Project number: 653454
H2020-DS-2014-1

Project start: October 1, 2015
Project duration: 3 years

D5.1
Functional Design

Document Identification	
Due date	March 31, 2017
Submission date	March 31, 2017
Revision	1.00

Related WP	WP5	Dissemination Level	PU
Lead Participant	ATOS	Lead Author	Nicolás Notario (ATOS)
Contributing Beneficiaries	AIT, FOKUS, GUF, TUG, OTE, SIC	Related Deliverables	D2.1, D4.1, D3.3

Abstract: This deliverable follows a solid architecture design methodology to describe, from a technical and a logical point of view the components and sub-components that will build CREDENTIAL system (e.g. the CREDENTIAL Wallet, the CREDENTIAL Mobile App, etc.). The deliverable, aligned with privacy by design and by default principles, also documents the most privacy-relevant decisions taken during the design phase, providing the rationale behind them. This work will serve as the blueprint to the next stage of the project, which is the development of the described components.

This document is issued within the CREDENTIAL project. This project has received funding from the European Union's Horizon 2020 Programme under grant agreement no. 653454.

Executive Summary

The CREDENTIAL system must support a set of generic business and logical use cases that are the result of the generalization of use cases linked to three different scenarios related to different domains. These use cases have been defined in a separate deliverable through the collaboration of scenario or pilot partners and all other CREDENTIAL technical and research partners, so they are relevant for the end users but also so they can showcase the truly potential of CREDENTIAL and its features. In order to come up with a functional design for the CREDENTIAL system, it was agreed to follow Kruchten's 4+1 architectural view model that relies on multiple views from the system (logical, development, process, physical and use case) that allows agreeing to a common understanding of the system with different stakeholders. The list of principles that were to govern the design process was also agreed upon at the beginning of the design stage. The principle that has driven many of the most relevant architectural decisions is the **privacy by design** one. The design of the system has considered the privacy implications of the different available choices before making the final decisions.

The logical view is comprised of 17 services. Some of these services will be required to run in the cloud while the others will have to be provided in the participant's domain (a CREDENTIAL or a third party mobile application or integrated with service providers' platforms). Finally, some of them will have to run jointly, in the cloud and in the participant's domain.

The development architecture provides a more detailed view of the system to be developed. The services to be run in the cloud are considered to be a part of the CREDENTIAL Wallet and are grouped, depending on their purpose in an "Identity and Access Management" and a "Data Management" component. It was decided that services or functionalities bound to be run in the participant's domain will be packed as a "Participant Toolkit" which will ease the integration process. An additional category of services, the request filters, will provide a horizontal set of functionalities that have to be applied to most of the services: auditing, authorization and sanitization.

The process view demonstrates how the different components described in the development view interact towards the realization of the business and logical use cases that characterize the system.

Finally, **the physical view** depicts the underlying hardware (i.e. physical machines) and software (virtual machines, operating systems, etc.) that could support the operation of CREDENTIAL.

Regarding the design decisions taken during this design phase, these have been gather in the design logbook, that includes the different options considered and the main reasons for the final decisions. One of the most impacting decisions taken during the design of CREDENTIAL was to base the development on an open source existing IAM (OpenAM). This will provide CREDENTIAL a solid foundation on which to deploy the different enhancements (e.g. proxy re-encryption, redactable signatures, anonymous and multiple accounts or FIDO biometric authentication) that will support the different identified use cases and requirements.

Document information

Contributors

Name	Partner
Nicolás Notario	ATOS
Juan Carlos Pérez Baun	ATOS
Welderufael Tesfay	GUF
Florian Thiemer	FOKUS
Stephan Krenn	AIT
Christoph Striecks	AIT
Andreas Happe	AIT
Andreas Abraham	TUG
Felix Hörandner	TUG
Evangelos Sfakianakis	OTE
Simon Roth	SIC
Pritam Dash	SIC

Reviewers

Name	Partner
Luigi Rizzo	INFOCERT
Simon Roth	SIC
Pritam Dash	SIC

History

0.1	2016-02-23	Juan Carlos Pérez Baun	Document creation
0.2	2016-02-23	Nicolás Notario	Detailed Table of Contents
0.3	2016-03-03	Nicolás Notario	Development of methodology section
0.4	2016-03-30	Welderufael Tesfay	Privacy by design/default
0.5	2016-04-29	Nicolás Notario Florian Thiemer	Updated list of principles/objectives/constraints; Use case view draft
0.6	2016-06-02	Nicolás Notario	Added logical use cases from Redmine
0.7	2016-07-01	Nicolás Notario	Added logical architecture, data model section and development component template
0.8	2016-07-05	Nicolás Notario	New template
0.9	2016-07-27	Andreas Happe Felix Hörandner	Added first version of AIT development components; Added descriptions for development components
0.10	2016-09-30	Florian Thiemer Felix Hörandner	Add Notification Service description; Added Technical Sequence diagrams
0.11	2016-10-04	Florian Thiemer	Add technical sequence diagram no5
0.12	2016-10-10	Evangelos Sfakianakis	Added physical view section and respective diagrams
0.13	2017-01-11	Florian Thiemer	Add Notification Broker component to Notification Service description
0.14	2017-01-23	Stephan Krenn	Added interfaces for advanced cryptographic protocols
0.15	2017-01-28	Nicolás Notario	Add design logbook
0.16	2017-01-30	Andreas Abraham	Added information on OpenAM instantiation

