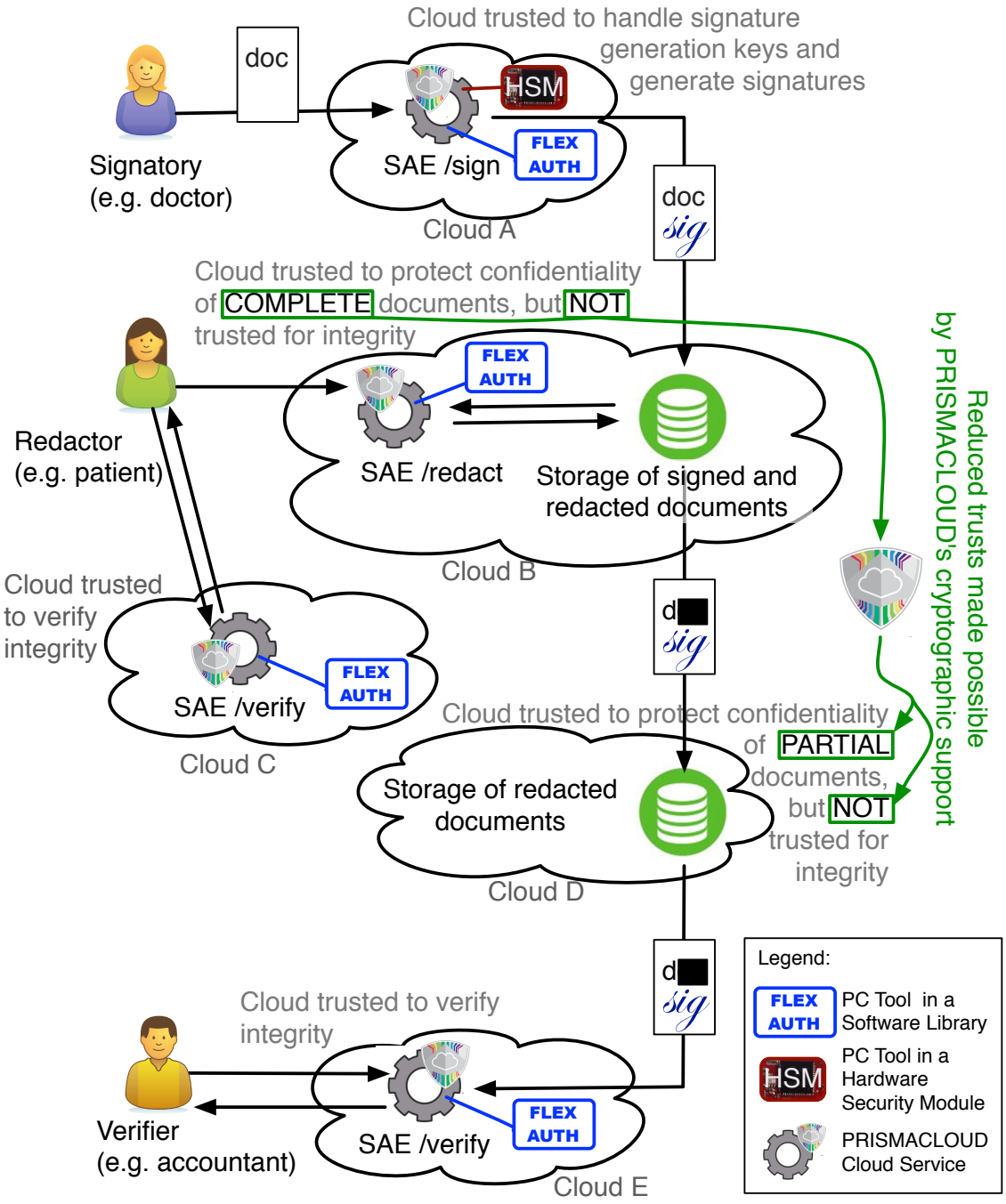


Figure 25: Deployment and actors of the selective authentic exchange service

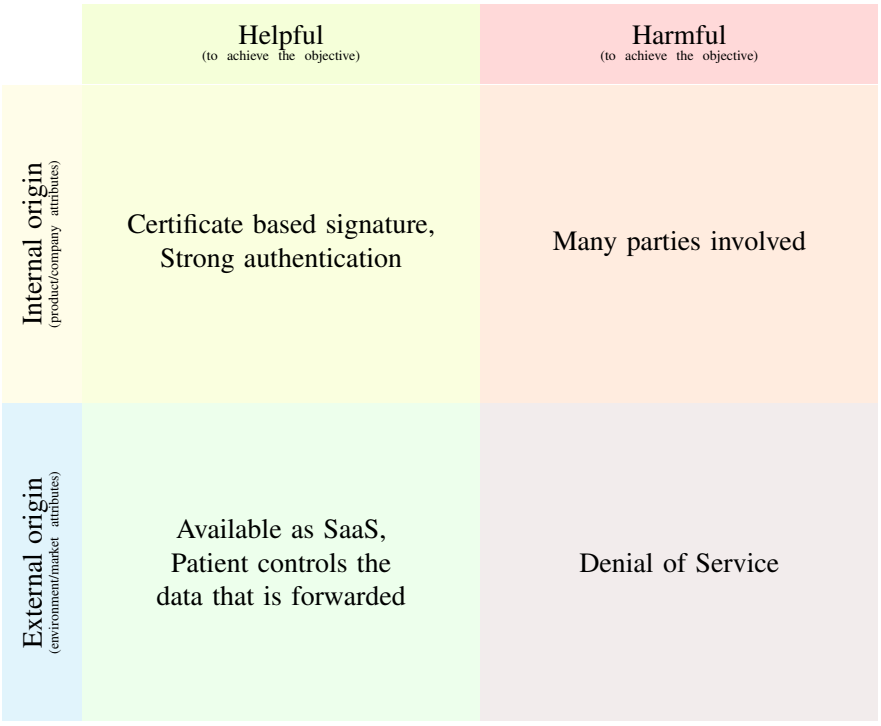


Figure 26: SWOT for the selective authentic exchange service

and can easily be accessed with any internet browser such as chrome, firefox, edge. Of course if the client does not want to entrust the correct execution of the service, it could run locally; assuming the required keys are locally available. Note, only for signature generation a secret key is required.

- *Hospital Proxy:* The hospital proxy sends Health Level Seven International (HL7)[11] messages in an XML format to the SAE-service via web service.

- *SAE-Service:* The SAE-service takes the XML-documents and linearizes the document so the malleable signature library is able to sign the document. After the document has got signed, the signature value is added to the XML-file and send to the eHealth cloud platform. The linearization of the document also happens when the service gets the document for redaction or verification purposes.

- *MOXIS:* XiTrust MOXIS receives the electronic health record, or more precisely the linearized XML data, which needs to be signed from the SAE-service. Therefore, the signerID, the documentID, the linearized messages, the original XML and paths

[11] Health Level Seven International (HL7) is a not-for-profit, ANSI-accredited standards developing organization dedicated to providing a comprehensive framework and related standards for the exchange, integration, sharing, and retrieval of electronic health information that supports clinical practice and the management, delivery and evaluation of health services; www.hl7.org [last accessed: Jun. 2017]

to the parts that are redactable are send via the SAEToMOXIS REST interface. The XML data, which gets signed is converted into a user-friendly pdf-document in order to visualize it to the doctor. The parts, which are redactable, are thereby marked with borders in colour. So, the doctor can control which parts are redactable and signs it if the document is fine. The bulk signature enables the doctor to sign all the controlled pdfs by entering the signature password only once. For the signing process, the malleable signature library is used and for calling the sign-function, the private key and the linearized messages are provided as parameters.
After the messages are signed, the signature, the original XML, the certificate as well as the documentID and the publicParameterID are send back to the SAE-service via the MOXISToSAE REST interface. The doctor has the possibility to control the document again and sees a visualization on the document, if the signature was computed successfully. The signature time is also available and visible. The signature visualization can be chosen by the doctor.
For the authentication of the doctor a two-factor-authentication is provided in order to be sure that he/she is really the person he/she pretends to be. This is done by entering the own mobile phone number plus the signature password and after a few second a one-time password is received on this mobile phone which needs to be entered in a second step.

In addition, the SAEToMOXIS interface provides methods, which are used to verify the malleable signature created with MOXIS or to redact parts of the signed data. These interfaces are handled synchronous as those functions are executed automatically when the user chooses this functionality at the eHealth cloud platform.

- *eHealth cloud platform:* This platform provides the user with a graphical user interface where the user can decide which documents he/she wants to redact or verify. In a next step, the user has the possibility to share the document with other people. The authentication at this platform is carried out with the same two-factor authentication procedure as with MOXIS.

This is also depicted in the interaction diagram shown in Deliverable D7.8, which especially focusses on the interaction between the two micro-services (SAE-Service and MOXIS).

4.3.5 Provider/Consumer Scope of Control

The deployment of the selective authentic exchange service involves the following parties:

- *Cloud Consumer:* The cloud consumers are the patients as well as other healthcare employees. The cloud consumer scope of control is reduced to a minimum. The consumer doesn't need to install or operate any software as the consumer only needs an internet browser to access the own health records and redact them or verify the signature of the doctor.

- *SAE Providers:* These providers are running the different or all functionality of the

SAE-service and are responsible for the correct transformation from XML-documents to linearized messages.

- *Tool Provider:* The tool provider is responsible for correctly calling the sign, redact and verify functions of the malleable signature library and for the sign functionality this requires the private key of the doctors. Further, the user management to restrict access to that secret key's usage is done by the tool provider as well. For the verification functionality still a trusted public key of the doctor is required to be maintained by the tool provider. Finally, the redact functionality does not require any keys.

- *Storage Providers:* This are several or one IaaS provider providing the real storage space where the unsigned, signed or redacted data will be stored.

4.3.6 Parameters

From the parameters the following four are most important:

- The userID for which the MOXIS is able to contact the natural person to authorise the usage of the natural person's secret signature generation key by the signature algorithm.
- The XML document of which certain defined parts shall be protected against any subsequent modification while some other defined parts shall be redactable by a RSS signature.
- A list of XPath expressions that define each redactable part (for security reasons only selection by absolute paths are allowed).
- A list of XPath expressions that define each fixed – non-redactable – part (for security reasons only selection by absolute paths are allowed).

4.3.7 Application Development

We utilized the Microsoft Threat Modelling Tool to retrieve Figure 27. A first preliminary examination of the threat report resulted in the following table:

STRIDE Category	Original	Not-Applicable	Mitigated/Design	Resulting
Spoofing	7	0	4	3
Tampering	3	2	0	1
Repudiation	7	0	4	3
Information Disclosure	0	0	0	0
Denial of Service	14	0	0	14
Elevation of Privilege	29	2	17	10
Total	60	4	25	31

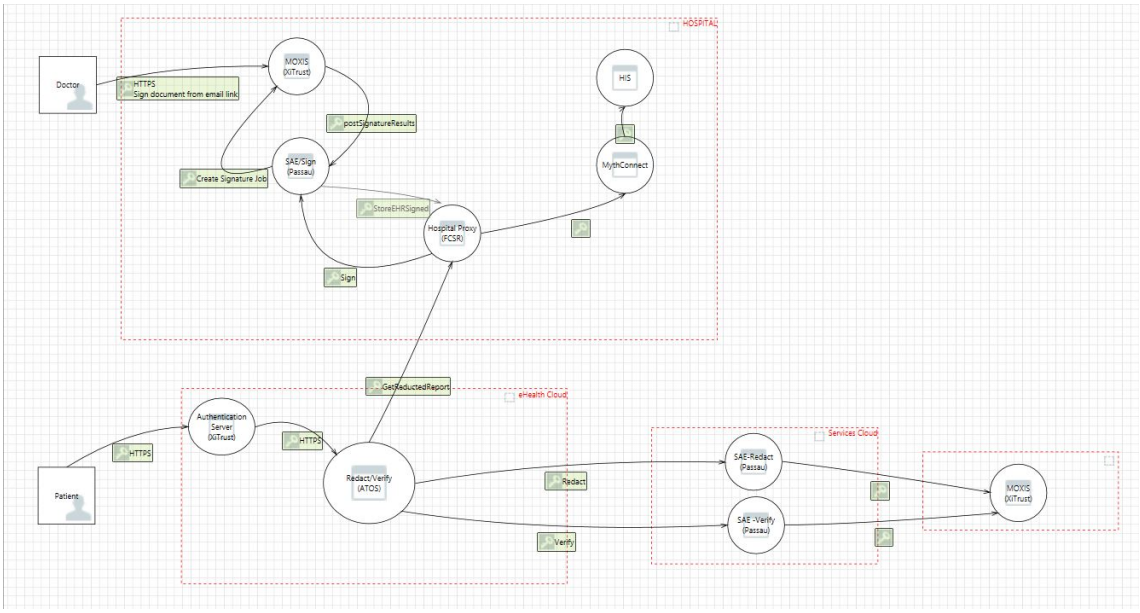


Figure 27: Attack surface model for the selective authentic exchange service

The most threats were found within the category elevation of privilege but most of the threats can be mitigated or are not applicable. The reason why most of them can be mitigated is that client authentication is in place for all the web interfaces and all the received data is validated before some code gets executed. Furthermore, a security audit for MOXIS is performed regularly by a trusted third party.
The spoofing threats are addressed with a two-factor authentication using the mobile phone. So, the attacker would need the mobile phone and the signature password of a registered user of MOXIS or from the Authentication Server in order to spoof the system. The threats, which are categorized as tampering are not applicable as the components don't get that much access to memory. Further, the repudiation threats can be mitigated as all the activities are logged.
We have carefully designed the XML support of SAE-service to not need to handle any keys. Further, the service restricts the selection of parts of the XML to absolute Xpaths using xpointer expressions in order to be not vulnerable to known wrapping attacks on XML signatures, like [MA05]. Hence, there is a restriction to the absolute paths, i.e., XPath has to start with the XML root (/).

4.4 Privacy Enhancing IDM (PIDMaaS)

4.4.1 Overview

This service offers the capability of a privacy enhanced identity management. In particular, it allows users to store their attribute credentials obtained from some entity in this component and to realize a selective attribute disclosure functionality. Therefore, the main

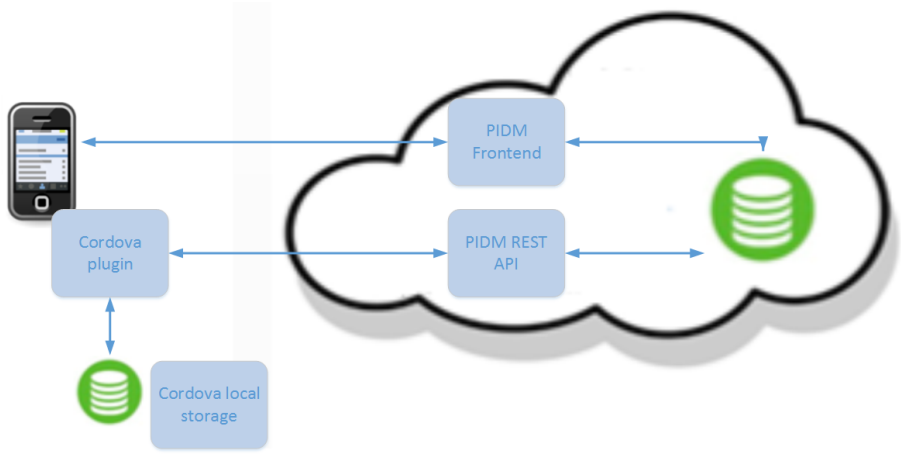


Figure 28: Privacy Enhancing IdM (PIDM) overview.

goal of this service is the preserving of users' privacy. That is, it offers client applications the possibility to manage their pool of users in a privacy-preserving manner. Operations needing authorization and susceptible of revealing sensible information about the users can be secured by setting a group-based permission system where individuals provide an anonymous proof of their inclusion in a group (which may be linked to a certain set of permissions), which is used by the application to authorize the action.

The Privacy Enhancing IdM service (PIDMaaS) is built on top of the Flexible Authentication with Selective Disclosure Tool. Since the service involves the generation of secrets that are bound to individual users, the complete service is provided in the form of two modules:

- Cloud service: offers application-wide functionalities, including:
  - Group management: users of the platform can manage their groups.
  - User management: users of the platform can manage their users and their association to groups.
  - Signature management: includes functionalities for verifying, linking or opening signatures.
- Cordova plugin for Android apps: offers those functionalities related to the operations that involve the secret keys of the individual users, and therefore need to be computed on their personal devices:
  - Signature management: includes functionalities for creating personal keys, signing documents and associating a user to a group (in interaction with the cloud service).