0.17	2017-02-08	Evangelos Sfakianakis	Added scaling explanation in physical view
0.18	2017-02-19	Nicolás Notario	Executive Summary and Conclussions
0.19	2017-03-06	Nicolás Notario	Modifications following reviewers feedback
0.20	2017-03-13	Juan Carlos Pérez Baun	Add 13 Appendix B – CREDENTIAL Generic Mobile application
0.21	2017-03-21	Nicolás Notario	Addressing second round of reviewer comments
0.22	2017-03-22	Nicolás Notario	QA version produced
0.23	2017-03-24	Nicolás Notario	Addressing QA team comments
1.00	2017-03-28	Nicolás Notario	Final version

Contents

List of Figures

List of Tables

List of Acronyms

ABC	Attribute-Based Credentials
ACF	Access Control Filter
ACL	Access Control List
API	Application Program Interface
ARM	Advanced RISC (Reduced Instruction Set Computer) Machine
BUC	Business Use Case
CORS	Cross-Origin Resource Sharing
CPU	Central Processing Unit
CRAM	Challenge Response Authentication Method
DB	Database
DRM	Digital Rights Management
eIDAS	Electronic Identification and Signature
EU	European Union
FCGSS	FCG Software Solutions
FIDO	Fast Identity Online
GCM	Google Cloud Messaging
GDPR	General Data Protection Regulation
HCI	Human Computer Interface
HD	Hard Drive
HTTP	HyperText Transfer Protocol
HW	Hardware
IAM	Identity and Access Management
IdM	Identity Management
IP	Internet Protocol
IT	Information Technology
JAX-RS	Java API (Application Programming Interface) for Restful Web Services
JWT	JSON Web Token
LDAP	Lightweight Directory Access Protocol
LTS	Long Term Support
LUC	Logical Use Case
MQTT	Message Queuing Telemetry Transport
OAuth	Open Authentication
OpenID	Open Identity
OSI	Open Systems Interconnection
OSI	Operating System
OWASP	Open Web Application Security Project
PC	Personal Computer
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PIN	Personal Identification Number
PIP	Policy Information Point

QR	Quick Response
RAM	Random Access Memory
REST	Representational State Transfer
RO	Resource Owner
RPT	Relying Party Token
SAML	Security Assertion Markup Language
SAS	Serial Attached SCSI (Small Computer System Interface)
SATA	Serial Advanced Technology Attachment
SMS	Short Message Service
SOLID	Single responsibility principle, Open/closed principle, Liskov substitution principle, Interface segregation principle, Dependency inversion principle
SP	Service Provider
SQL	Structured Query Language
SSD	Solid State Discs
SSO	Single Sign On
STORK	Secure Identity Across Borders Linked
SW	Software
TCP	Transmission Control Protocol
TEE	Trusted Execution Environment
U2F	Universal 2nd Factor
UAF	Universal Authentication Framework
UI	User Interface
UMA	User Managed Access
UML	Unified Modelling Language
URL	Uniform Resource Locator
USB	Universal Serial Bus
UUID	Universal(ly) Unique Identifier
VM	Virtual Machine
XACML	eXtensible Access Control Markup Language
XML	Extensible Markup Language

1 Introduction

One of the major goals of the CREDENTIAL project is to develop an innovative cloud based service for managing and sharing personal data. The resulting system will be required to address the different needs expressed through the use cases by piloting partners and will need to include the researching results from the different research partners applicable to different areas (e.g. authentication or cryptography).

This report provides the blueprint that will support the development of each of the subcomponents that will be deployed for the piloting phase, for which it follows a solid engineering methodology. Instead of directly addressing requirements related to the pilots, the design focuses on supporting the generalization of such requirements, ensuring the design remains as scenario-agnostic as possible. This deliverable will be the starting point for the development efforts that will result in different components and its final integration in D5.6 "Reference environment" and its instantiation in the different pilots.

1.1 Scope

The objective of this deliverable is to serve as a guide to develop the CREDENTIAL Wallet and other necessary CREDENTIAL components. In this deliverable, the development components that will have to be built, as well as how they interact, are described in a generic way, meaning that there are no particular technological constraints and that anyone can choose to implement the system can choose e.g. the development language or platform. However, and as the goal of CREDENTIAL is to build and pilot an actual system, the generic design is then instantiated upon a technology/platform chosen by CREDENTIAL consortium (see Appendix A – OpenAM Instantiation).

This deliverable shows the results of multiple activities that have been conducted by CREDENTIAL's partners:

Deliverable activity	CREDENTIAL Objectives
Identify and define, in close collaboration with end-user partners, the logical components that will provide CREDENTIAL generic functionality. Iterate upon the identified logical components identifying and defining corresponding development components.	Elaborate a functional and technical design based on the requirements identified with the collaboration of end-user partners
Periodical meetings with end user partners in order to discuss and address the design in relationship to security and privacy requirements	Align the functional and technical design with security- and privacy-by-design principles

Table 1: Activities vs Objectives

Deliverable 5.1 is part of a bigger effort which is to design a Cloud Identity Wallet addressing end-users and businesses' needs. Figure 1 shows the positioning of this deliverable in such global picture. This deliverable will provide a design which takes into account the generalized business and logical use cases. The design efforts will be responsible of identifying and specifying the components that to be developed in order to materialize the functional and non-functional requirements. Pilots will be specified to

demonstrate the relevant domain-specific use cases and in accordance to the instantiation of specified generic functional design.

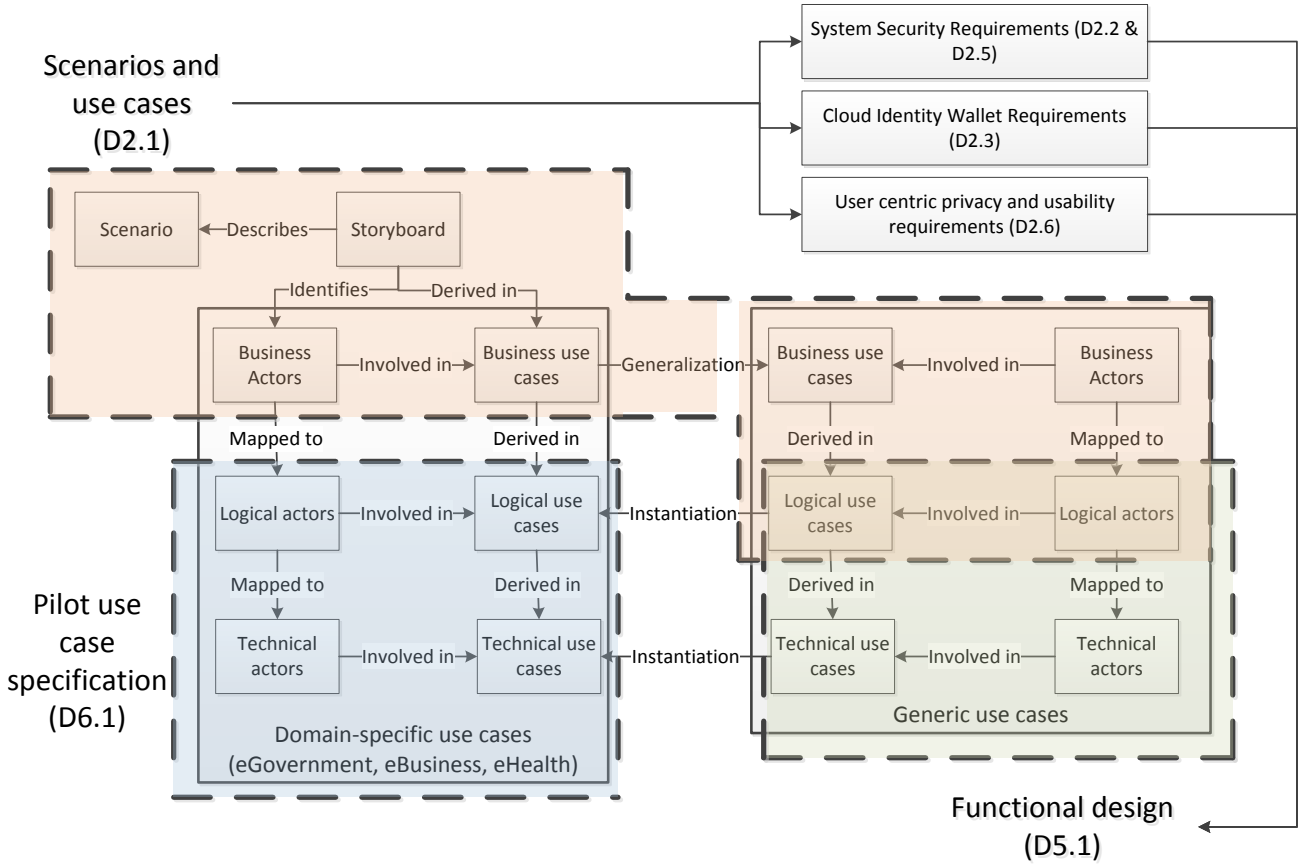


Figure 1: D5.1 as Part of a Broader Mission

1.2 Relation to Other Deliverables

The following list briefly describes other related deliverables and their connection to this deliverable:

D2.1: Scenarios and use-cases: D2.1 provides the list of generic logical use cases that the CREDENTIAL Wallet and related systems must satisfy;

D4.1: Assessment report on cryptographic technologies, protocols and mechanisms: D4.1 provides an assessment of technologies considered for CREDENTIAL. Given the similar timeline, periodical alignments have been part of the work in order to take into account D4.1 considerations for the final design;

D3.3: Recommendations on privacy enhancing mechanisms: D3.3 has raised questions and recommendations that have been considered for the final design. Periodical synchronizations have been necessary to consider their work as it run parallel to D5.1 work.

D5.2 Security protocols early prototype library and **D5.3 IAM early prototype library**: the prototypic libraries will be built following the design provided in this deliverable

D5.4 Security protocols reference component library and **D5.5 IAM reference component library**: the final version of the libraries will be also built following the design provided in this deliverable

D2.6 User centric privacy and usability requirements: some of the requirements will be identified by looking at the actual design of the system and the requirements will drive some design decisions

D2.2 System Security Requirements: some of the requirements will be identified by looking at the actual design of the system and the requirements will drive some design decisions

D2.2 Cloud Identity Wallet Requirements: some of the requirements will be identified by looking at the actual design of the system and the requirements will drive some design decisions

Many other deliverables indirectly depend or are influenced by this deliverable but do not rely on this deliverable as an input. E.g**. D3.2 UI Prototypes V2 and HCI Patterns** will provide UI mockups for functionality that has been identified in the use cases mentioned in this deliverable.