4.4.2 Key Features

In Figure 29 we show a SWOT analysis of the given service. As can be seen in figure 29, the use of Privacy Enhancing IdM service and its location in the cloud have the following

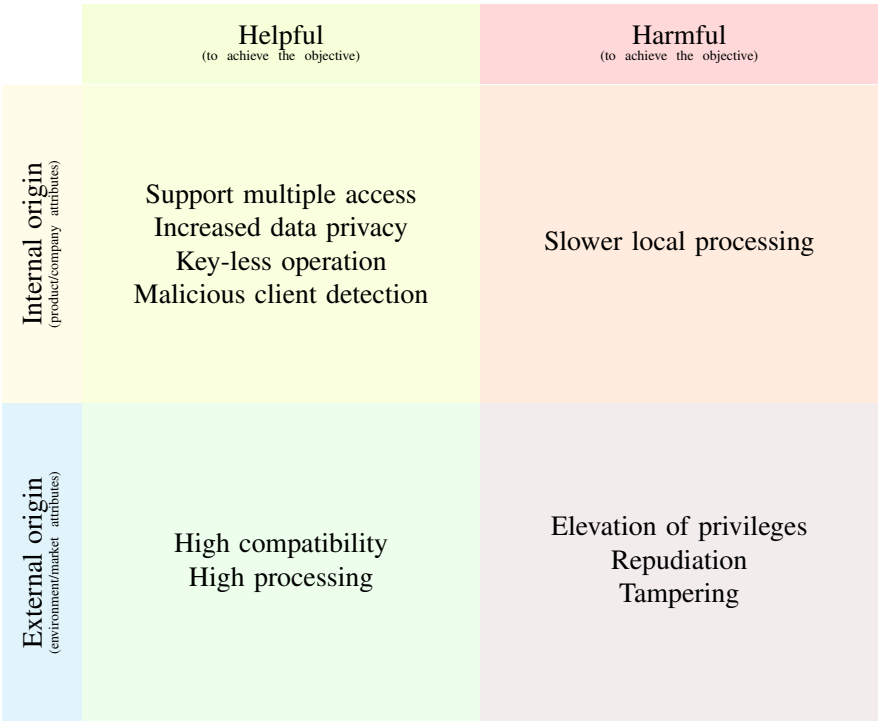


Figure 29: SWOT analysis of Privacy Enhancing IdM Service (PIDMaaS).

advantages:

- **User anonymity:** Registered users can obtain a secret token which allows them to authenticate to the service as a member of some group of authorized users without revealing their identity. This enables applications to enforce authorization without violating the users' privacy.
- **Unlinkability of interactions:** The service allows users to authenticate in a way that different authentications of the same user cannot be linked together. This prevents an application from tracing users.
- **Privacy-friendly misuse detection:** The service when given a special detection information can detect whether a user provides a clone of the secret token to another users to authenticate at the same time. This can be performed in a privacy-preserving way by the service, i.e., without the need to identify the respective user.
- **Reidentification of misbehavers:** The service when given a special re-identification information can re-identify misbehaving users. Thus, misbehaving users can be penalized.

In the opposite way, the main weak is the Slower local processing, due the cordova plugin is not native. As a result, the processing is slower than the case of native code.

The threats has been detected by using the Microsoft Threat Modelling Tool:

- *Elevation of privileges:* Cross-site request forgery (CSRF or XSRF) is a type of attack in which an attacker forces a user's browser to make a forged request to a vulnerable site by exploiting an existing trust relationship between the browser and the vulnerable web site. In a simple scenario, a user is logged in to web site A using a cookie as a credential. The other browses to web site B. Web site B returns a page with a hidden form that posts to web site A. Since the browser will carry the user's cookie to web site A, web site B now can take any action on web site A, for example, adding an admin to an account. The attack can be used to exploit any requests that the browser automatically authenticates, e.g. by session cookie, integrated authentication, IP whitelisting, ... The attack can be carried out in many ways such as by luring the victim to a site under control of the attacker, getting the user to click a link in a phishing email, or hacking a reputable web site that the victim will visit. The issue can only be resolved on the server side by requiring that all authenticated state-changing requests include an additional piece of secret payload (canary or CSRF token) which is known only to the legitimate web site and the browser and which is protected in transit through SSL/TLS. See the Forgery Protection property on the flow stencil for a list of mitigations.
- *Repudiation:* A repudiation attack happens when an application or system does not adopt controls to properly track and log users' actions, thus permitting malicious manipulation or forging the identification of new actions. This attack can be used to change the authoring information of actions executed by a malicious user in order to log wrong data to log files. Its usage can be extended to general data manipulation in the name of others, in a similar manner as spoofing mail messages. If this attack takes place, the data stored on log files can be considered invalid or misleading.
- *Tampering:* Attackers who can send a series of packets or messages may be able to overlap data. For example, packet 1 may be 100 bytes starting at offset 0. Packet 2 may be 100 bytes starting at offset 25. Packet 2 will overwrite 75 bytes of packet 1.

4.4.3 Usage Model and Stakeholders

Due the service is composed of two modules, besides a provider and a customer, we have considered one extra provider belonging to it. Therefore, three actors take part in the service:

- The *Cloud Customer*, or client, is the part that generates users' keys (public and private), groups joins and signatures.
- The *Privacy Enhancing IdM Provider* runs the Privacy Enhancing IdM service and offers it to Cloud Customers. It manages the execution of the Privacy Enhancing IdM software. It is necessary that a trust relation exists between the Cloud Customer and the Privacy Enhancing IdM Provider.
- The *Cloud Provider* manages the cloud infrastructure where the applications of the Cloud Customers are deployed. The Privacy Enhancing IdM service manages groups and users registration and verifies signatures.

Figure 30: Diagram of the PIDMaaS elements.

4.4.4 Service Model and Interaction Dynamics

In this section, we describe the main components of the service and how they are related. This can be shown on picture 30.

In the client side, an **android lib** is running. This lib composes the cloud customer and its main functionalities are:

- User key generation
- Group joining
- Sign

That is, in the mobile part of the service, users can create public and private keys, ask for a group joining and sign operations. Apart of the own user keys, this side will create public and private keys related to each group that an user joins to.

In the provider side, the *Cloud service* is oriented to group management. Its functionalities are:

- Application management
- Group management
- User registration
- Verification:
  - Verification of user's signature (*Verification*).
  - Identification of signatures delivered by same user (*Link*).
  - Identification of signing user (*Open*).

4.4.5 Provider/Consumer Scope of Control

The model of this service is PaaS and the service is deployed in a public cloud. In the Provider/Consumer scope of control the actors are the following:

- The Cloud consumer has the control over the user's key store, due it generates public and private keys related to user and groups which user belongs to.
- The Cloud provider manages the cloud infrastructure where the applications of the Cloud Customers are deployed.

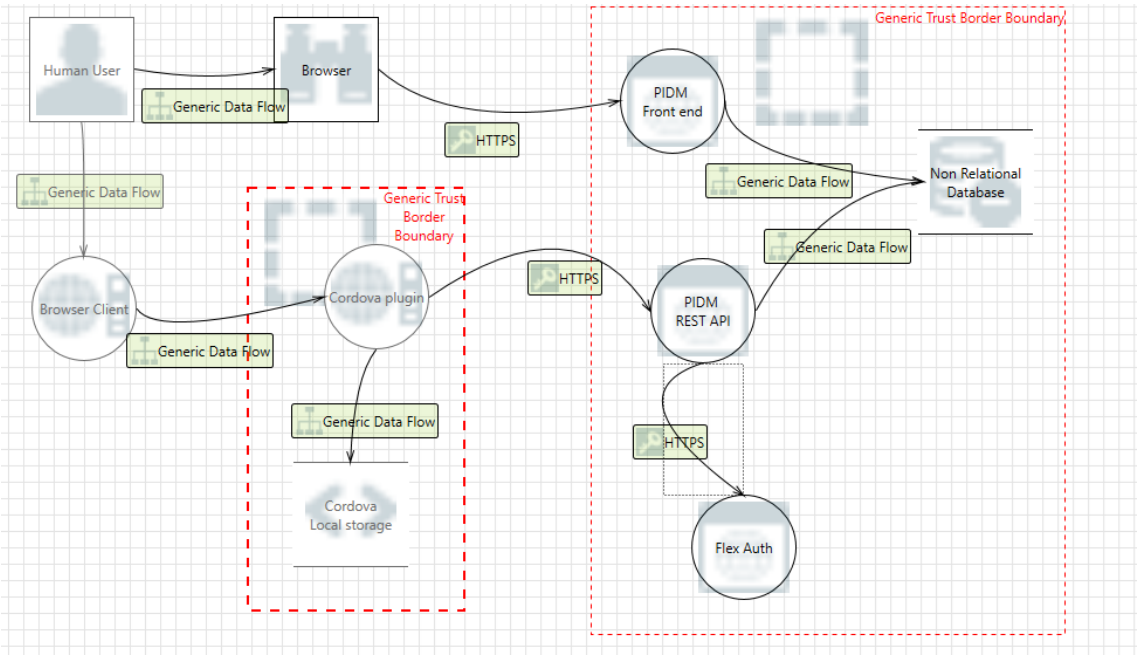


Figure 31: Attack surface model for the Privacy enhancing IDM service.

4.4.6 Parameters

The PIDM service will consider two parameters:

- The current location of the user.
- The proof of permission to park in the area.

4.4.7 Application Development

Attack Surface Analysis

The PIDM service has these two interaction points to the Cloud Customers:

- The *PIDM frontend* is a set of webservices that allows user execute main operations using his/her proof of credentials.
- The *PIDM REST API* provides the part of the service related to the user's keys and signatures.

The rest of the interaction among components in the service are internal, so they are under control of the environment. The figure 31 represents the attack surface model for the service

Gathering threats with STRIDE

We utilized the Microsoft Threat Modelling Tool to arrive at Figure 31. The generated graph was the base for automatically generated threats—a first preliminary examination

of those resulted in the following table:

STRIDE Category	Original	Not-Applicable	Mitigated/Design	Resulting
Denial of Service	9	7	2	0
Elevation of Privilege	12	4	6	2
Information Disclosure	5	3	2	0
Repudiation	3	0	0	3
Spoofing	4	4	0	0
Tampering	12	7	4	1
Total	45	25	12	8

The complete list of threats can be found in the appendix in Section B.4.

Analyzing and Mitigating remaining Threats

Summarizing, main threats that need further investigation deal with:

- *Elevation of privileges:* the implementation of the service needs to be tested to confirm the non-existence of problems related to XSS and remotely execution of code .
- *Repudiation:* proper auditing mechanisms are advised to be included in the service, in order to detect unfair customers.
- *Tampering:* it is necessary to check posibility of collision attacks.

4.5 Verifiable Statistics (VSaaS)

4.5.1 Overview

This chapter is devoted to detail the main functionalities provided by the Service Cloud that provides the Verifiable Statistics functionality. These functionalities are based on a set of state of the art cryptographic primitives that allow the delegation of computations on outsourced data to third parties so that the data owner and/or other third parties can verify that the outcome has been computed correctly. Nevertheless, the details regarding the primitives used for the development of this service cloud can be consulted in detail within the deliverable *D5.8 Overview of verifiable computing techniques providing private and public verification.*

The idea behind this service is two provide the end user with the necessary resources and functionalities that permit the checking of the veracity and authenticity of computed data. Describing this process from the end to the beginning the computed data will need to be verified and in order to do so the data will need some associated metadata to allow the verification or, in other words, the computed data must be signed. In order to produce this signed computed data, it is necessary to have original data electronically signed in the first place. Therefore, to enable this service a public key infrastructure is also needed.

4.5.2 Key Features

As stated in the previous section, the key feature of this service it to give users access to *trusted computations in the cloud* by leveraging verifiable computing. This enables outsourcing of computations to less trusted third parties while still providing *strong guarantees about the integrity and authenticity of computation outcomes* on user data.

The verifiable computing concept is a relatively young concept but in its short life has made huge academic improvements, especially in the last few years. However, at the moment of applying these new improvements to a real environment there is a constraint to take into consideration: the computational overhead of the server which performs the computation.

Hence the main research efforts devoted within PrismaCloud have been applied to reduce and minimize this constraint but always preserving the integrity and confidentiality of the data. In this sense new advances in fields like FHE, pairings, multi-linear maps, circuit generation, or garbled circuits will each benefit the state of the art verifiable computing. Note that so far there is only one scheme where both the time required for generation and verification is $O(T)$ with T as the time required to compute the function.

In addition, privacy has been one of the main aims in the research; PrismaCloud has searched to find the best suitable combination of verifiable computing schemes which preserve privacy and secure against adaptive adversaries. However, there are no instantiations so far that allow building a construction that at the same time is secure in the strong adversary model and provides efficiency and privacy. For many applications, such a primitive would be very valuable. Thus, developing a corresponding solution is an interesting task for future work. Further information regarding the particularities of the different algorithms and schemata selected can be found within *D5.8 Overview of verifiable computing techniques providing private and public verification.*

4.5.3 Usage model and stakeholders

Figure 32 shows an architecture diagram of the service. It shows the different actors and stakeholders involved in the use of the service.

It is worth mentioning that the functionality provided by the Service Cloud can be classified into trusted services (in green) or non-trusted services (in red). Hence the details are as follows:

- *The non-trusted cloud provider:* it allows the end user to compute the data passed as a parameter. This data must be already signed with the proper verification code. As a result of this operation, the service will respond with signed computed information (the compute data plus the verification computing information) to allow the verification of the data later on. It is worth mentioning that this operation is performed on the fly, therefore this operation doesn't need to store any information within the cloud.
- *The trusted cloud provider:* the main functionality provided by this service is the

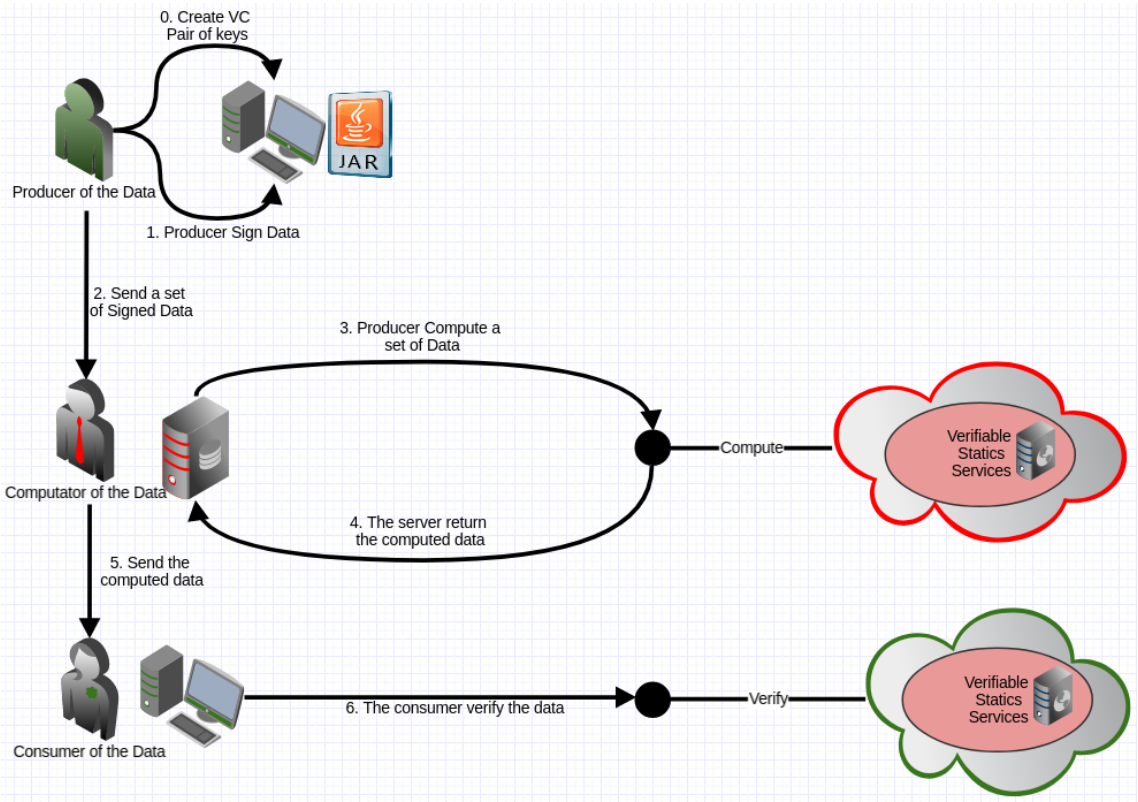


Figure 32: Verifiable Statistics Service (VSaaS) in the Service Cloud Architecture.

verification of the data. For that propose the service takes as a parameter the signed computed data and will answer with "true" if the data is valid and "false" if it is not.

- *The producer of the data:* it will not be part of the service cloud system itself as it requires the use of public and private keys that must be in possession of the owner and therefore must be performed within a trusted environment. However, the service will provide a tool package that will allow the producer to work locally where all the necessary operations are provided.

4.5.4 Service Model and Interaction Dynamics

As can be seen in figure 32, shown in the previous section, the iterations within the system is as follows:

1. *Create Verify Computing Pair of keys:* as a previous step the producer of the data needs to produce a pair of keys with a specific schema, using the package tool provided by the service
2. *Producer Sign Data:* using the pair of keys generated in the previous step and signed by a certificate authority (who will assure the identity of the user), the producer of the data can sign the data using the tool package provided by the service
3. *Send a set of Signed Data:* The producer of the data can then send a set of the signed data to a third party to outsource the computation of the data.
4. *Producer Compute a set of Data:* the entity in charge of the computation of the data can use the remote cloud service in order to compute the set of signed data.
5. *The server return the computed data:* the service, as a result of the operation, will respond with signed computed data.
6. *Send the computed data:* Once the data is computed, the entity responsible for the computation can send the signed computed data to a third entity.
7. *The consumer verifies the data:* Once the consumer of the data has the signed computed data in his/her possession, he/she can verify the integrity and validity of it at any time. In order to do so, the consumer can use the service available, who will answer with a "true" if the data is valid and with a "false" if it is not.