1.3 Outline

This document in structured in three main parts:

- Section 2 identifies what are the main architectural principles, objectives and constraints that will govern all design decisions, taking into account that the main objective is to fulfil end-user requirements (identified and specified in deliverables D2.1and D6.1).
- Sections 4, 5, 6, 7, 8 provide different views of CREDENTIAL's Wallet design (i.e. logical, process, development) as proposed in the design methodology described in Section 2.
- Section 9 includes a justification of major design decisions in order to provide some level of design accountability.

Finally, Section 10 provides some closure to the first design process' step and identifies what are the next challenges and steps for the design and implementation of the Wallet.

Three Appendices complement this deliverable:

- Appendix A – OpenAM Instantiation: showing how the generic architecture described in section 6 can be instantiated over an existing Identity and Access Management product.
- Appendix B – CREDENTIAL Generic Mobile Application: highlighting key design aspects of the Mobile App to be developed.
- Appendix C – Mapping between Use Cases and Development Components: showing the relationship between development components and the use cases. This traceability matrix will provide some guidance during the final validation of the Wallet.

2 Design Methodology

It has been agreed between CREDENTIAL partners involved in the design of its Wallet, to follow an approach based on Kruchten's 4+1 architectural view model [1] in order to provide a functional design of the system. The main benefits of following this approach are:

- Provides useful information for different stakeholders (end-users, developers, project managers, security and privacy analysts…).
- Enables to navigate from development components to specific functionalities as described in the use cases (and vice versa).
- Fully compatible with prevalent standards such as UML.
- Easy to provide thumb-rules to ensure the integrity of the design.

Kruchten proposes to have 4+1 views (see Figure 2) that relate to each other and that will be described below.

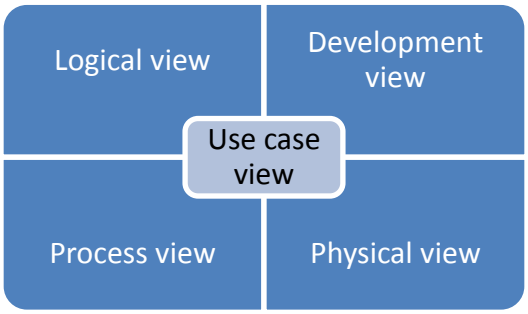


Figure 2: Kruchten's 4+1 View Model

2.1 CREDENTIAL's Approach

Following CREDENTIAL's pilot-driven bottom-up approach, for the design of the CREDENTIAL Wallet the scenarios and use cases described in deliverable D2.1will be used as the starting point. Actually the Wallet will only focuses on the identified generic use cases, as these are supposed to generalize the functionality required in each of the scenarios. Using these generic use cases and following a logical perspective of the system, the functional description of the different logical components and services will be provided. These elements will be mapped to development components, which will implement the described functionality and will be described with enough detail for CREDENTIAL implementation tasks. Interaction among development components will be demonstrated within the process view. There is a close relationship between both views and will be worked in parallel, with a constant interaction. Finally, a generic physical view of the system will be provided, showing how the different components can be physically deployed.

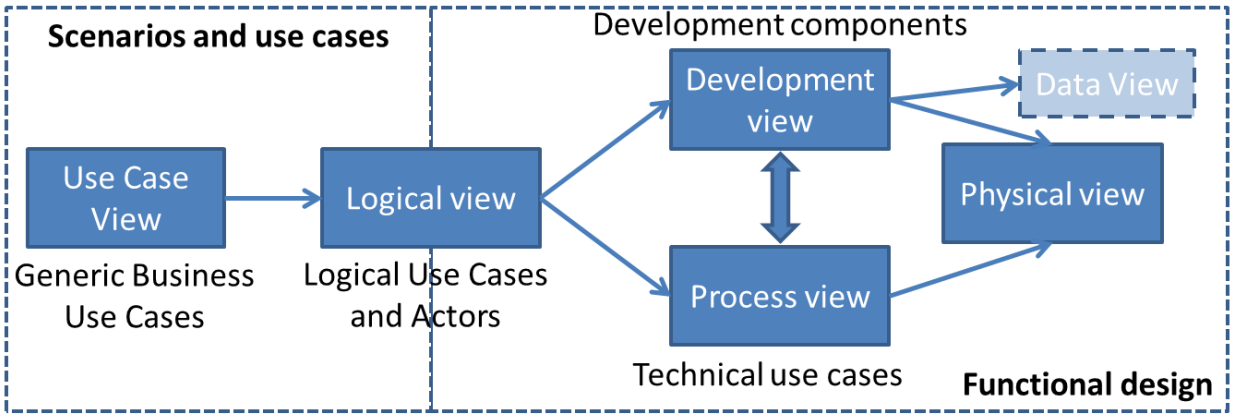


Figure 3: CREDENTIAL Functional Design Process

During the design process (see Figure 3) the security, privacy and legal requirements will be considered, along with the privacy recommendations. These requirements and recommendations will drive the final selection of technologies and mechanisms and how they cooperate in order to implement the CREDENTIAL Wallet use cases. During the process, all significant decisions will be justified and linked to use cases, requirements or recommendation in the design log book (see Section 9). A very relevant section of this deliverable is the one devoted to present the data model of the system which has been specially designed to enhance the privacy and security properties of the Wallet. This data model is explained with full detail in Section 6.1.

2.2 Use Case View

"The Use Case View encompasses the use cases that describe the behaviour of the system as seen by its end users and other stakeholders" [3]

The use case view "is redundant with the other ones (hence the '+1')" [3], but it serves two main purposes:

- As the set of requirements that the architecture design must fulfil
- As a validation and an illustration tool to ensure the design is complete

Business level diagrams are the proposed tool to capture this view.

2.3 Logical View

"The logical architecture primarily supports the functional requirements—what the system should provide in terms of services to its users" [1]

This view will focus on describing system's functionality in terms of abstract elements (mechanisms, services, actors). This view can be decomposed vertically according to functional areas (e.g. Identification and Access Management (IAM) vs data sharing) or horizontally, focusing on the different layers (i.e., presentation, service, business logic or data access).

Within CREDENTIAL, instead of using the derivation of Booch's notation [2], as suggested by Kruchten, we will use UML diagrams (as proposed by FCGSS [3]) which will still allow us to provide the static view of the system, as expected. The following UML diagram types can be considered:

- Class or structural diagrams to define the basic blocks and their operations
- Package diagrams if it is necessary to divide the model into logical containers
- Composite structure diagrams can be leveraged to understand the relationships and interfaces with other components

Each of the abstract components identified will be fully described and linked to the CREDENTIAL Wallet use cases so end-users can easily understand its functionalities and role. A high level view of the components' interfaces will also be provided.

2.4 Development View

"The development architecture focuses on the actual software module organization on the software development environment. The software is packaged in small chunks—program libraries, or subsystems—that can be developed by one or a small number of developers" [1]

While the logical view is related to a conceptual level, this view focuses on the actual components that will be developed or integrated as part of the CREDENTIAL Wallet framework. As in the logical view, the development view can be decomposed in layers (e.g. OSI layers or Model View Controller layers) to provide a better understanding of the system.

Each of the identified components:

- Will be mapped to logical components as they will implement the functionality described at a logical level;
- Will have an owner, meaning a partner that will be responsible for overseeing its implementation within CREDENTIAL;
- Will provide a detailed description of its functionality, technology, dependencies, license, APIs and interfaces;

The UML diagrams that will best represent this view are component diagrams.

The essential difference between the logical and the development views is that the logical one shows all the components and functionalities at a conceptual level, which not necessarily matches the actual components that will be part of the final system (development view).

2.5 Process View

"The process architecture takes into account some non-functional requirements, such as performance and availability. It addresses issues of concurrency and distribution, of system's integrity, of fault-tolerance, and how the main abstractions from the logical view fit within the process architecture—on which thread of control is an operation for an object actually executed" [1]

While the logical and development views focus on the static aspects of the system the process view is fully focused on its dynamic aspects. It describes how each of the system's components interacts with each other towards a common goal. According to the 4+1 architecture view model, the process architecture can be described at several levels of abstraction, each level addressing different concerns. For CREDENTIAL this view will focus on the development components. Architectural significant business and logical use cases will be decomposed in terms of development components, resulting in a set of **architecturally significant generic technical use cases** that will provide, along with a set of legal, security and privacy requirements and recommendations, the functional description of the system.

Each of the technical use cases will be mapped to:

- One or several CREDENTIAL Wallet business or logical use cases;
- One or several development components identified in the development or logical views

The UML diagrams that better capture the dynamicity of a system are: sequence, communication, activity, timing and interaction overview diagrams.

2.6 Physical View

The different development components need to be mapped to where they are going to be executed in a network of computing devices

In the case of CREDENTIAL it will be very relevant to identify where each of the development components will be deployed as it has many implications regarding security, privacy and legal compliance.

The physical architecture that will be presented in this view represents a generic view of the system and must be instantiated into particular architectures and physical environment depending on the pilots and their specific needs.

In order to display the system physical's architecture, the most relevant UML diagram is the deployment diagram, which shows physical disposition of the different components in an operational environment. Each of the identified nodes (e.g. embedded such as an application server or physical nodes such as mobile devices) will have a description and may have a list of associated requirements (i.e. the mobile phone must be compatible with Android starting from version X.x and with secure hardware (HW) capabilities such as fingerprint reader).

3 Architectural Goals, Principles and Constraints

While CREDENTIAL consortium has decided to follow the 4+1 framework for the definition of the CREDENTIAL Wallet system, it is also necessary to identify which architecture principles will govern the design. These principles "are proven guidelines that you should apply when developing or modifying an architecture" [4]. While the selection of these principles is highly coupled to the objectives of the system to be developed or designed, there are ten basic and generic principles [4] that should always be considered and that will be certainly followed within CREDENTIAL.

3.1 Generic Principles

3.1.1 Loose Coupling

In system engineering, coupling is a qualitative measure regarding how much of one component depends on the inner workings of another one. When one component depends on the inner implementation of a different component it implies that a change in the dependable component will affect the dependent components, meaning additional effort, risk of adding bugs etc.