4.5.5 Provider/Consumer Scope of Control

It is worth mentioning that all the operations provided by the cloud are performed on the fly. In other words, the cloud will never make a persistent storage of the information computed or verified. In addition, the services will be agnostic regarding the treatment of the information. From the point of view of the service itself, there won't be any difference between sensitive or non-sensitive information as it operates with the information as a black box without looking at it. Another important characteristic of the service is that it will be used on demand and therefore it won't be necessary to implement any real time (near to real time) access. The only possible damage of a break down of the service will be the company reputation or images, as it won't carry on any loss of data. Hence the

liabilities associated are very limited. In any case, with regards to the Provider/Consumer scope, the description of the different actors involved in the service is as follows:

- *Cloud Consumer:* the end user of the services will not need any specific control or management tool of the service cloud. He/She can use the services, on demand, using the Graphical user Interface provided by the cloud or using the REST API if necessary.
- *Cloud Provider:* As mentioned before, the stopping or failing of the service won't have any associated loss or wear and tear of the data. Nevertheless, in order to maintain the company's reputation, the cloud provider should integrate the resources necessary to monitor the status of the service and to ensure that it will be in service.

4.5.6 Application Development

This section is devoted to detail the result of the application of the STRIDE analysis performed with the Microsoft Threat Modeling Tool. The main aim of this analysis is to have a clear idea of the main threats that the particular system can be exposed to followed by the identification of the different threats which security controls can be used to palliate or minimize them.

Attack Surface Analysis

In the particular case of the VSaaS two different environments can be defined: local and remote. The local environment is considered as out of the system for this particular case, therefore this analysis will be focused on remote access. On the other hand, it is worth highlighting that, as the results from threat modeling show, there are two different environments that must be trusted: the environment running the sign operation and the other for the verify.

As shown in the figure 33, the weaker part of the system will be in the communication with a non-trusted environment and so that is where our analysis is more incisive. In this particular case the focus of the analysis has been done in the computed operation (signed data) as shown in 34.

Gathering Threats with STRIDE

As described in the previous section the weaker part of the system and therefore the principal part of the STRIDE analysis is the computation call. This section is devoted to detail all the threats found for this particular case and, in case of being necessary, to propose some possible controls.

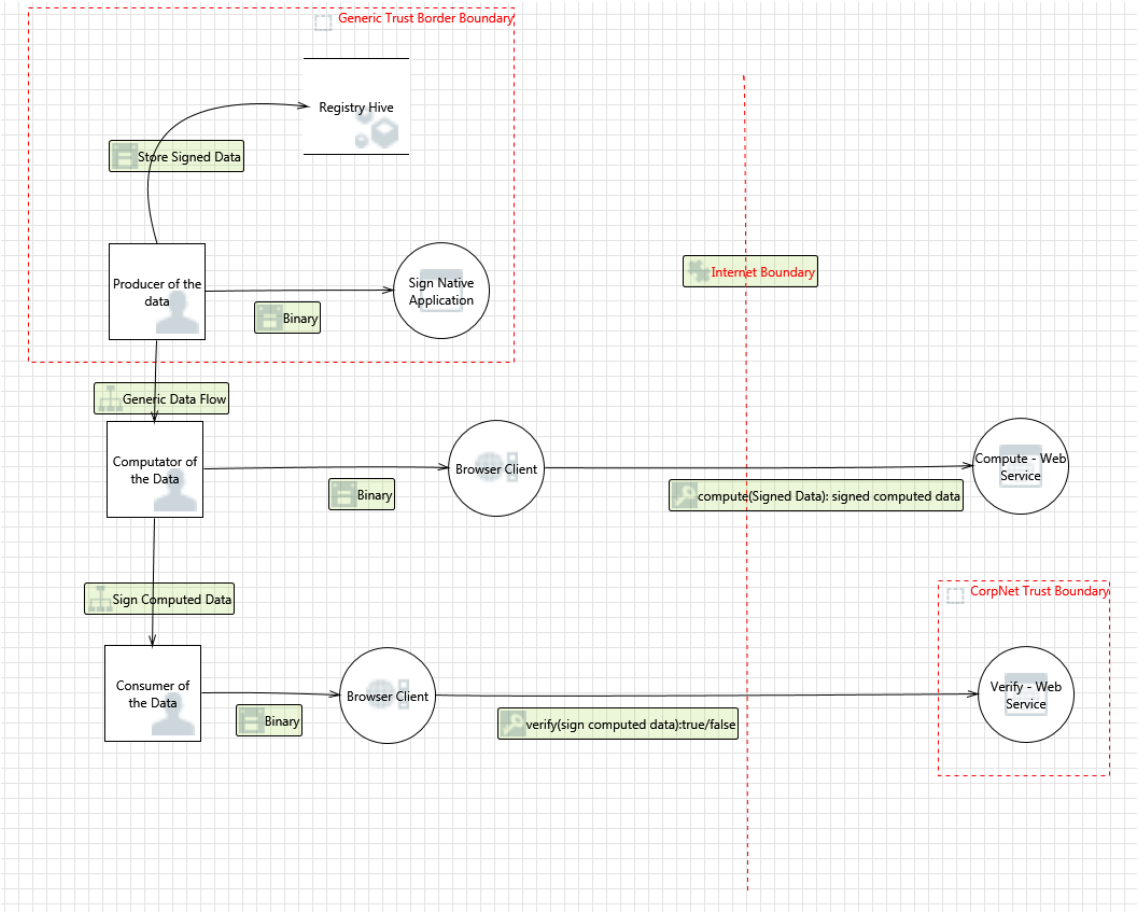


Figure 33: Attack surface model for the verify computing service.

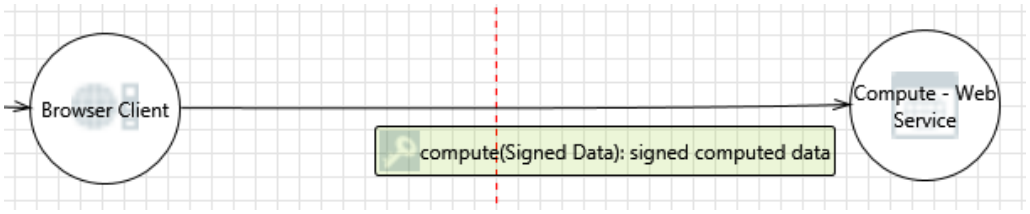


Figure 34: Interaction: compute(Signed Data):signed compute data

All major findings are summarized in the following tables:

Spoofing the Browser Client Process	State: Mitigate
Category	Spoofing
Description	Browser Client may be spoofed by an attacker and this may lead to unauthorized access to Compute - Web Service. Consider using a standard authentication mechanism to identify the source process
Justification	Use of encrypted communications (HTTPS), will minimize the risk associated to this threat

Browser Client Process Memory Tampered	State: Not Applicable
Category	Tampering
Description	If a Browser Client is given access to memory, such as shared memory or pointers, or is given the ability to control what the Compute - Web Service executes (for example, passing back a function pointer.), then the Browser Client can tamper with the Compute - Web Service. Consider if the function could work with less access to memory, such as passing data rather than pointers. Copy in data provided, and then validate it.
Justification	It is not applicable as in this particular case, any information resulting from the different operations will not be shown in the browser, will be downloaded directly to a file

Potential Data Repudiation by Web Service	State: Not Applicable
Category	Repudiation
Description	Compute - Web Service claims that it did not receive data from a source outside the trust boundary. Consider using logging or auditing to record the source, time, and summary of the received data
Justification	As the use of these services are not associated with personal use it won't be necessary to audit the operations

Potential Process Crash or Stop for Web Service	State: Mitigate
Category	Denial Of Service
Description	Compute - Web Service crashes, halts, stops or runs slowly; in all cases violating an availability metric
Justification	A contingency plan with the specific countermeasures has been specified

Data Flow Compute Signed Data Is Potentially Interrupted	State: Mitigate
Category	Denial Of Service
Description	An external agent interrupts data flowing across a trust boundary in either direction
Justification	The encrypted communication (HTTPS) of the operations will minimize the risk associated to this threat.

Web Service May be Subject to Elevation of Privilege Using Remote Code Execution	State: Mitigate
Category	Elevation Of Privilege
Description	Browser Client may be able to remotely execute code for Compute - Web Service
Justification	This service will not be based on an Authorization Service; therefore, it will be not applicable

Elevation by Changing the Execution Flow in Web Service	State: Not Applicable
Category	Elevation Of Privilege
Description	An attacker may pass data into Compute - Web Service in order to change the flow of program execution within Compute - Web Service to the attacker's choosing
Justification	This service will not be based on an Authorization Service; therefore, it will be not applicable

Analysing and Mitigating Remaining Threats

There are no significant remaining threats as seen as a result of the STRIDE analysis detailed above.

4.6 Infrastructure Auditing (IAaaS)

4.6.1 Overview

The infrastructure auditing service (IA or IAaaS) offers the capability for cloud providers to certify their infrastructure and to prove properties of their infrastructure to tenants. In this section, we initially introduce key features, explain the proposed service view, discuss functional requirements and finally focus upon the security impact of our use-case prototype.

The main idea of the infrastructure auditing service is to provide a way for cloud providers to request an auditor to sign their cloud infrastructure and prove policy predicates to tenants without disclosing the layout of their infrastructures. In the e-government use case it is required that the physical servers or virtual machines are hosted in separated locations. This requirement is fulfilled in our prototype by using a geo-location separation service which offers an interface for the tenant to ask the cloud provider, for instance, if the virtual machines are in different locations.

The key design concept was to separate the main functionalities into smaller self reliant services instead of having a monolithic infrastructure auditing service. This way the responsibilities of each service are clearer and we can tune the configuration of each of the

services according to their needs. Therefore, we are decomposing the service into three different services following the micro-service paradigm.

4.6.2 Key Features

In Figure 35 we show a SWOT analysis of the Infrastructure Auditing service. According to the SWOT analysis the service has the following advantages:

- **Reduction of costs:** Using the infrastructure auditing service enables auditors to automate the certification of cloud infrastructures. By making sure that SLAs and especially security objectives are fulfilled, tenants can reduce their costs by utilizing cloud infrastructures and deploying their services to the cloud resulting in a reduction of their costs.
- **Privacy:** The service allows a cloud provider to prove properties on a cloud topology to a tenant without disclosing any information apart from that it is fulfilled or not. Thus, the blueprint of the provider's infrastructure is not disclosed to the tenant.
- **Trust:** The service can prove properties to tenants and tenants can verify the proofs. This way tenants are assured that policy predicates such as geo-location separation are fulfilled. As a result the tenants' trust to the cloud provider increases.

The IA service has a number of disadvantages according to the SWOT analysis. Threats and weaknesses have been analyzed with Microsoft Threat Modelling Tool, whose mitigation is treated in section 4.6.7. The disadvantages for the service are the following:

- **Performance degradation:** Resource consumption attacks can be hard to deal with, because the way an attacker request web resources is like that of any legitimate client, and the only differentiating attribute is their intention.
- **Elevation of privileges:** Cross-site request forgery (CSRF or XSRF) is a type of attack in which an attacker forces a user's browser to make a forged request to a vulnerable site by exploiting an existing trust relationship between the browser and the vulnerable web site. In a simple scenario, a user is logged in to web site A using a cookie as a credential. The other browses to web site B. Web site B returns a page with a hidden form that posts to web site A. Since the browser will carry the user's cookie to web site A, web site B now can take any action on web site A, for example, adding an admin to an account. The attack can be used to exploit any requests that the browser automatically authenticates, e.g. by session cookie, integrated authentication, IP whitelisting, ... The attack can be carried out in many ways such as by luring the victim to a site under control of the attacker, getting the user to click a link in a phishing email, or hacking a reputable web site that the victim will visit. The issue can only be resolved on the server side by requiring that all authenticated state-changing requests include an additional piece of secret payload (canary or CSRF token) which is known only to the legitimate web site and the browser and which is protected in transit through SSL/TLS.
- **Repudiation:** A repudiation attack happens when an application or system does not adopt controls to properly track and log users' actions, thus permitting malicious manipulation or forging the identification of new actions. This attack can be used to

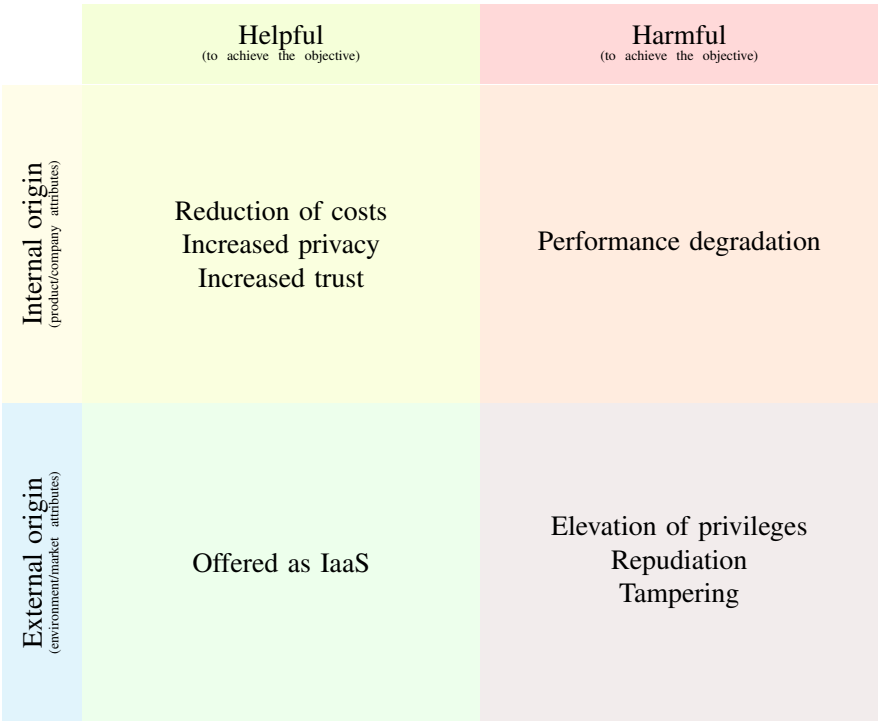


Figure 35: SWOT analysis of Infrastructure Auditing Service.

change the authoring information of actions executed by a malicious user in order to log wrong data to log files. Its usage can be extended to general data manipulation in the name of others, in a similar manner as spoofing mail messages. If this attack takes place, the data stored on log files can be considered invalid or misleading. Regarding the IA service this attack can be realized when the cloud provider is not the one that has sent the graph topology to the auditor for certification. This means that the auditor will not sign the correct graph topology.

4.6.3 Usage Model and Stakeholders

The IA Service has three main stakeholders:

- **Auditor:** This stakeholder defines the graph encoding that is going to be used during certification of the cloud infrastructure.
- **Cloud Provider:** The Provider requests from the auditor to sign the topology and to provide him with a topology certificate.
- **Tenant:** This user requests from the provider to prove that a policy predicate is fulfilled. Then the tenant verifies the result of the proof process.

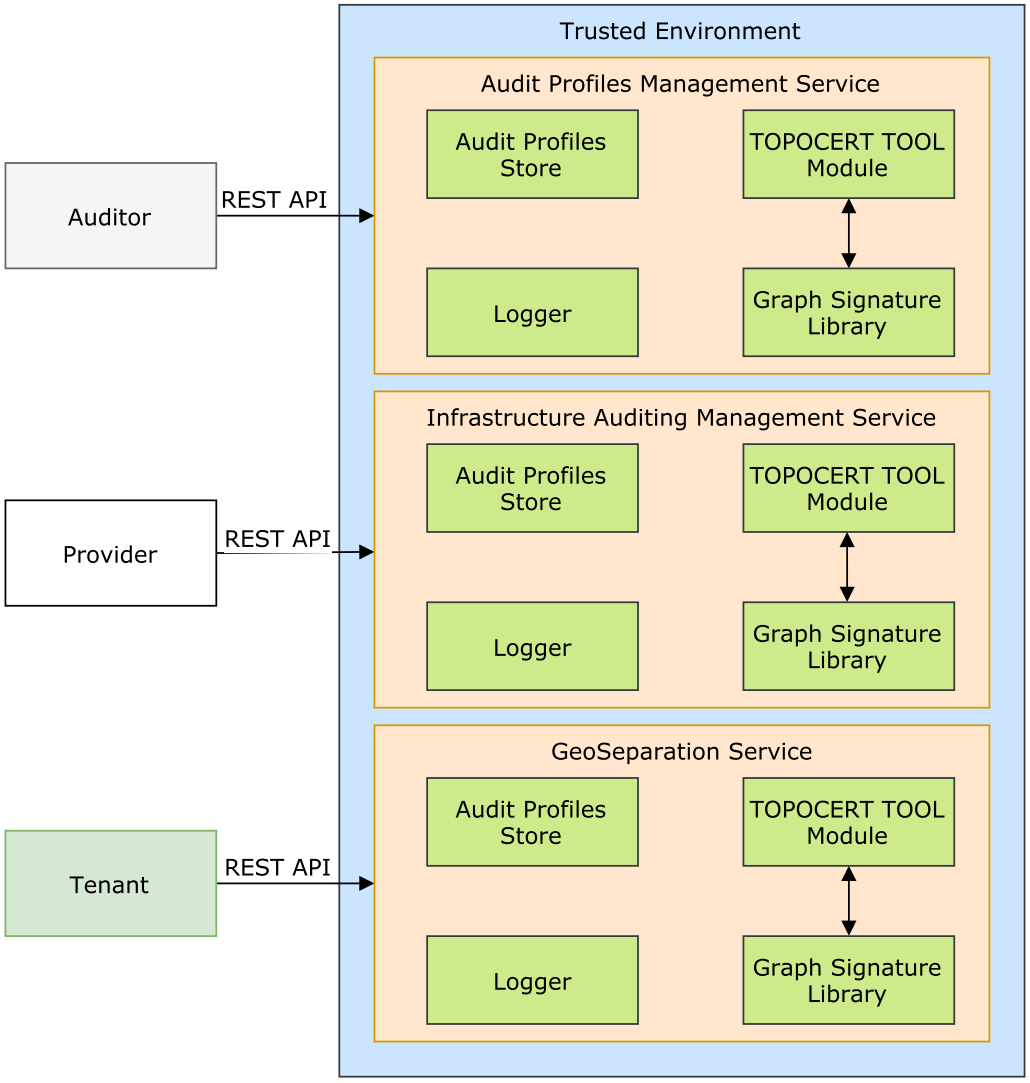


Figure 36: Interaction dynamics for infrastructure auditing service

4.6.4 Service Model and Interaction Dynamics

The high level component shown in Figure 36 shows main component and the main services that are interacting in the system