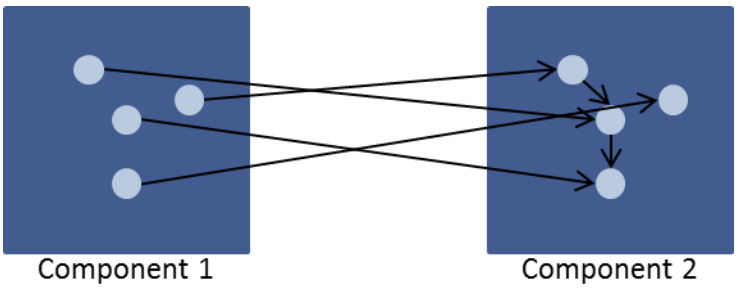


Figure 4: Tightly Coupled Components

Some design decisions that highly affect the level of coupling in a system are data structures. System-wide defined structures minimize the level of coupling while the usage of component-specific structures in other components, inherently increases coupling.

In CREDENTIAL we will achieve the loose coupling objective by ensuring that each individual component can only interact with others by exposing and consuming a clear API, which will work as a contract between components and will ensure that each component is fully independent of the internal implementation of every other one. Other means to ensure loose coupling that will be considered are the usage of queues and standards for communication.

The major advantages of having a loosely coupled system is that its components are exchangeable (with other components implementing the same API), independently updateable (modifications in one component do not affect other ones), more testable (by using mock objects implementing the same API) and that we will avoid dependencies regarding technological choices such as implementation language, development platform etc.

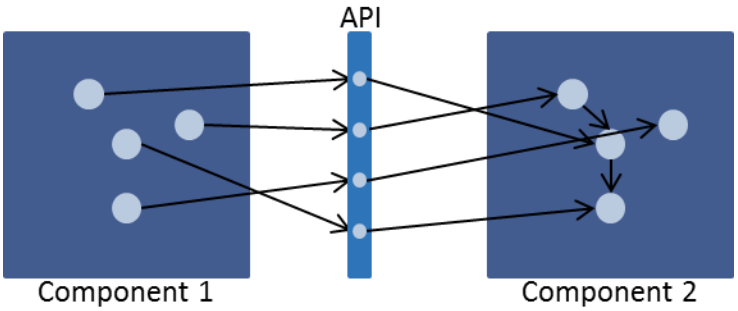


Figure 5: Loosely Coupled Components

3.1.2 High Cohesion

High cohesion is a desired property of a design which measures the dependencies within a specific component (loose coupling referred to dependencies among components). In CREDENTIAL we will achieve our high cohesion objective by ensuring that each individual component is oriented to one unique task and splitting into separate subcomponents whenever necessary. High cohesive modules are preferred as they are associated to desirable properties such as: robustness, reliability, reusability, understandability and maintainability.

3.1.3 Design for Change

IT Systems are not static but constantly evolving products that will be required to include new functionalities and will have to address existing issues or bugs overtime.

In CREDENTIAL we will provide a highly flexible design taking into account that the system will evolve in time and will include mechanisms (e.g. an abstraction layer) that will allow its extensibility (e.g. new cryptographic methods) or modifiability with minimal costs in terms of time or development. The usage of clear APIs, already discussed as useful to achieve other architectural goals, is also essential to provide a flexible design.

3.1.4 Separation of Concerns

This principle is much related to other principles such as modularity, loose coupling, high cohesion, etc. The Separation of Concerns principle states that "you should separate different aspects of a problem and deal with each of these sub-problems individually" [4]. It is the engineering application of the roman "divide and conquer" policy.

In CREDENTIAL we will follow a layered design, encapsulating logical functionality and concepts into specialized development components, separating high-level aspects such as user interface, storage or processing.

3.1.5 Information Hiding

Also related to loose coupling and high cohesion principles, the inner aspects of a component should be hidden from other components, which will prevent strong coupling. As already explained, in CREDENTIAL we will provide well-defined APIs that will only expose information and functionality required by other components, abstracting and hiding the implementation details. This APIs will allow the components to work as a "black box" which is a concept that realizes the information hiding principle.

3.1.6 Abstraction

According to [4] "abstraction means focusing on aspects of a concept that are relevant for a specific purpose and neglecting unimportant details for this particular purpose". It is useful to realize loose coupling and is also entangled with other principles such as information hiding. In CREDENTIAL, the design and implementation will follow the following techniques:

- Define explicit interfaces that will be the only means to communicate with the components;
- Separate the definition and implementation of the API
- Whenever applicable, follow object-oriented SOLID principles:
  - Liskov substitution principle[5]: components in a system should be replaceable with other components with same API without altering the correctness of that program
  - Interface segregation principle: “Many client-specific interfaces are better than one general-purpose interface” [5]
  - Dependency inversion principle: one should “Depend upon Abstractions. Do not depend upon concretions" [5]

3.1.7 Modularity

Today the complexity of computer software is growing more and more. One possible reason for the growing complexity is the increasing number of software requirements. Complex software can produce issues in extending or maintaining the software itself. One way to solve this issue is to use modularity in the software architecture. Modularity can be seen as one of the most important goals in software architecture.