- *Audit-Profiles Management Service*: The goal of this service is to provide a REST interface to auditors that can specify and certify audit profiles. Upon obtaining a graph representation G using the REST API the service will issue a partial graph signature. The service comprises of 4 components:
  - The *audit profile store* provides a REST API to store audit profiles for the auditors.
  - The *logger* is an interface that the service can use for logging information regarding the correct execution of operations.
  - The *TOPOCERT module* is responsible for the setup of the certification environment for the topology certification, which includes the certification of the topology language. Computes a partial graph signature which is handed over to the provider.
  - The *Graph Signature library* provides the signer low-level role that is responsible for the low-level key setup, for certifying and encoding scheme and to sign graphs.
- *Infrastructure Auditing Management Service*: providers use this service to search and select audit profiles according to their requirements. The service also provides a REST API for providers that want to sign their infrastructure and provide a graph representation to auditors. It comprises of the same 4 components as the previous service. We only consider their differences for the discussion of this service.
  - The *audit profile store* offers a REST API to initiate the issuing protocol with the auditor and select an appropriate audit-profile offered by the auditor or import an audit-profile.
  - The *logger* provides an interface that the service can use for logging information regarding the correct execution of operations for the providers.
  - The *TOPOCERT Tool module* provides the functionality for the high level recipient role which offers a topology signature.
  - The *Graph Signature library* provides the recipient low-level role that is responsible for creating graph commitments and completing the partial signature sent by the signer with his randomness R.
- The *GeoSeparation Service* offers a REST API for providers to list policy predicates and tenants to issue queries about the topology that the provider offers. This services uses the same components as the previous services. However, each of the component uses its appropriate facet for this service. We discuss the differences that exist with the 4 components:
  - The *audit profiles store* uses a REST API for storing audit profiles for the provider to retrieve its information used for the zero knowledge proof of knowledge.
  - The *logger* provides an interface that is used to log the verification result.
  - The *TOPOCERT module* enters the prover role and draws on the graph signature on the topology and the graph representation to create a zero-knowledge

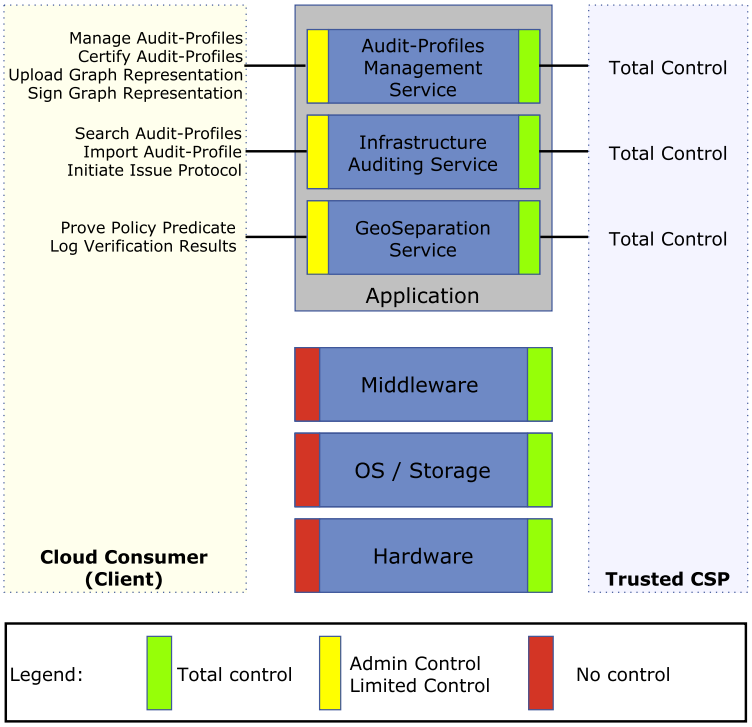


Figure 37: Scope of control for Infrastructure Auditing service deployment

proof of knowledge on policy predicate $\mathfrak{P}$ using the low-level graph signature library.

– The *Graph Signature library* computes zero-knowledge proofs of knowledge with a policy predicate $\mathfrak{P}$ on graph signatures.

4.6.5 Provider/Consumer Scope of Control

The deployment of the Infrastructure Auditing service consists of deploying three different services. The Cloud Provider is responsible for deployment and maintenance of the services. The Cloud Consumer is able to configure the services according to its requirements and make use of the available REST API that each service provides. The scope of control in the software stack is displayed in Figure 37 and discussed in the following:

- *Cloud Consumer*: the client is only allowed to perform the operations required to make the service work following the particular requirements for each client's role. For instance a client with an Auditor role can only access the API of the Audit-Profiles Management service. The same method is used for the Provider and Tenant roles.
- *Cloud Provider*: the Cloud Provider is responsible for hosting and operating the Infrastructure Auditing service. The service needs to be deployed in cloud infrastructure that has public access from the Internet.

4.6.6 Parameters

There are no special parameters available.

4.6.7 Application Development

This section discusses the result of the STRIDE analysis performed with the Microsoft Threat Modelling Tool. The aim of this analysis is to identify the threats that the infrastructure auditing service can be exposed to and the security controls that mitigate or minimize them.

Attack Surface Analysis In the particular case of the Infrastructure Auditing Service, there are three services that enable the topological infrastructure certification and verification. The Audit-Profiles Management service provides auditors the ability to manage and certify audit profiles alongside signing graph topologies. The auditor user manages the services using a UI where only an authorized user can make adjustments to the service. Both the provider and tenant users use the same approach. The users manage their respective services using a UI where only authenticated users can make changes to the configuration of the service.

There are three interaction points for an auditor, provider, or tenant user that constitute the network attack surface:

1. The Audit-Profiles Management service is a set of webservices and a UI that allows an auditor to setup the required details for issuing and certifying audit profiles and signing graph representations.
2. The Infrastructure Audit Management service is a set of webservices and a UI that allows a provider to search for audit profiles, select the one that is required and initiate the issue protocol to sign a graph representation.
3. The GeoSeparation service is a set of webservices and a UI that allows a tenant to ask the provider to prove a particular policy predicate $\mathfrak{P}$. The service also offers the tenant a way to log the verification result.

Gathering Threats with STRIDE

We utilized the Microsoft Threat Modelling Tool to arrive at Figure 38. The generated graph was the base for automatically generated threats—a first preliminary examination of those resulted in the following table:

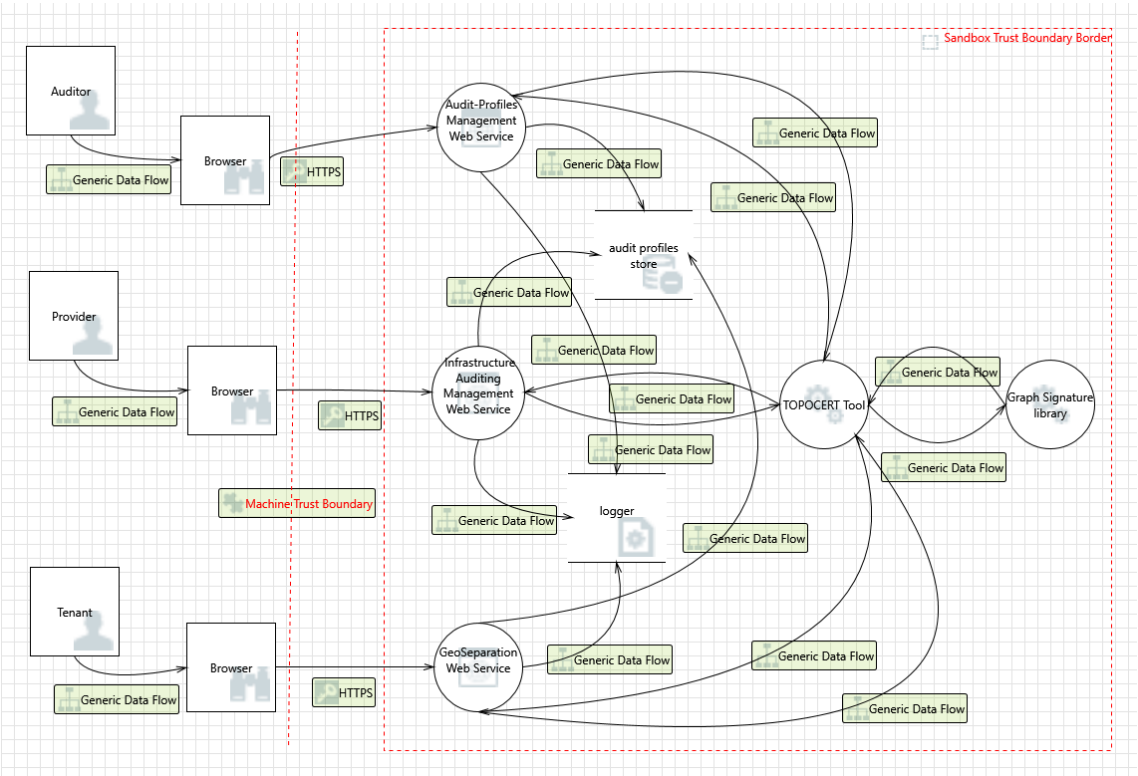


Figure 38: Attack surface model for the infrastructure auditing service.

STRIDE Category	Original	Not-Applicable	Mitigated/Design	Resulting
Spoofing	6	3	0	3
Tampering	13	0	3	10
Repudiation	3	0	0	3
Information Disclosure	9	0	0	9
Denial of Service	12	3	0	9
Elevation of Privilege	17	3	3	11
Total	60	9	6	45

Analyzing and Mitigating remaining Threats

There are two main types of threats that require further investigation:

- *Performance degradation*: the infrastructure auditing service requires a way to mitigate excessive resource consumption of the service. One possible solution for performance degradation is to deploy an API gateway or a Web Application Firewall (WAF) to limit API usage both for the absolute number of APIs calls and the rate of API calls. This approach reduces massive API requests that cause denial of services, mitigates brute-force attacks and mitigates misuses of the services. Monitoring API usage and sending alerts when the service is overloaded with requests or the API call patterns seem suspicious.
- *Repudiation*: an auditing mechanism should be included in the infrastructure auditing service to detect that service requests originate from a legitimate entity. This mechanism provides an audit trail during the service operation.

4.7 Encryption Proxy (EPaaS)

4.7.1 Overview

The main objective of the Encryption Proxy service is allowing organizations to move existing server applications to the cloud, in a way that all sensitive data reaches the cloud in a encrypted way, and with no need of modifications on the original software.

In order to achieve this objective, the Encryption Proxy service offers a configurable reverse proxy which is capable of analyzing the traffic transversing it (most common internet protocols have been considered) and encrypting or decrypting the sensitive fields as necessary.

The Encryption Proxy service is provided in the form of a cloud service, allowing different organizations to register themselves and configure the details of their applications APIs, information which is used to automatically create an endpoint for protecting the corresponding application by encrypting and decrypting its traffic on-the-fly.

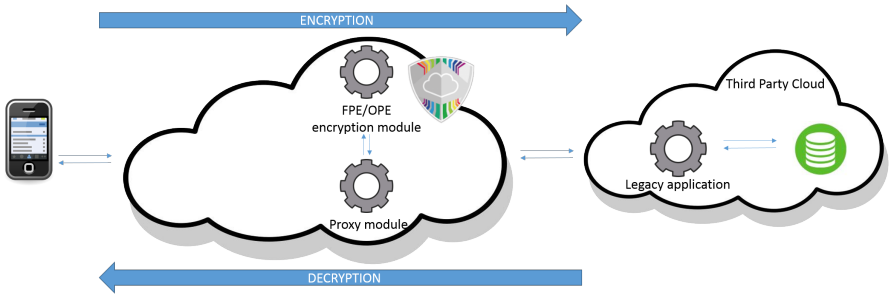


Figure 39: Encryption Proxy (EP) overview.

4.7.2 Key Features

In Figure 40 we show a SWOT analysis of the given service.

As it can be seen in the SWOT analysis, the major advantages of the service are:

- **Reduction of costs.** The service allows organizations to take benefit of moving their existing applications to the cloud (costs outsourcing, increased availability, cloud elasticity to adapt to organization requirements...) without investing in re-designs
- **Increased privacy.** Organizations willing to move their applications to public clouds can do it in a way that the sensitive data remains hidden to the cloud provider or other any third party
- **Offered as SaaS.** Organizations can register themselves and configure the service accordingly to their requirements, no deployment of extra software is needed by their side

However, some disadvantages appears for the service. Threats and weaks have been analyzed with Microsoft Threat Modelling Tool, whose mitigation is treated in section 4.7.7. These disadvantages are:

- **Performance degradation:** Resource consumption attacks can be hard to deal with, because the way an attacker request web resources is like that of any legitimate client, and the only differentiating attribute is their intention.
- **Elevation of privileges:** Cross-site request forgery (CSRF or XSRF) is a type of attack in which an attacker forces a user's browser to make a forged request to a vulnerable site by exploiting an existing trust relationship between the browser and the vulnerable web site. In a simple scenario, a user is logged in to web site A using a cookie as a credential. The other browses to web site B. Web site B returns a page with a hidden form that posts to web site A. Since the browser will carry the user's cookie to web site A, web site B now can take any action on web site A, for example, adding an admin to an account. The attack can be used to exploit any requests that the browser automatically authenticates, e.g. by session cookie, integrated authentication, IP whitelisting, ... The attack can be carried out in many ways such as by luring the victim to a site under control of the attacker,

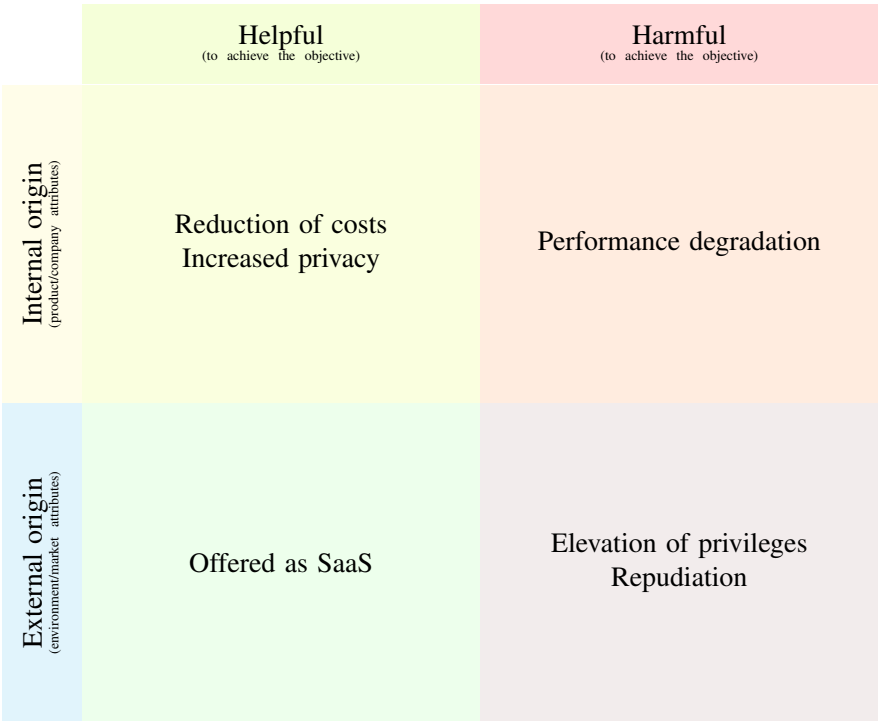


Figure 40: SWOT analysis of Encryption Proxy Service.

getting the user to click a link in a phishing email, or hacking a reputable web site that the victim will visit. The issue can only be resolved on the server side by requiring that all authenticated state-changing requests include an additional piece of secret payload (canary or CSRF token) which is known only to the legitimate web site and the browser and which is protected in transit through SSL/TLS. See the Forgery Protection property on the flow stencil for a list of mitigations

- **Repudiation:** A repudiation attack happens when an application or system does not adopt controls to properly track and log users' actions, thus permitting malicious manipulation or forging the identification of new actions. This attack can be used to change the authoring information of actions executed by a malicious user in order to log wrong data to log files. Its usage can be extended to general data manipulation in the name of others, in a similar manner as spoofing mail messages. If this attack takes place, the data stored on log files can be considered invalid or misleading.

4.7.3 Usage Model and Stakeholders

As remarked in Figure 39, execution of the service logic is performed in the central point that needs to be trusted, since encryption/decryption of sensitive fields takes place there. Encryption Proxy service needs therefore to be offered by a trusted provider entity. Three different classes of stakeholders are involved:

- The *Cloud Customer*, or client, is the organization that moves the application to be protected to the cloud. He is able to benefit from all the features provided by the cloud, while protecting sensitive data without managing the complex details that it entails.
- The *Secure Proxy Provider* runs the Secure Proxy service and offers it to Cloud Customers. It manages the execution of the Secure Proxy software, which in fact means handling the encryption keys of the different cloud customers. It is therefore necessary that a trust relation exists between the Cloud Customer and the Secure Proxy Provider.
- The *Cloud Provider* manages the cloud infrastructure where the applications of the Cloud Customers are deployed. The Secure Proxy service allows Cloud Customers to use this infrastructure without a existing trust relation with the Cloud Provider, since sensitive data reaches this infrastructure in a encrypted way.

4.7.4 Service Model and Interaction Dynamics

The high level component diagram shown in Figure 41 shows the main concept plus required software components which are interacting in the system:

- *Client:* This is the client-side of the customer application.
- *Server:* This is the server-side of the customer application, which has been moved to external infrastructure offered by a cloud provider.
- *Encryption proxy service:* This is the main component of the system, which offers different endpoints (to be used by different applications) accordingly to the registered organizations - Cloud Customers - and the details provided about their applications. The service comprises two main components: the Proxy and Protocol Analyzer and the Encryption/Decryption service.
  - The *Proxy and Protocol Analyzer* receives the messages being exchanged between client and server applications, analyzing them, extracting the sensitive fields and changing those with their corresponding encrypted or decrypted version (depending on the direction of the communication).
  - The *Encryption/Decryption service* is the component which actually performs the Format and Order Preserving Encryption of the sensitive fields identified by the Proxy and Protocol Analyzer.

4.7.5 Provider/Consumer Scope of Control

The deployment of the Encryption Proxy service is as usual for SaaS software, where the Secure Proxy Provider is responsible for deployment and maintenance of the service, while the Cloud Customer is able to set-up the service accordingly to its needs, and make use of it. The scope of control in the software stack is shown in Figure 42 and discussed in the following.

- *Cloud Consumer:* cloud consumer is only allowed to perform those set-up operations needed to make the service work with the particularities of the application to be

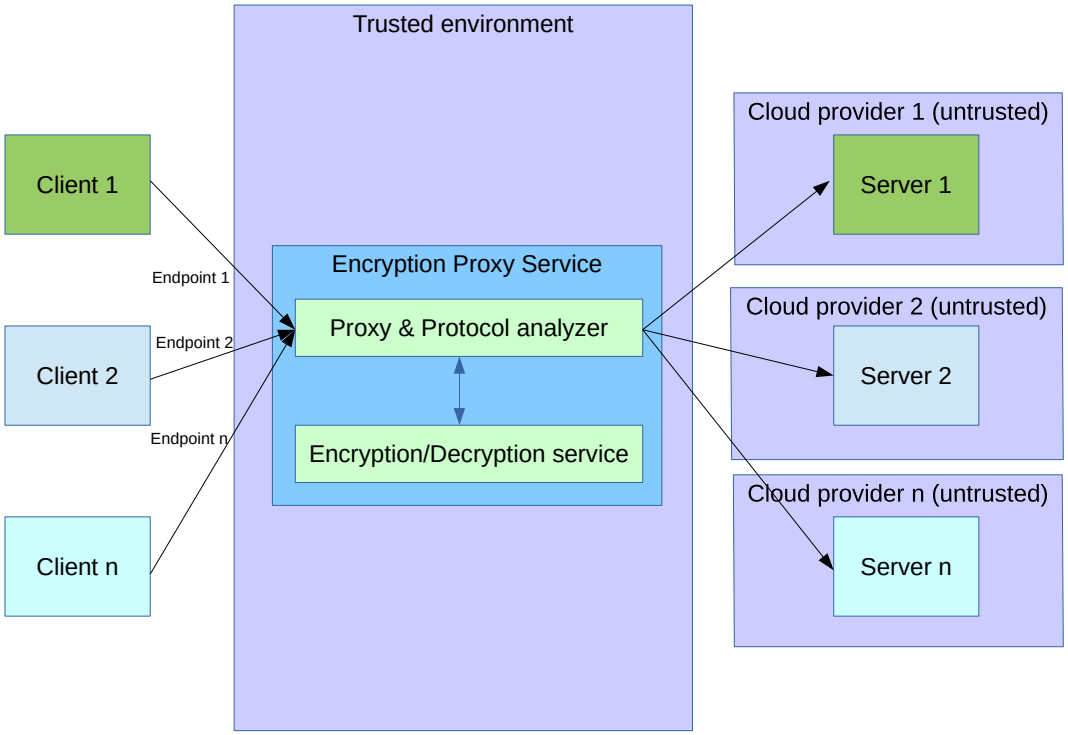


Figure 41: Interaction dynamics for encryption proxy service.

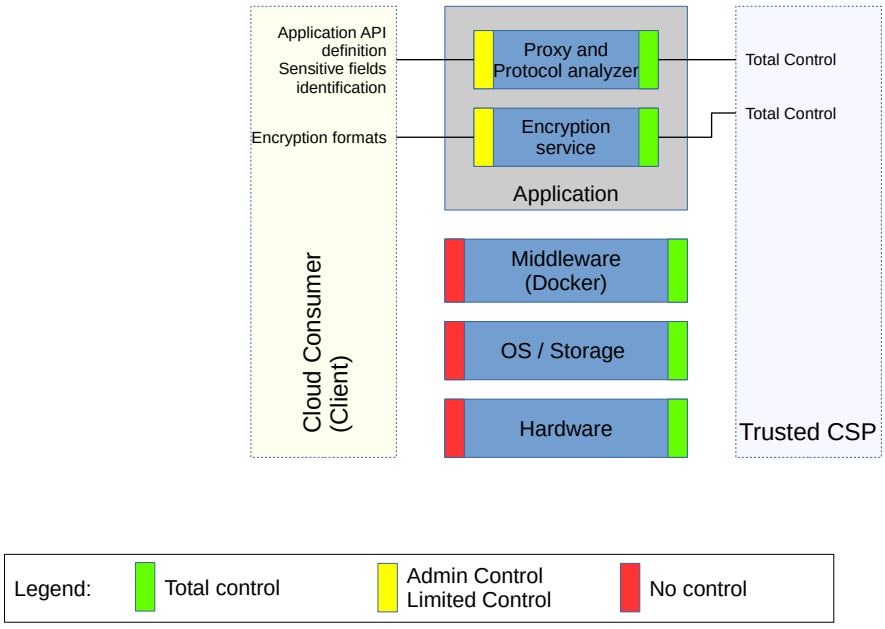


Figure 42: Scope of control encryption proxy service deployment.

protected. This mainly has to do with the specification of the API calls and sensitive fields to be protected, and the definition of the formats of those fields that will be used to perform the encryption and decryption.

- *Encryption Proxy Provider:* The Encryption Proxy provider is responsible for hosting and operating the PRISMACLOUD Encryption Proxy solution. The service needs to be deployed in a compatible infrastructure with public access from the internet.

4.7.6 Parameters

- *Proxy and protocol analyzer*
  - *Target endpoint:* the actual endpoint of the server side of the protected application
  - *Application API details:* in order to perform its work, the protocol analyzer needs to know which particular messages of the protocol contain sensitive fields, and the identification of those.
- *Encryption and decryption service*
  - *Sensitive fields format:* the advantage of the Format and Order Preserving Encryption performed by the Encryption Proxy service is that it allows the protected application to remain unchanged, since the produced encrypted data type of each sensitive field is fully compatible with the one expected. This formats are defined by the Cloud Consumer, and are particular for its application.

4.7.7 Application Development

Attack Surface Analysis

The Encryption Proxy service mainly 2 different interaction points to the Cloud Customers:

- The *Encryption Proxy frontend* is a set of webservices and a UI allowing customer registration and set-up of the required details for the proxy to properly protect an application.
- The *Proxy and Protocol Analyzer* provides an endpoint that proxies the traffic between the client-side and server-side parts of the protected application.

All other interactions of the service are between internal components of it, and therefore kept under the control of the trusted environment. Figure 43 summarizes our potential attack surface.

Gathering threats with STRIDE

We utilized the Microsoft Threat Modelling Tool to arrive at Figure 43. The generated graph was the base for automatically generated threats—a first preliminary examination of those resulted in the following table:

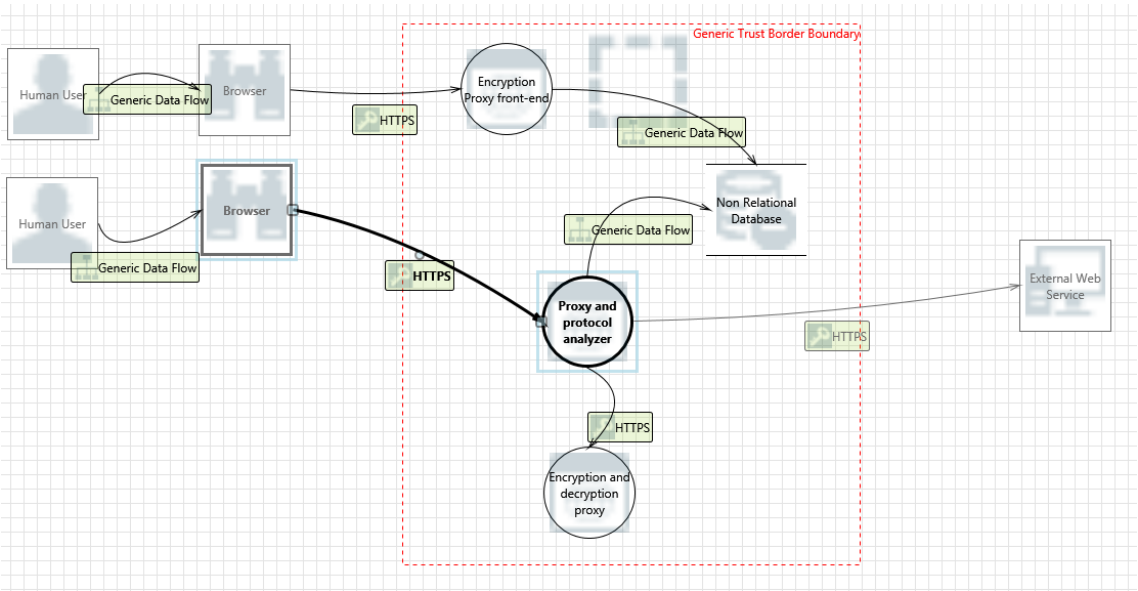


Figure 43: Attack surface model for the encryption proxy service.

STRIDE Category	Original	Not-Applicable	Mitigated/Design	Resulting
Spoofing	4	3	1	0
Tampering	2	1	1	0
Repudiation	2	0	0	2
Information Disclosure	2	0	2	0
Denial of Service	6	2	0	4
Elevation of Privilege	9	2	2	5
Total	25	8	6	11

The complete list of threats can be found in the appendix in Section B.3.

Analyzing and Mitigating remaining Threats

Summarizing, main threats that need further investigation deal with:

- *Performance degradation:* control mechanisms for monitoring and dealing with excessive resource consumption of the service need to be put in place.
- *Elevation of privileges:* the implementation of the service needs to be tested to confirm the non-existence of usual web-based applications threats.
- *Repudiation:* proper auditing mechanisms are advised to be included in the service, in order to detect unfair customers.

4.8 Big Data Anonymization (BDAaaS)

4.8.1 Overview

The Big Data Anonymization service (BDAaaS) is motivated by our eHealth use case and is designed to support the anonymization of large data sets – those data sets that do not fit in main memory. The service allows for the upload and download of huge files (Gigabytes), and supports the anonymization process by employing Spark – a general engine for large-scale data processing – to distribute and parallelize the anonymization algorithms.

The service and has been designed to seamlessly support both smaller sized data sets as well as larger data sets. In the rest of this section we will show that the service supports advanced anonymization algorithms (such as Hilbert, RTree and more), is extended to support t-closeness and provides a simple and consistent API for both small and large data sets.

4.8.2 Key Features

In this section, we highlight the main advantages of the BDA service, which are the following:

- Increased Privacy: support for releasing data sets while preserving personal privacy: The service supports several advanced k-anonymity algorithms able to produce anonymized data sets which preserve personal privacy as well as data utility. These include: Mondrian, Cluster based, Hilbert and R+-Tree based algorithms. The service provides support for t-closeness for all algorithms.
- Support for big data: The service supports large data sets. Specifically, the two best performing algorithms (Hilbert and R+-Tree) have been modified to support scalable anonymization of large data sets.
- Flexible Configuration: The service enables the user to configure the anonymization algorithm, the parameters k and t as well as the data privacy types (PII, QI, SA). In turn, the user can apply a risk based approach and release data which contains more utility (information) with a higher degree of risk for personal privacy or, alternatively, a data set with a low risk for personal privacy and less utility.

In Figure 44 we show a SWOT analysis of the given service.

4.8.3 Usage Model and Stakeholders

The BDA Service has two main stakeholder:

- The customer is the one who has access to the BDA Service. He can perform anonymization without knowledge of the anonymization algorithms, or worries whether the algorithm scales to handle big data. The customer benefits from the anonymization service without the complexities of deploying and managing the service or the

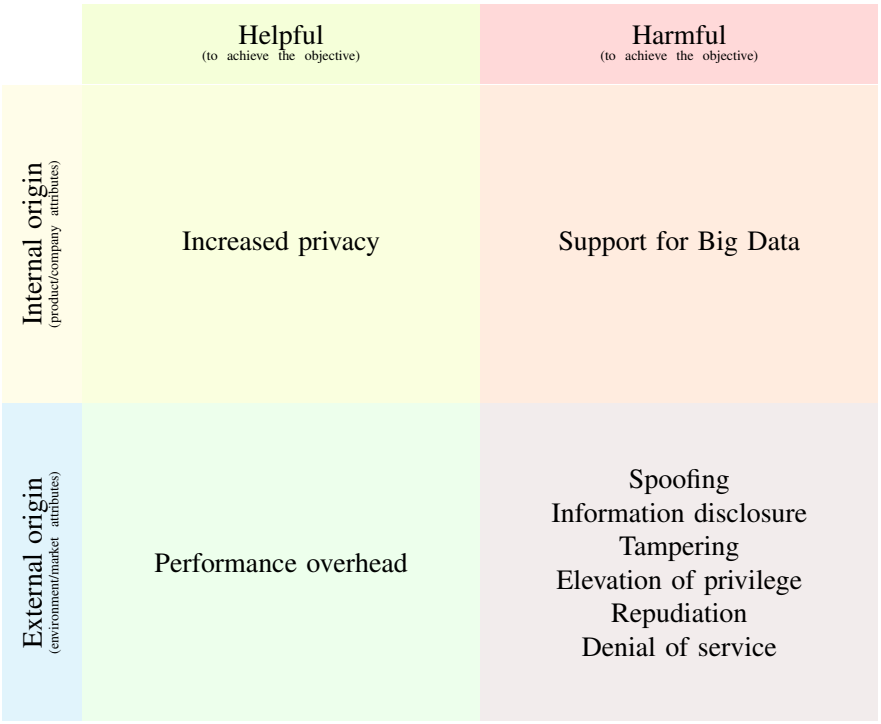


Figure 44: SWOT analysis of Secure Archiving Service (BDAaaS).

Spark cluster. As the service provider has access to the data the customer must trust the service provider.

- The BDAaaS provider is the one anonymizing the data. He is the one maintaining and deploying the anonymization algorithms and is responsible for setting up the spark cluster. He has access to the data and must secure access to it. However, the provider may remove all data once the processing is complete, removing the need for (external) persistence storage and minimize the security risk.

4.8.4 Service Model and Interaction Dynamics

In this section, we explain the interaction with the service and review the service major components.

The **cloud customer**, using a web browser or a local application, is the one initiating all the interactions. To anonymize a data set the following steps must be taken:

1. Upload file
2. Upload configuration
3. Anonymize
4. Download anonymized file

Each step comprises of a single REST API call.

The **BDAaaS provider** is the one running the Anonymization service. The service has the following components (see Figure 1):

1. Anonymization core controller – a component responsible for handling all REST API calls and implements the business logic.
2. DATAPRIV library – this library implements the underlying anonymization algorithms. It can be called from the controller or from the BDA Spark component.
3. BDA Spark – a component that provides support for big data anonymization. It is the one managing the Spark cluster and utilizes the DATAPRIV library for its core anonymization algorithms.
4. Spark cluster – a general purpose spark cluster which can handle large scale data sets.

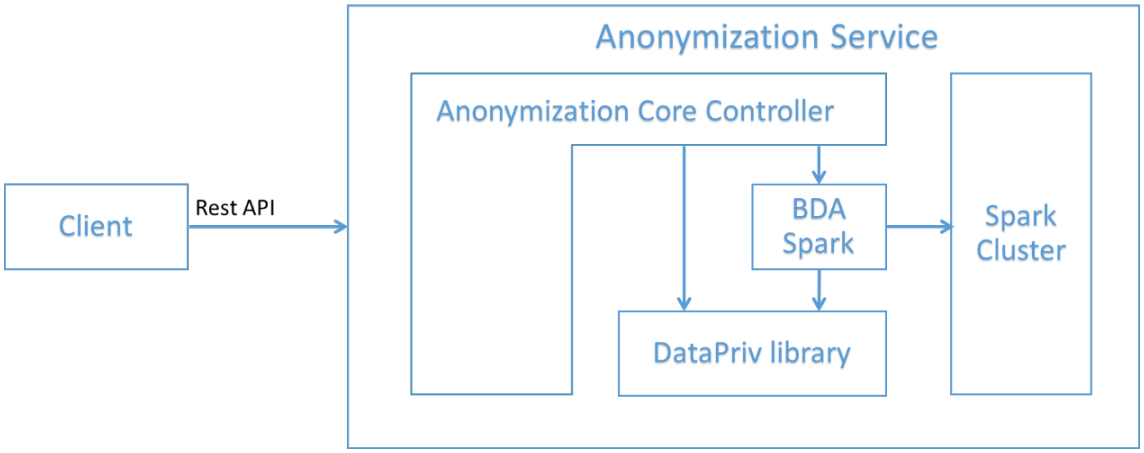


Figure 45: The architecture of the service

4.8.5 Provider/Consumer Scope of Control

The BDAaaS is deployed in a very typical manner. It involves two parties, a consumer and provider.