Modularity is a concept used in many different fields to make complex systems easier to handle. This concept follows the idea of splitting the whole system into smaller units the so-called modules. Each module is structurally independent from each other but work together. These modules are powerful elements which are relatively weakly connected. In other words, the whole system architecture can be seen as a sum of many powerful modules which have a weak connection to one another. The idea of modularity can be described by abstraction, information hiding and interface [12].

Utilizing modularity has many benefits for the CREDENTIAL project. Basically, modularity makes complexity easier to manage and it can also accommodate potential changes. It also enables parallel work because of the loose connection. Another important advantage using modularity in CREDENTIAL is that it will be easier to extend and maintain the system. Moreover, CREDENTIAL will become more tolerant to the uncertainty of using modularity. Altogether, utilizing modularity makes the system more flexible and easier to manage.

3.1.8 Traceability

This principle of Traceability, Self-documentation (Section 3.1.9) and Incrementality (Section 3.1.10) ones are more related to the design process than to the design itself. Authors of Software Architecture – A comprehensive framework and guide for practitioners [4] state that "the traceability of architectural structures and decisions is important to ensure that an architecture can be understood" and that "should also be possible to assign a requirement to the system building blocks that implement it" [4]

The chosen design process for CREDENTIAL enables full traceability from the requirements and use cases to the development components physically deployed. Each generic use case is built as a generalization of scenario-specific needs. Logical components identified in Section 5 are mapped to these business use cases and are also linked to development components described in Section 6, which will be responsible for satisfying the identified related requirements. Also, Section 9 provides a logbook where complex decisions are explained and traceable.

3.1.9 Self-Documentation

This principle tries to ensure that the information regarding a development component is embedded in the own component. Self-documenting components partially avoid the de-synchronization between development components and its documentation. While this deliverable provides the base design for the components, CREDENTIAL will propose developers to rely in language-specific best practices and tools to achieve self-document within the actual components' implementation.

3.1.10 Incrementality

As it is very hard to provide a full design right away, the design process in CREDENTIAL includes two iterations, with the provisioning of an initial design and a refinement period before providing the final version of the architecture. However, this architecture will not be considered definitive and further modifications in later phases will be allowed if required.

3.2 CREDENTIAL Principles

Besides the generic principles, identified in previous section, CREDENTIAL has agreed to consider some additional objectives/principles related to the scope of CREDENTIAL, its privacy and security objectives and its cloud-orientation:

3.2.1 Minimize Modifications in Existing Services

A large number of companies are using software every single day. Each company has its own requirements related to the software they use. Therefore, the demand of interoperability is increasing. If a company decides to buy a new software application to extend their services, it can be a tough decision to choose the right application. One important decision making point will be based on how difficult it is to integrate the new software into the company's existing services. If significant changes need to be made to adapt the new software, companies might think of alternative software to use or try to develop their own solution.

In order to be able to adapt new applications to a wide range of services, the application acceptance has to be high. A possible reason for struggling in acceptance is if the service provider has to modify its services to be able to use the application. Within CREDENTIAL we aim to minimize the modifications in existing services. This aim is one of CREDENTIAL's architectural goals used to design the system architecture. This goal should keep the effort of the service provider as small as possible to integrate our cloud identity Wallet. For example, this goal can be reached by utilizing standards in communication protocols, standardized interfaces or standardized libraries.

3.2.2 Scalable

In CREDENTIAL's context scalability is the property of a system that reflects its capacity to handle a large amount of work, or to easily grow to accommodate such work. CREDENTIAL's vocation as a

cloud Wallet, connecting data subjects and other participants (such as other data subjects or online services) and its business objectives will require the system to scale according to the usage demand. While there are two types of scalability, horizontal and vertical, it is only the latter which is of interest for CREDENTIAL:

- **Vertical scalability** means adding more resources to one processing node, i.e. adding an extra processor to a computer. This kind of scaling has no impact on the architecture as it is fully transparent to the system.
- **Horizontal scalability** implies having a cluster of "nodes" and adding extra ones to scale. Horizontal scalable systems have to consider this in its design as it is not transparent (i.e. requests from one user may be processed in different nodes).

Horizontal scalability is considered more important as commodity hardware (typically used in clouds) is cheaper compared to cost of special configuration hardware (super computer).

The scalability feature will be addressed in CREDENTIAL through several design decisions:

- **Partition the system in logic layers** that can be individually deployed and scaled;
- Use **clustering technologies** such as load balancing, auto scaling, etc.
- Promote **stateless** components and features: having stateless features and components implies that each request can be addressed by an independent machine (physical or virtual) without any overhead, making transparent to the users the removal or addition of extra machines. This approach is the preferred for horizontally scalable systems.

3.2.3 Privacy by Design

Privacy has been regarded as fundamental human right [11], whose values have to be translated into the information and communications system design and implementation exercises. To address this, researchers and practitioners have suggested the notion of incorporating privacy requirements into the system design. Hence, software engineering processes should consider privacy protection of users from the onset of the system design and development [8]. This includes collecting privacy requirements, embedding privacy into the system design, and analysing privacy preservation or violation in the information flow. Furthermore, in order to mitigate privacy concerns of users and fulfil data protection regulations, the system development has to rigorously follow privacy-by-design principles and guidelines [9][10]. Likewise, the CREDENTIAL project will follow privacy-by-design principles and guidelines when designing the architecture and pilot systems.