- Consumer – The consumer scope of control is minimal. The consumer is able to initiate the interaction and control the service operation. The exposed controls allow the user to customize the anonymization operation (the algorithms and parameters) and specify the degree of parallelism (number of cluster nodes participating).
- BDAaaS provider – The BDAaaS provider is responsible for operating and managing the cloud and the service. In addition, the provider is responsible for running and managing the spark cluster.

4.8.6 Paramters

There are no special parameters available.

4.8.7 Application Development

Attack Surface Analysis

During our analysis, we consider the following attack surfaces:

Network attack surface:

The service builds upon the Spring MVC framework to implement the REST API. Therefore, the network attack surface of Spring MVC framework is included in the service network attack surface. Additionally, the BDA service uses the Spark cluster to handle big data. Therefore, the network attack surface of Spark is also included in the service's network attack surface. Both the Spark cluster and Spring MVC are trusted. We assume the Spark cluster itself is deployed within the Trust boundary. Therefore, the network attack surface is minimal .

Software Attack surface:

The service provides the following operations: Post MacroConfiguration Retrieve MacroConfiguration Delete MacroConfiguration Upload file Download file Delete file Perform anonymization We note that for input validation we use the standard Spring MVC mechanisms. In particular, converting the MacroConfiguration json to an object is done by Spring MVC serialization mechanism and erroneous formats will be caught. We note that using authentication mechanism will reduce the software attack surface and prevent the attackers from exploiting the service's REST API. Additionally, further decreasing the attack surface can be done by deploying the service inside the enterprise and prevent unauthenticated users from accessing the service.

Physical Attack surface:

Securing the servers – by the cloud provider – on which the service is deployed will prevent physical access of the attackers. Reducing the attack surface even more can be done by deploying the servers inside the enterprise (or private cloud) and allow access only to authorized users.

We used Microsoft Threat Modelling tool to create the attack surface model for the anonymization service (see Figure 2)

Gathering Threats with STRIDE

STRIDE Category	Original	Not-Applicable	Mitigated/Design	Resulting
Spoofing	6	3	2	1
Tampering	8	3	1	4
Repudiation	1	-	-	1
Information Disclosure	3	1	-	2
Denial of Service	3	-	1	2
Elevation of Privilege	6	1	3	2
Total	27	8	7	12

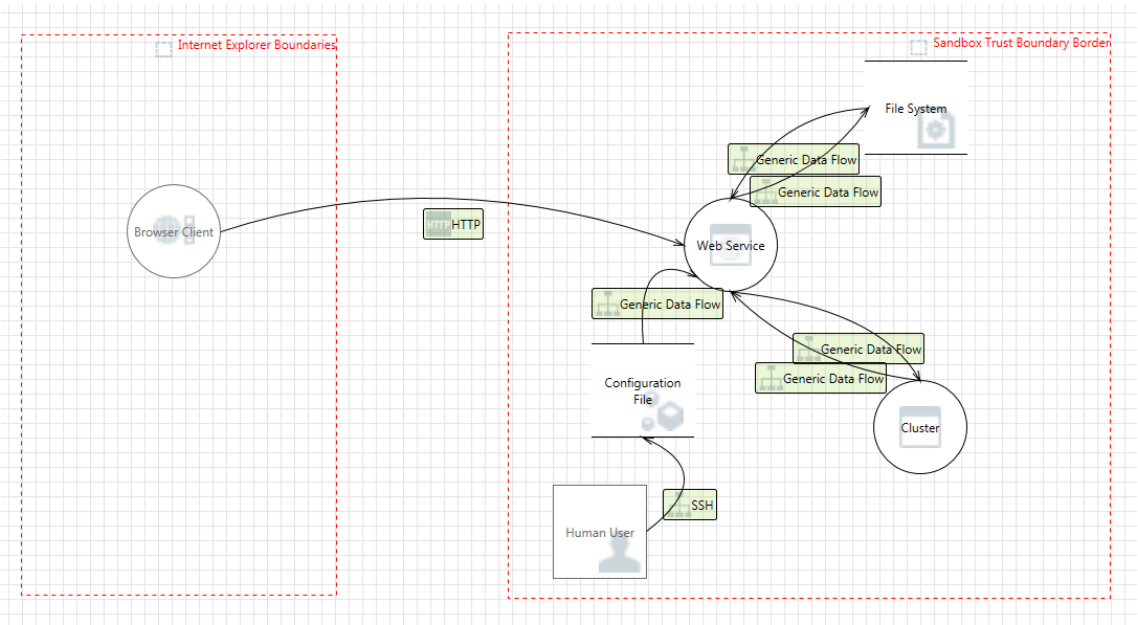


Figure 46: Attack surface model for the anonymization service.

In the following sections we address the threats, and propose solutions for the unsolved threats.

File System Threats:

The web service uses the local file system as storage for the uploaded file and resulting anonymized data. The file system is accessible only the OS of the machine, which is a trusted party.

ID	Type	Description	Status/Solution
1	Spoofing	File System may be spoofed by an attacker and this may lead to data being written to the attacker's target instead of File System.	use a standard authentication mechanism to identify the destination data store
2	Denial of Service	Does Web Service or File System take explicit steps to control resource consumption? Resource consumption attacks can be hard to deal with, and there are times that it makes sense to let the OS do the job. Be careful that your resource requests don't deadlock, and that they do timeout.	Mitigated by Design

3	Spoofing	File System may be spoofed by an attacker and this may lead to incorrect data delivered to Web Service. Consider using a standard authentication mechanism to identify the source data store.	N/A
4	Information Disclosure	Improper data protection of File System can allow an attacker to read information not intended for disclosure.	Review authorization settings and remove files once the operation has completed

Cluster Related Threats:

Our design assumes the cluster is deployed locally or within the trust boundary. When the BDAaaS provider deploy a cluster in a trusted environment we assume that the provider is the only authorized person who can access the cluster or manage it.

ID	Type	Description	Status/Solution
5	Tampering	If Web Service is given access to memory, such as shared memory or pointers, or is given the ability to control what Cluster executes (for example, passing back a function pointer.), then Web Service can tamper with Cluster. Consider if the function could work with less access to memory, such as passing data rather than pointers. Copy in data provided, and then validate it.	N/A. The service does not control access or control the Cluster.
6	Tampering	Packets or messages without sequence numbers or timestamps can be captured and replayed in a wide variety of ways. Implement or utilize an existing communication protocol that supports anti-replay techniques (investigate sequence numbers before timers) and strong integrity.	use authentication and integrity mechanism between the web service and the Cluster
7	Tampering	Attackers who can send a series of packets or messages may be able to overlap data. For example, packet 1 may be 100 bytes starting at offset 0. Packet 2 may be 100 bytes starting at offset 25. Packet 2 will overwrite 75 bytes of packet 1.	Use authentication, authorization and integrity mechanisms

8	Elevation Of Privilege	Cluster may be able to impersonate the context of Web Service in order to gain additional privilege.	Mitigated by Design
9	Tampering	If Cluster is given access to memory, such as shared memory or pointers, or is given the ability to control what Web Service executes (for example, passing back a function pointer.), then Cluster can tamper with Web Service. Consider if the function could work with less access to memory, such as passing data rather than pointers. Copy in data provided, and then validate it.	Mitigated by Design
10	Tampering	Packets or messages without sequence numbers or timestamps can be captured and replayed in a wide variety of ways. Implement or utilize an existing communication protocol that supports anti-replay techniques (investigate sequence numbers before timers) and strong integrity.	use authentication and integrity mechanism between the web service and the Cluster
11	Tampering	Attackers who can send a series of packets or messages may be able to overlap data. For example, packet 1 may be 100 bytes starting at offset 0. Packet 2 may be 100 bytes starting at offset 25. Packet 2 will overwrite 75 bytes of packet 1. Ensure you reassemble data before filtering it, and ensure you explicitly handle these sorts of cases.	Use authentication, authorization and integrity mechanisms
12	Elevation Of Privilege	Web Service may be able to impersonate the context of Cluster in order to gain additional privilege.	N/A. The web Service is trusted.

Configuration file related threats:

The configuration file contains very little data – the port number which web service should use. In addition, the configuration file is located on the same machine as the web service, and, the only party who can access and manipulate the file is the service provider.

ID	Type	Description	Status/Solution
13	Information Disclosure	Improper data protection of Configuration File can allow an attacker to read information not intended for disclosure. Review authorization settings.	N/A No sensitive data is stored in the file

14	Spoofing	Configuration File may be spoofed by an attacker and this may lead to incorrect data delivered to Web Service.	Mitigated by Design
15	Spoofing	Configuration File may be spoofed by an attacker and this may lead to data being written to the attacker's target instead of Configuration File.	Mitigated by Design

Client browser related threats:

Currently the client is using the http protocol to query the web service (see Figure), which leads to several threats. However an industrial strength deployment would address those threats by configuring the web service to use https as well as a trusted authentication and authorization mechanism.

Id	Type	Description	Status/Solution
16	Elevation Of Privilege	Web Service may be able to remotely execute code for Browser Client.	Scan and validate the service code for bugs. Use a standard authentication mechanism to identify the source process.
17	Spoofing	Browser Client may be spoofed by an attacker and this may lead to unauthorized access to Web Service.	Use a standard authentication mechanism to identify the source process.
18	Spoofing	Web Service may be spoofed by an attacker and this may lead to information disclosure by Browser Client.	N/A (HTTPS)
19	Tampering	Data flowing across HTTP may be tampered with by an attacker. This may lead to a denial of service attack against Web Service or an elevation of privilege attack against Web Service or an information disclosure by Web Service. Failure to verify that input is as expected is a root cause of a very large number of exploitable issues.	N/A (HTTPS)

20	Tampering	If Browser Client is given access to memory, such as shared memory or pointers, or is given the ability to control what Web Service executes (for example, passing back a function pointer.), then Browser Client can tamper with Web Service.	Mitigated by Design
21	Repudiation	Web Service claims that it did not receive data from a source outside the trust boundary.	use logging or auditing to record the source, time, and summary of the received data. .
22	Information Disclosure	Data flowing across HTTP may be sniffed by an attacker. Depending on what type of data an attacker can read, it may be used to attack other parts of the system or simply be a disclosure of information leading to compliance violations.	N/A (HTTPS)
23	Denial Of Service	Web Service crashes, halts, stops or runs slowly; in all cases violating an availability metric.	Test service before deployment.
24	Denial Of Service	An external agent interrupts data flowing across a trust boundary in either direction.	Use an authorization mechanism for the trusted boundary.
25	Elevation Of Privilege	Web Service may be able to impersonate the context of Browser Client in order to gain additional privilege.	Mitigated by Design (input check)
26	Elevation Of Privilege	Browser Client may be able to remotely execute code for Web Service.	Mitigated by Design (input check)
27	Elevation Of Privilege	An attacker may pass data into Web Service in order to change the flow of program execution within Web Service to the attacker's choosing.	Mitigated by Design (input check

4.8.8 Operational Aspects

As explained above, when the service is anonymizing a large data set, the BDA Spark component is the one responsible for interacting with the Spark cluster. During the development of the service, we assumed that the cluster service is deployed locally, or alternatively inside the trust boundary. Note, however, that if the service provider utilizes a cluster outside the trust boundary additional security measures must be taken.

Lastly, to support deployments where a Spark cluster is not available, the service supports a local deployment of Spark. This deployment allows the user to configure the service to run parallel algorithms on smaller sized data sets although there is no cluster available.

5 Summary and Conclusions

In this report we presented the results of the effort undertaken in the project to guide the development of security enhanced services. The challenges addressed in this work were manifold and the following goals have been achieved:

- Bring together cryptographers and industry people to develop novel security enhanced applications
- Exploit the potential of the multidisciplinary consortium as good as possible
- To standardize and streamline the development and documentation in the project
- Develop a reference manual for PRISMACLOUD services

To reach the first goal, we had to improve the communication in the project and identify the right languages for different levels and phases of developments in the project. The most important step towards this goal was the development of a **reference architecture**. The architecture served the purpose to identify and separate the different expert domains we found in the project and define interfaces in order to coordinate documentation and coordination among the project partners, i.e., to implement a very efficient and lean infrastructure to coordinate the interdisciplinary work. The development of a reference architecture worked out to be extremely helpful in the project and did work well. It was used throughout all work packages to identify interaction points and to group and locate work items in the large effort PRISMACLOUD is. The architecture helped to develop the concept of the PRISMACLOUD toolkit and the PRISMACLOUD services as key exploitable results of the project, and made clear how they can be exploited independently of the applications developed to further improve the impact of the project. The architecture was also ideally suited to communicate the structure of the project to external people, no matter if they are technical experts or from a different field.

Additionally, together with the architecture we defined a **development methodology** which leveraged the structure of the architecture and its very natural interfaces. It is based on the idea of component reuse and the well known V-model in software development. The requirements are defined, extracted and translated in a top down approach and the development and testing is done bottom up. However, the major idea of the methodology is to support reuse and encapsulate knowledge of lower layers in well known black box components. Systems components can be reused without deeper understanding of inner workings in a straight forward way, i.e., to prevent potential misunderstanding or wrong configuration.

The basic methodology was than mapped into **guidelines and best practices** to support secure by design service composition and development. The most important components of the proposed guidelines have also been defined as project standards for documentation of the PRISMACLOUD services. For us, it was important to prepare documentation for the most important steps of software and service development, such that they can later be reused when real project development starts.

As a result of this activities the document also contains a **reference documentation of the Prismacloud services**. All eight PRISMACLOUD services have been analysed

according the project standard and follow the overall architectural design guidelines. In particular they are based on the developed tools specified in WP5 and the services can be considered as a very specific and easy way to deliver the functionality of the tools to application developers.

All in all, work described in this report was extremely fruitful for the overall project and fostered many interesting discussions. It helped to organize the work and communication as well as enabled a common reference documentation of PRISMACLOUD tools and services. We now have a well documented set of generic domain independent PRISMACLOUD services which can be used in many contexts to build novel applications. The work was conducted in very close cooperation with other WP7 tasks but also all other work packages, especially in the development of the architecture all project partners were involved.

References

[a4c14] D:c-2.1 report detailing conceptual framework, version: Final of 13.10.2014. A4Cloud deliverable, 2014.

[AFHP+15] Alaa Alaqra, Simone Fischer-Hübner, John Sören Pettersson, Frank van Geelkerken, Erik Wästlund, Melanie Volkamer, Thomas Länger, and Henrich C. Pöhls. PRISMACLOUD public deliverable D2.1: Legal, Social and HCI Requirements, 2015.

[AIS77] Christopher Alexander, Sara Ishikawa, and Murray Silverstein. *A Pattern Language: Towns, Buildings, Construction.* Oxford University Press, 1977.

[BDD+15] Johannes Buchmann, Denise Demirel, David Derler, Lucas Schabhüser, and Daniel Slamanig. PRISMACLOUD D5.8: Overview of Verifiable Computing Techniques Providing Private and Public Verifiability. Technical report, H2020 Prismacloud, www.prismacloud.eu, 2015.

[BFF+09] Christina Brzuska, Marc Fischlin, Tobias Freudenreich, Anja Lehmann, Marcus Page, Jakob Schelbert, Dominique Schröder, and Florian Volk. Security of sanitizable signatures revisited. In *Proceedings of the 12th International Conference on Practice and Theory in Public Key Cryptography (PKC)*, volume 5443 of *LNCS*, pages 317–336, Berlin, Heidelberg, 2009. Springer-Verlag.

[BPS13] Christina Brzuska, Henrich C. Pöhls, and Kai Samelin. Non-interactive public accountability for sanitizable signatures. In Sabrina De Capitani di Vimercati and Chris Mitchell, editors, *Public Key Infrastructures, Services and Applications: 9th European Workshop, EuroPKI 2012, Pisa, Italy, September 13-14, 2012, Revised Selected Papers*, pages 178–193, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[Can01] Ran Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001*, pages 136–145. IEEE Computer Society, 2001.

[CCK+06] Ran Canetti, Ling Cheung, Dilsun Kirli Kaynar, Moses Liskov, Nancy A. Lynch, Olivier Pereira, and Roberto Segala. Time-bounded task-pioas: A framework for analyzing security protocols. In Shlomi Dolev, editor, *Distributed Computing, 20th International Symposium, DISC 2006*, volume 4167 of *Lecture Notes in Computer Science*, pages 238–253. Springer, 2006.

[CDPW07] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In Salil P. Vadhan, editor, *Theory of Cryptography, 4th Theory of Cryptography Conference, TCC 2007*, volume 4392 of *Lecture Notes in Computer Science*, pages 61–85. Springer, 2007.

[CEK+16] Jan Camenisch, Robert R. Enderlein, Stephan Krenn, Ralf Küsters, and Daniel Rausch. Universal composition with responsive environments. *IACR Cryptology ePrint Archive*, 2016:34, 2016.

[CKL+15] Jan Camenisch, Stephan Krenn, Anja Lehmann, Gert Læssøe Mikkelsen, Gregory Neven, and Michael Østergaard Pedersen. Formal Treatment of Privacy-Enhancing Credential Systems. In Orr Dunkelman and Liam Keliher, editors, *Selected Areas in Cryptography - SAC 2015*, volume 9566 of *Lecture Notes in Computer Science*, pages 3–24. Springer, 2015.

[DDH+16] Denise Demirel, David Derler, Christian Hanser, Henrich C. Pöhls, Daniel Slamanig, and Giulia Traverso. PRISMACLOUD public deliverable D4.4: Overview of Functional and Malleable Signature Schemes, 2016.

[DG13] Nick Doty and Mohit Gupta. Privacy Design Patterns and Anti-Patterns. In *Workshop "A Turn for the Worse: Trustbusters for User Interfaces Workshop" at SOUPS 2013 Newcastle, UK*, 2013.

[DKLT16] Denise Demirel, Stephan Krenn, Thomas Lorünser, and Giulia Traverso. Efficient and Privacy Preserving Third Party Auditing for a Distributed Storage System. In *International Conference on Availability, Reliability and Security – ARES 2016*. IEEE, 2016. (to appear).

[DKS16] David Derler, Stephan Krenn, and Daniel Slamanig. Signer-Anonymous Designated-Verifier Redactable Signatures for Cloud-Based Data Sharing, 2016. (currently under submission).

[ERS02] Eastlake, Reagle, and Solo. XML-signature syntax and processing. W3C recommendation. www.w3.org/TR/xmldsig-core/, Feb. 2002.

[Eur16] European Commission. Regulation (EU) 2016/679 of The European Parliament and of The Council, of 27 April 2016, on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation), 2016. (online 20.7.2017).

[FHKP+11] S. Fischer-Hübnner, C. Köffel, J.-S. Pettersson, P. Wolkerstorfer, C. Graf, and L. Holtz. HCI Pattern Collection–Version 2, 2011. (PrimeLife project).

[fSC05] International Organization for Standardization and International Electrotechnical Commission. Information technology – Security techniques – Information security management systems – Requirements). Standard, July 2005.

[GHJV94] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley. ISBN 0-201-63361-2, 1994.

[Gro14] Thomas Groß. Efficient certification and zero-knowledge proofs of knowledge on infrastructure topology graphs. In *ACM Workshop on Cloud Computing Security (CCSW 2014)*, pages 69–80. ACM, 2014.

[HTL+14] A. Hudic, M. Tauber, T. Lorünser, M. Krotsiani, G. Spanoudakis, A. Mauthe, and E. R. Weippl. A multi-layer and multitenant cloud assurance evaluation

methodology. In *Cloud Computing Technology and Science (CloudCom), 2014 IEEE 6th International Conference on*, pages 386–393, Dec 2014.

[Kra01] Hugo Krawczyk. The order of encryption and authentication for protecting communications (or: How secure is ssl?). In *Annual International Cryptology Conference*, pages 310–331. Springer, 2001.

[KT13] Ralf Küsters and Max Tuengerthal. The IITM model: a simple and expressive model for universal composability. *IACR Cryptology ePrint Archive*, 2013:25, 2013.

[Küs] Ralf Küsters. Simulation-based security with inexhaustible interactive turing machines. In *Computer Security Foundations Workshop, (CSFW-19 2006)*.

[LHS15] Thomas Loruenser, Andreas Happe, and Daniel Slamanig. ARCHISTAR: Towards Secure and Robust Cloud Based Data Sharing. In *Cloud Computing Technology and Science (CloudCom), 2015 IEEE 7th International Conference on*, pages 371–378, nov 2015.

[MA05] M. McIntosh and P. Austel. XML signature element wrapping attacks and countermeasures. In *Proceedings of Workshop on Secure Web Services*, 2005.

[Mau11] Ueli Maurer. Constructive cryptography - A new paradigm for security definitions and proofs. In Sebastian Mödersheim and Catuscia Palamidessi, editors, *Theory of Security and Applications - Joint Workshop, TOSCA 2011*, volume 6993 of *Lecture Notes in Computer Science*, pages 33–56. Springer, 2011.

[MR11] Ueli Maurer and Renato Renner. Abstract cryptography. In Bernard Chazelle, editor, *Innovations in Computer Science - ICS 2010*, pages 1–21. Tsinghua University Press, 2011.

[NA12] Nist and Emmanuel Aroms. *NIST Special Publication 800-53 Revision 3 Recommended Security Controls for Federal Information Systems and Organizations*. CreateSpace, Paramount, CA, 2012.

[Pah15] Claus Pahl. Containerization and the paas cloud. *IEEE Cloud Computing*, 2(3):24–31, 2015.

[PH11] Henrich C. Pöhls and Focke Höhne. The Role of Data Integrity in EU Digital Signature Legislation - Achieving Statutory Trust for Sanitizable Signature Schemes. In *International Workshop on Security and Trust Management (STM)*. Springer LNCS, 2011.

[PW00] Birgit Pfitzmann and Michael Waidner. Composition and integrity preservation of secure reactive systems. In Dimitris Gritzalis, Sushil Jajodia, and Pierangela Samarati, editors, *CCS 2000, Proceedings of the 7th ACM Conference on Computer and Communications Security*, pages 245–254. ACM, 2000.

[SFBH+06] Markus Schumacher, Eduardo Fernandez-Buglioni, Duane Hybertson, Frank Buschmann, and Peter Sommerlad. *Security Patterns - Integrating Security and Systems Engineering*. John Wiley & Sons, Ltd. West Sussex, England, 2006.

[SG07] J. Schwenk S. Gajek, L. Liao. Towards a Formal Semantic of XML Signature. In *W3C Workshop Next Steps for XML Signature and XML Encryption*, 2007.

[SH12] Daniel Slamanig and Christian Hanser. On cloud storage and the cloud of clouds approach. In *ICITST 2012*, pages 649–655, 2012.

[vGPFH15] F.W.J. van Geelkerken, Henrich C. Pöhls, and Simone Fischer-Hübner. The legal status of malleable- and functional signatures in light of Regulation (EU) No 910/2014. In *Proc. of 3rd International Academic Conference of Young Scientists on Law & Psychology 2015 (LPS 2015)*. L'viv Polytechnic Publishing House, Nov. 2015.

A Toolkit Overview

In this section we are going to quickly review the tools developed in PRISMACLOUD and their most important implications when it comes to cloudification. The final specifications of the tools will be developed in WP5, however, a preliminary high-level view on them will show its structure and how they are related to the service design according to the PRISMACLOUD methodology.

As already mentioned, a tool can be thought of as an abstract concept, or piece of software, which is composed of various primitives and can be parametrized in various different ways. Subsequently, we briefly discuss all the tools.

A.1 Secure Object Storage Tool (SECSTOR)

This tool is dedicated to build cloud storage systems with strong security guarantees, i.e., confidentiality and availability. It builds upon the idea of a federated or multi-cloud application [SH12] and the different components of the tool are shown in Figure 47. A *dealer* component generates data fragments and sends them to storage nodes called *server*. The servers can be considered storage nodes which are also comprising active components with the ability to execute protocol logic in addition to the basic read and write operations supported by passive cloud storage interfaces. The *reader* component is reconstructing the data for read operations and the *verifier* is able to audit the storage system, i.e., it can remotely check the state of the data stored in the system.

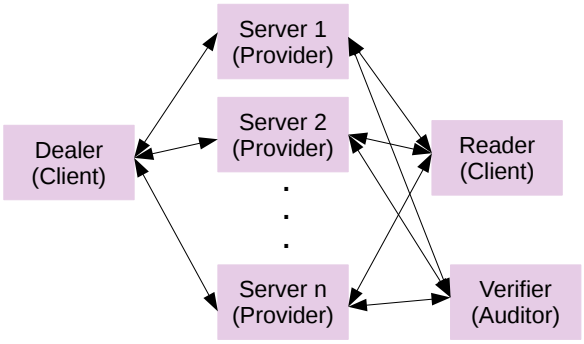


Figure 47: Secure Object Storage Tool (SECSTOR).

To achieve the desired security properties we are leveraging the concept of secure information dispersal – data is fragmented and distributed over different servers to build a secure and reliable virtual service on top of multiple less reliable services. Due to the use of secret sharing the individual fragments contain no information about the plaintext data. Our goal is to design a storage layer which is more than the sum of its parts. No single cloud provider shall have access to plaintext data or be able to tamper with them by modifying local fragments nor are all fragments necessary to recover the plaintext In summary, the overall availability achieved by the Secure Object Storage Tool is better than the one of

individual nodes and the tool also prevents from most of provider related threats in cloud usage.

In PRISMACLOUD we apply secure variants of data fragmentation called secret sharing. With secret sharing, data can be encoded into multiple fragments such that no single fragment contains any information about the original content and a predefined threshold of k fragments is required to reconstruct the data. If these fragments (shares) are distributed to different storage servers, the data stays secure w.r.t. confidentiality protection as long as less than k servers are colluding to reconstruct the information and the data stays secure w.r.t. availability as long as more than k honest servers are reachable. Additionally, it must be noted that the confidentiality and availability guarantees of the PRISMACLOUD *Secure Object Storage Tool* are also keyless, which further facilitates sharing of data between different users in the system.

From a functional point-of-view, the tool provides a way for outsourcing data storage to potentially less trustworthy and reliable storage providers with both increased data availability and confidentiality [LHS15]. Furthermore, we are targeting robustness against strong adversary models like active attackers or adversaries with unlimited computational power. Additionally the tool is supporting concurrent multi-user access by potentially malicious clients and provides other interesting features discussed below.

Dealer Component. The main function of the dealer component (Dealer) is to encode the data and generate the different fragments used in the distributed storage network. The dealer works on unstructured data, and at its core, it implements secret sharing protocols. However, in order to support different application scenarios, adversary models and network models, the component must support various encoding schemes and protocols. The common functionality behind the different protocols is that of threshold secret sharing. The choice of threshold sharing used ranges from plain information theoretical solutions (PSS) to more efficient but only computational secure variants (CSS) and also verifiable versions is the dealer component is running on untrusted platforms. On the connectivity level, the dealer has an authentic and private comunication channel to each server in the system.

Server Component. The server component (Server) represents the storage backend used to compose the cloud-of-clouds storage layer. Multiple servers are required and each of them communicates with the Dealer over a secure channel (authentic and private). The different servers are holding the data and represent storage nodes in different trust zones of the system. The trust zones are used to model the non-collusion assumption. In practice, storage options range from fully-federated dedicated cloud providers to storage nodes under different administrative domains within a cloud provider's data center. The server components are communicating with the dealer over secure channels and for some of the features provided by the tool, the servers also have secure channels between them.

Reader Component. The reader component (Reader) is responsible for reconstruction of data stored in the system. Basically it encapsulates the reconstruction procedure of the used secret sharing method plus the interaction protocol to get sufficient shares to recover the plaintext information desired. We assume secure channels between a reader and all involved servers in the system.

Table 7: Features of the SECSTOR tool.

ID	Feature	C/I/A	Method
PC.Tool.1.F.01	Protect data from provider access	C	SSS (CSS)
PC.Tool.1.F.02	Provide long-term secuirty	C	SSS (SS)
PC.Tool.1.F.03	Long-term security with proactivity	C	SSS (PSS)
PC.Tool.1.F.04	Self-healing properties to recover from failure	A	SSS (PSS)
PC.Tool.1.F.05	Robustness against malicious clients	I	SSS (VSS)
PC.Tool.1.F.06	Mean to verify the retrievabiity of data	I/A	RDC
PC.Tool.1.F.07	Enable third party auditing	I/A	RDC
PC.Tool.1.F.08	Support self audititing	I/A	RDC
PC.Tool.1.F.09	Efficient means to verify large data sets	I	RDC (BatchRDC)
PC.Tool.1.F.10	Robustness against passive attacks after sharing	C	
PC.Tool.1.F.11	Robustness against active attacks after sharing	C/I	BFT
PC.Tool.1.F.12	Robustness against passive attacks	C	
PC.Tool.1.F.13	Robustness against failstop attacks	C/I/A	PAX
PC.Tool.1.F.14	Robustness against active attacks	C/I/A	BFT
PC.Tool.1.F.15	Support multiple concurrent clients	I	
PC.Tool.1.F.16	Support multiple concurrent users	I	IAM
PC.Tool.1.F.17	Support secure deletion	C	BFT
PC.Tool.1.F.18	Secure and robust access logs	C/I/A	BFT
PC.Tool.1.F.19	Access privacy for plaintext data	P	PIR
PC.Tool.1.F.20	Access privacy for confidential data	P	PIR
PC.Tool.1.F.21	Access privacy with leakage detection	P	PIR (SPIR)

Verifier Component. This component is responsible for the verification of data stored in the system. Together with the servers it conducts a protocol to obtain a proof about the retrievability of stored data, i.e., it remotely checks if the servers are still storing consistent fragments. The verifier is not considered to be trusted, therefore, the auditing protocols executed must be privacy preserving, i.e., it must not leak any information about the data stored. Based upon this, the verifier must have an authentic channel to each of the servers, but those channels don't need to be confidentiality protected. The Verifier is intended to model a third-party auditing service which can check the data consistency remotely with strong cryptographic properties but without learning anything about the data.

The features supported by the SECSTOR tool are shown in Table 7. It provides various different features which can be built into the system. However, some of them have a major impact in system design and some of them are conflicting with others. A full reference will be given in WP5 with all protocol explanation and configuration guidance.

A.2 Flexible Authentication with Selective Disclosure Tool (FLEXAUTH)

This tool supports the authentication of arbitrary messages (or documents) by means of digital signatures with selective disclosure features. This tool has three different compo-

nents (cf. Figure 48), being an *authentication component*, a *selective disclosure component*, and a *verification component*.

Authentication Component. The originator generates a signed message that contains well defined rules (policy) what parts of the message can be selectively disclosed.

Selective Disclosure Component. Given a signed message from the authentication component, it provides the functionality to selectively disclose parts of the information of the original signed message (or document) to other receiving parties. When this selective disclosure happens according to the defined policy, the authenticity can still be verified.

Verification Component. A verifying party can then use this part of the tool to verify the authenticity of any partial information that was created from authentic information just by means of the originator's verification key. Note, the *verification component* checks if only authorised, i.e., conforming to policy, selections were done.

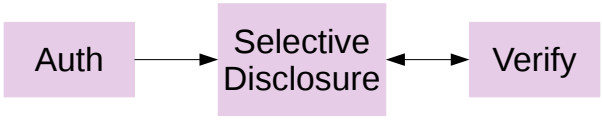


Figure 48: Flexible Authentication with Selective Disclosure Tool

For the realisation of this component several cryptographic primitives can be used. Especially if the authentication component shall be decoupled from the selective disclosure component, so that the latter can operate bound by the policy but without the need of an interaction, then a group of special digital signature schemes referred to as malleable signatures can be facilitated.

A.3 Verifiable Data Processing Tool (VERIDAT)

This tool supports the delegation of processing authenticated data in a way that the result can be efficiently verified for correctness. It comprises three different components depicted in Figure 49. The data (and potentially some additional metadata) originates at the *client component*. The *data processing component* is given a set of input data and a description of the processing rules, and outputs the result of the computation, as well as a proof certifying the correctness of the delegated computation. The *verification component* takes a result and a proof (and potentially additional information) and can efficiently verify the correctness of the computation.

On a high level, the Verifiable Data Processing Tool allows to perform verifiable computations on data such as computing statistics on medical data. Depending on the type of verifiability, thereby, the data processed can be either only verified by the data owner or any third party. Although there are very powerful technical tools for very broad classes of functions, they are not yet practically efficient. In deliverable D5.8 [BDD+15] we provide a comprehensive overview of the current state of the art in verifiable computing and discuss

shortcomings of existing solutions.

Figure 49: Verifiable Data Processing Tool.

In PRISMACLOUD we aim at developing a tool that provides efficient solutions for restricted classes of functions. Besides efficiency, we aim for schemes that are secure against so called strong (adaptive) adversaries, i.e., adversaries who can adaptively ask computation queries and also learn about whether the forgery attempts verify. This is required, because some of the processed data has a high protection level. For the same reason we also want to support schemes that offer input and/or output privacy. In a first step we only address computationally bounded adversaries and aim at extending the tool by variations covering computationally unbounded adversaries at a later stage. With respect to the underlying primitives, we focus on approaches where primitives with reasonable performance are available. For each variation we will take both types of verifiability, i.e., private and public verifiability, into account. The latter one is preferable, because it allows to perform third party audits. However, for some applications it might be sufficient that only the data owner, i.e., client component, is able to perform the verification and for this scenario we might get a more efficient solution when providing only private verifiability. The tool will heavily rely on malleable signature primitives. In particular, to authenticate the input data which then support to evaluate arithmetic circuits. In addition the tool may also require secret sharing schemes, functional signature schemes and zero-knowledge proofs.

The components of the Verifiable Data Processing Tool are as follows:

Client Component. The *client component* produces a set of signed data on which it wants to have a function, represented as an (arithmetic) circuit $\mathcal{C}$, evaluated.

Computation Component. The *computation component* receives a circuit $\mathcal{C}$ and signed personal data from one or more client components and produces a signed output together with a proof of correct computation.

Verification Component. The *verification component* takes the result and proofs computed by the computation component and verifies the correctness of the evaluation of a circuit $\mathcal{C}$.

The tool developed will contain several variations that differ with respect to the functionalities supported. More precisely, we will provide different classes of operations and several levels of security, privacy, and verifiability. This allows to use the tool for many different applications.

A.4 Topology Certification Tool (TOPOCERT)

The Topology Certification Tool supports the application of graph signatures to certify and prove properties of topologies represented as graphs. The tool is realized as an interactive protocol framework between the roles of an *issuer*, a *prover* and a *verifier*. It consists of three abstract components, depicted in Figure 50, that encapsulate the different roles. The topology is provided by another entity in a standard graph format.

Figure 50: Topology Certification Tool.

This specification relates to the cryptographic protocol framework proposed by Groß [Gro14].

Issuer Component. Given a graph representation of the topology in a standard format, the issuer is responsible for the certification of the encoding for the topology certification framework, as well as for issuing a topology certificate to the prover. The issuer outputs a graph signature on that graph.

Prover Component. The prover compiles a zero-knowledge proof on the topology certificate that can convince a verifier of the requested security properties of the graph in a zero-knowledge proof of knowledge.

Verifier Component. The verifier validates the proof against the public key of the issuer and is convinced of the requested security properties.

On a high level, the Topology Certification Tool supports the creation of digital signatures on topology graphs in such a way that the graphs are accessible to zero-knowledge proofs of knowledge. This will support the infrastructure auditing service of PRISMACLOUD which is tasked to show isolation of different resources, for example, whether the resources of a Tenant A are segregated from the resources of a Tenant B. Consequently, the infrastructure auditing service requires additonal functionalities beyond the foundational operations of the Issuer, Prover and Verifier components from the Topology Certification Tool. First, the Issuer needs to be capable of issuing topology certificates in the size of the infrastructure in question. Second, the Prover and the Verifier need to be able to compute isolation proofs on topology certificates. Such proofs have been specified by Groß [Gro14]. These proofs require that the proof of possession on the topology certificate needs to be further decomposed into commitments on the edges of the graph. Furthermore, the prover needs to prove equality of the committed values with the topology certificate, compute a cumulative product in commitments and finally show that these products are co-prime.

To facilitate these proofs, the graph considered should not be edge-labeled. For the interface of the infrastructure we only consider undirected edge-unlabeled graphs. The

decomposition of the topology certificate in a number of commitments takes computational time proportional to the number of edges of the entire topology graph. To allow for live demonstrations, the topology size should be selected such that the proof can be computed in reasonable time.

A.5 Data Privacy Tool (DATAPRIV)

This tool provides the means for processing data in different ways before they are moved to untrusted environments. It includes components providing the capabilities to encrypt data while preserving the format or ordering of the data. This tool enables users of legacy applications to move their databases to a public cloud, while preserving data privacy and confidentiality. Moreover, the tool provides components for data generalization as means for anonymizing bulk data using k-anonymity techniques.

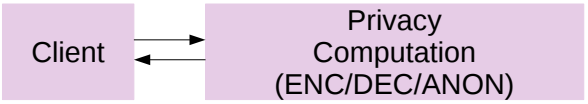


Figure 51: Data Privacy Tool.

On a high level, this tool will provide the means to process data in different ways, supporting different purposes and different privacy requirements. It will include different components providing the following capabilities: (1) data encryption while preserving format or order, and (2) data generalization to guarantee k-anonymity. The tool's input may range from a single data item (e.g., to be encrypted) to a complete data set (e.g., to be anonymized). Based on the input data and the preferred privacy method, the processing will produce the required output.

B Threat Analysis Results

B.1 Threats for Secure Archiving

B.1.1 Tampering

ID	Name
9	Risks from Logging
17	Risks from Logging
25	XML DTD and XSLT Processing
38	The Cloud Storage Data Store Could Be Corrupted
40	Risks from Logging
50	The Configuration File Data Store Could Be Corrupted

B.1.2 Denial Of Service

ID	Name
7	Potential Excessive Resource Consumption for Web Service or Cache
15	Potential Excessive Resource Consumption for Web Service or File System
20	Potential Process Crash or Stop for Web Service
21	Data Flow HTTPS Is Potentially Interrupted
22	Data Store Inaccessible
28	Potential Process Crash or Stop for Web Service
29	Data Flow HTTPS Is Potentially Interrupted
34	Potential Excessive Resource Consumption for Web Service or Cloud Storage
35	Data Store Inaccessible
36	Data Flow HTTPS Is Potentially Interrupted
48	Data Flow HTTPS Is Potentially Interrupted
52	Data Flow Generic Data Flow Is Potentially Interrupted
53	Data Store Inaccessible

B.1.3 Spoofing

ID	Name
1	Spoofing of Source Data Store Configuration File
4	Spoofing of Source Data Store Cache
5	Spoofing of Destination Data Store Cache
8	Spoofing of Destination Data Store File System
16	Spoofing of Source Data Store Cloud Storage
32	Spoofing the External Web Application External Entity
33	Spoofing of Destination Data Store Cloud Storage
46	Spoofing of the External Web Application External Destination Entity
49	Spoofing of Destination Data Store Configuration File

B.1.4 Information Disclosure

ID	Name
2	Weak Access Control for a Resource
3	Weak Access Control for a Resource
6	Authorization Bypass
14	Authorization Bypass
19	Weak Access Control for a Resource
39	Authorization Bypass
45	Weak Credential Storage

B.1.5 Repudiation

ID	Name
10	Lower Trusted Subject Updates Logs
11	Data Logs from an Unknown Source
12	Insufficient Auditing
13	Potential Weak Protections for Audit Data
18	Potential Data Repudiation by Web Service
27	Potential Data Repudiation by Web Service
37	Data Store Denies Cloud Storage Potentially Writing Data
41	Lower Trusted Subject Updates Logs
42	Data Logs from an Unknown Source
43	Insufficient Auditing
44	Potential Weak Protections for Audit Data
47	External Entity External Web Application Potentially Denies Receiving Data
51	Data Store Denies Configuration File Potentially Writing Data

B.1.6 Elevation Of Privilege

ID	Name
23	Web Service May be Subject to Elevation of Privilege Using Remote Code Execution
24	Elevation by Changing the Execution Flow in Web Service
26	Elevation Using Impersonation
30	Web Service May be Subject to Elevation of Privilege Using Remote Code Execution
31	Elevation by Changing the Execution Flow in Web Service

B.2 Threats for Secure Sharing

B.2.1 Tampering

ID	Name
23	Potential Lack of Input Validation for Browser Client
27	Web Service Process Memory Tampered
40	Browser Client Process Memory Tampered
48	Web Service Process Memory Tampered
51	Browser Client Process Memory Tampered

B.2.2 Denial Of Service

ID	Name
2	Potential Excessive Resource Consumption for Web Service or Cloud Storage
6	Potential Excessive Resource Consumption for Web Service or Cloud Storage
9	Potential Excessive Resource Consumption for Browser Client or HTML5 Local Storage
13	Data Flow Generic Data Flow Is Potentially Interrupted
19	Data Flow Generic Data Flow Is Potentially Interrupted
20	Potential Process Crash or Stop for Browser Client
29	Potential Process Crash or Stop for Browser Client
30	Data Flow Generic Data Flow Is Potentially Interrupted
37	Data Flow Generic Data Flow Is Potentially Interrupted
38	Potential Process Crash or Stop for Web Service
45	Data Flow Generic Data Flow Is Potentially Interrupted
46	Potential Process Crash or Stop for Browser Client
53	Potential Process Crash or Stop for Web Service
54	Data Flow Generic Data Flow Is Potentially Interrupted

B.2.3 Spoofing

ID	Name
1	Spoofing of Destination Data Store Cloud Storage
3	Spoofing of Source Data Store Cloud Storage
5	Spoofing of Destination Data Store Cloud Storage
7	Spoofing of Source Data Store Cloud Storage
10	Spoofing of Destination Data Store HTML5 Local Storage
12	Spoofing of Source Data Store HTML5 Local Storage
15	Spoofing of the Human User External Destination Entity
24	Spoofing the Human User External Entity
25	Spoofing the Browser Client Process
28	Spoofing the Web Service Process
41	Spoofing the Browser Client Process
49	Spoofing the Web Service Process
50	Spoofing the Browser Client Process

B.2.4 Information Disclosure

ID	Name
4	Weak Access Control for a Resource
8	Weak Access Control for a Resource
11	Weak Access Control for a Resource
21	Data Flow Sniffing

B.2.5 Repudiation

ID	Name
14	External Entity Human User Potentially Denies Receiving Data
22	Potential Data Repudiation by Browser Client
26	Potential Data Repudiation by Browser Client
39	Potential Data Repudiation by Web Service
47	Potential Data Repudiation by Browser Client
52	Potential Data Repudiation by Web Service

B.2.6 Elevation Of Privilege

ID	Name
16	Elevation by Changing the Execution Flow in Browser Client
17	Browser Client May be Subject to Elevation of Privilege Using Remote Code Execution
18	Elevation Using Impersonation
31	Elevation Using Impersonation
32	Browser Client May be Subject to Elevation of Privilege Using Remote Code Execution
33	Elevation by Changing the Execution Flow in Browser Client
34	Elevation by Changing the Execution Flow in Web Service
35	Web Service May be Subject to Elevation of Privilege Using Remote Code Execution
36	Elevation Using Impersonation
42	Elevation by Changing the Execution Flow in Browser Client
43	Browser Client May be Subject to Elevation of Privilege Using Remote Code Execution
44	Elevation Using Impersonation
55	Elevation Using Impersonation
56	Web Service May be Subject to Elevation of Privilege Using Remote Code Execution
57	Elevation by Changing the Execution Flow in Web Service
58	Cross Site Request Forgery

B.3 Threats for Encryption Proxy

B.3.1 Denial of service

ID	Name
17	Potential Excessive Resource Consumption for Encryption Proxy front-end or Non Relational Database
20	Potential Excessive Resource Consumption for Proxy and protocol analyzer or Non Relational Database
31	Potential Process Crash or Stop for Encryption Proxy front-end
32	Data Flow HTTPS Is Potentially Interrupted
37	Potential Process Crash or Stop for Proxy and protocol analyzer
38	Data Flow HTTPS Is Potentially Interrupted

B.3.2 Elevation Of Privilege

ID	Name
3	Elevation Using Impersonation
5	Elevation Using Impersonation
23	Elevation Using Impersonation
33	Encryption Proxy front-end May be Subject to Elevation of Privilege Using Remote Code Execution
34	Elevation by Changing the Execution Flow in Encryption Proxy front-end
35	Cross Site Request Forgery
39	Proxy and protocol analyzer May be Subject to Elevation of Privilege Using Remote Code Execution
40	Elevation by Changing the Execution Flow in Proxy and protocol analyzer
41	Cross Site Request Forgery

B.3.3 Information Disclosure

ID	Name
16	Weak Credential Storage
19	Weak Credential Storage

B.3.4 Repudiation

ID	Name
30	Potential Data Repudiation by Encryption Proxy front-end
36	Potential Data Repudiation by Proxy and protocol analyzer

B.3.5 Spoofing

ID	Name
2	Spoofing the Browser External Entity
15	Spoofing of Destination Data Store Non Relational Database
18	Spoofing of Destination Data Store Non Relational Database
21	Spoofing the Browser External Entity

B.3.6 Tampering

ID	Name
4	Proxy and protocol analyzer Process Memory Tampered
22	Cross Site Scripting

B.4 Threats for Privacy enhancing IDM

B.4.1 Denial Of Service

ID	Name
25	Potential Excessive Resource Consumption for PIDM REST API or Non Relational Database
28	Potential Excessive Resource Consumption for PIDM Front end or Non Relational Database
47	Potential Process Crash or Stop for PIDM REST API
48	Data Flow HTTPS Is Potentially Interrupted
53	Potential Process Crash or Stop for PIDM Front end
54	Data Flow HTTPS Is Potentially Interrupted
72	Potential Excessive Resource Consumption for Cordova plugin or HTML5 Local Storage
82	Potential Process Crash or Stop for Cordova plugin
83	Data Flow Generic Data Flow Is Potentially Interrupted

B.4.2 Elevation Of Privilege

ID	Name
19	Elevation Using Impersonation
22	Elevation Using Impersonation
31	Elevation Using Impersonation
49	PIDM REST API May be Subject to Elevation of Privilege Using Remote Code Execution
50	Elevation by Changing the Execution Flow in PIDM REST API
51	Cross Site Request Forgery
55	PIDM Front end May be Subject to Elevation of Privilege Using Remote Code Execution
56	Elevation by Changing the Execution Flow in PIDM Front end
57	Cross Site Request Forgery
74	Elevation Using Impersonation
84	Cordova plugin May be Subject to Elevation of Privilege Using Remote Code Execution
85	Elevation by Changing the Execution Flow in Cordova plugin

B.4.3 Information Disclosure

ID	Name
24	Weak Credential Storage
27	Weak Credential Storage
30	Weak Authentication Scheme
63	Weak Authentication Scheme
86	Weak Authentication Scheme

B.4.4 Repudiation

ID	Name
46	Potential Data Repudiation by PIDM REST API
52	Potential Data Repudiation by PIDM Front end
80	Potential Data Repudiation by Cordova plugin

B.4.5 Spoofing

ID	Name
20	Spoofing the Browser External Entity
23	Spoofing of Destination Data Store Non Relational Database
26	Spoofing of Destination Data Store Non Relational Database
70	Spoofing of Destination Data Store HTML5 Local Storage

B.4.6 Tampering

ID	Name
17	Browser Client Process Memory Tampered
18	Cross Site Scripting
21	Cross Site Scripting
29	PIDM REST API Process Memory Tampered
68	Replay Attacks
69	Collision Attacks
71	Authenticated Data Flow Compromised
73	Browser Client Process Memory Tampered
79	Potential Lack of Input Validation for Cordova plugin
87	Replay Attacks
88	Collision Attacks
89	Authenticated Data Flow Compromised

B.5 Threats for Infrastructure Auditing

B.5.1 Tampering

ID	Name
1	Authenticated Data Flow Compromised
11	Collision Attacks
12	Replay Attacks
21	Collision Attacks
22	Replay Attacks
29	Collision Attacks
31	Replay Attacks
34	TOPOCERT Tool Process Memory Tampered
36	TOPOCERT Tool Process Memory Tampered
38	TOPOCERT Tool Process Memory Tampered
46	JavaScript Object Notation Processing
53	JavaScript Object Notation Processing
60	JavaScript Object Notation Processing

B.5.2 Denial Of Service

ID	Name
4	Potential Excessive Resource Consumption for Audit-Profiles Management front-end or audit profiles store
7	Potential Excessive Resource Consumption for Audit-Profiles Management front-end or logger
14	Potential Excessive Resource Consumption for Infrastructure Auditing Management front-end or logger
17	Potential Excessive Resource Consumption for Infrastructure Auditing Management front-end or audit profiles store
24	Potential Excessive Resource Consumption for GeoSeparation front-end or audit profiles store
26	Potential Excessive Resource Consumption for GeoSeparation Web Service or logger
41	Potential Process Crash or Stop for Audit-Profiles Management front-end
42	Data Flow HTTPS Is Potentially Interrupted
48	Potential Process Crash or Stop for Infrastructure Auditing Management front-end
49	Data Flow HTTPS Is Potentially Interrupted
55	Potential Process Crash or Stop for GeoSeparation front-end
56	Data Flow HTTPS Is Potentially Interrupted

B.5.3 Spoofing

ID	Name
3	Spoofing of Destination Data Store audit profiles store
6	Spoofing of Destination Data Store logger
13	Spoofing of Destination Data Store logger
16	Spoofing of Destination Data Store audit profiles store
23	Spoofing of Destination Data Store audit profiles store
28	Spoofing of Destination Data Store logger

B.5.4 Information Disclosure

ID	Name
5	Authorization Bypass
8	Authorization Bypass
10	Weak Authentication Scheme
15	Authorization Bypass
18	Authorization Bypass
20	Weak Authentication Scheme
25	Authorization Bypass
27	Authorization Bypass
32	Weak Authentication Scheme

B.5.5 Repudiation

ID	Name
40	Potential Data Repudiation by Audit-Profiles Management front-end
47	Potential Data Repudiation by Infrastructure Auditing Management front-end
54	Potential Data Repudiation by GeoSeparation front-end

B.5.6 Elevation Of Privilege

ID	Name
2	Elevation Using Impersonation
9	Elevation Using Impersonation
19	Elevation Using Impersonation
30	Elevation Using Impersonation
33	Elevation Using Impersonation
35	Elevation Using Impersonation
37	Elevation Using Impersonation
39	Elevation Using Impersonation
43	Elevation Using Impersonation
44	Audit-Profiles Management front-end May be Subject to Elevation of Privilege Using Remote Code Execution
45	Elevation by Changing the Execution Flow in Audit-Profiles Management front-end
50	Elevation Using Impersonation
51	Infrastructure Auditing Management front-end May be Subject to Elevation of Privilege Using Remote Code Execution
52	Elevation by Changing the Execution Flow in Infrastructure Auditing Management front-end
57	Elevation Using Impersonation
58	GeoSeparation front-end May be Subject to Elevation of Privilege Using Remote Code Execution
59	Elevation by Changing the Execution Flow in GeoSeparation front-end