D2.6 of the CREDENTIAL project will deliver a good-and-must have wish-list of privacy requirements that the core CREDENTIAL architecture and the three pilots will have to follow. These privacy requirements are also mapped into usability requirements. By considering these requirements, not only the CREDENTIAL technology will abide by the EU data protection regulations, but also showcase the fact that privacy preservation doesn't present significant hurdles to system usability.

At the core of the privacy-by-design concept reside the following principles and design patterns that CREDENTIAL will have to achieve during its lifetime.

Principle	Description

Data minimization	The amount of personal data that is collected or processed should be restricted to the minimal amount possible. For CREDENTIAL, this would, for example, mean implementing privacy preserving authentication techniques that require minimal set of user credentials. Therefore, the design of the architecture is constrained to fulfil this requirement.
	Techniques: Privacy preserving IdMs, Privacy-ABCs, Redactable signatures
Control	Data subjects should be provided the means to control as to how much and what kind of their information is processed by service providers.
	Techniques: User-centric IdMs
Separation	Personal data has to be processed in distributed manners in order to avoid complete presence of personal data at a single location thereby making user profiling an easy task. In CREDENTIAL, this would translate into, for example, separating the IdM and the services.
	Technique: federated systems
Transparency	Data subjects have the right to get informed about which information of theirs is processed, for what purpose, and by which means. In CREDENTIAL, this could be applied by creating a set of services that will show participants to who, when, and under what sharing preferences other participants have accessed their data.
	Techniques: Transparency enhancing mechanisms
Hide	Personal data and other (indirect) data related to a given user have to be kept technically hidden, therefore not available in plain and viewable form. In CREDENTIAL this will be achieved by using re-encryption techniques that will hide data from other participants than the ones authorized by the owner of the data. I.e. the cloud provider will not be able to see non-encrypted data
	Techniques: anonymisation techniques, encryption techniques

3.2.4 Privacy by Default

Privacy by default requires to optimally setting privacy preferences as the user starts to use a given service. Due to the fact that privacy settings play an important role in protecting users' privacy, the service providers are expected (and now obliged to, Art. 25 of REGULATION (EU) 2016/679 [12] "… *controller shall implement appropriate technical and organizational measures for ensuring that, by default, only personal data which are necessary for each specific purpose of the processing are processed. That obligation applies to the amount of personal data collected, the extent of their processing, the period of their storage and their accessibility. In particular, such measures shall ensure that by default personal data are not made accessible without the individual's intervention to an indefinite number of natural persons...*") to provide users privacy preferences that are transparent to their privacy. Furthermore, service providers must provide control to the users over the ways their data is collected, used, and stored.

In line with these requirements, the CREDENTIAL project should provide default privacy settings that protect the user's privacy. By providing optimal privacy preferences, the CREDENTIAL project will also save users from the hurdles of setting own privacy preferences. Particularly, these privacy preserving default settings will be of importance in the pilots, e.g., e-Health. Furthermore, the CREDENTIAL project should also provide users with the cross-border compatibility by default, in that users would not have to go through steps to perform credentials translations. This will be, for example, required in the e-Government pilot where citizens will perform cross-border electronic transactions.

4 Use Case View

CREDENTIAL will be integrated in three pilots covering the domains e-Government, e-Health and e-Business. Each pilot brings different challenges for CREDENTIAL technologies. The e-Government pilot targets Identity Management, the e-Health pilot targets sharing of sensitive medical data across multiple users and the e-Business pilot targets sharing of data with service providers. This section briefly introduces the use cases which have been described in D2.1 and D6.1. In order to understand the functionality of a potential CREDENTIAL System, an understanding of the stakeholders in scenarios which includes CREDENTIAL technology should be clear. The mentioned use cases were collected and described in a project-specific Redmine[1] instance, with links that allow moving between generic and scenario-specific use cases as well as from functional, to business and to technical use cases. While this deliverable is focused on the generic aspects of the system, and this section specifically in business use cases, the followed methodology ensures that all scenario specific considerations are taken into account. The rest of the section will describe, at a very high level, the system's actors and the main functionalities that will be provided.

The main actors for CREDENTIAL as a System are:

CREDENTIAL Wallet: The CREDENTIAL Wallet stores user data and identity data in a secure cloud. It is a cloud platform which offers sharing of those participant's data with other participants

CREDENTIAL Mobile Application: A specific mobile application that allows the user to securely interact with the Wallet. While not in the scope of the project, additional CREDENTIAL (e.g. desktop) applications could be also developed after the project. The mobile application is the main way for participants to interact with the cloud

CREDENTIAL Participant: A CREDENTIAL Participant is any person or service interacting with the Wallet for storing or accessing data. Two main kinds of participants are considered:

- Users: natural persons using the CREDENTIAL Wallet to manage their own data or to interact with other participant's data
- Service providers: entities providing an external service, offering domain specific functionality and that access or stores data in the Wallet but are not directly tied to a natural person. Service providers may have many users behind interacting with the data (e.g. a hospital and its doctors)

The scenarios for the CREDENTIAL System can be composed in three different categories:

- Identity Management
- Data Sharing
- Account Management

[1] http://www.redmine.org/

The Identity Management use cases focuses on integrating identity data stored within the CREDENTIAL Wallet in the authentication mechanisms towards other service providers. The use of proxy re-encryption technologies allows sharing the identity data in the CREDENTIAL Wallet in a secure and privacy aware way.

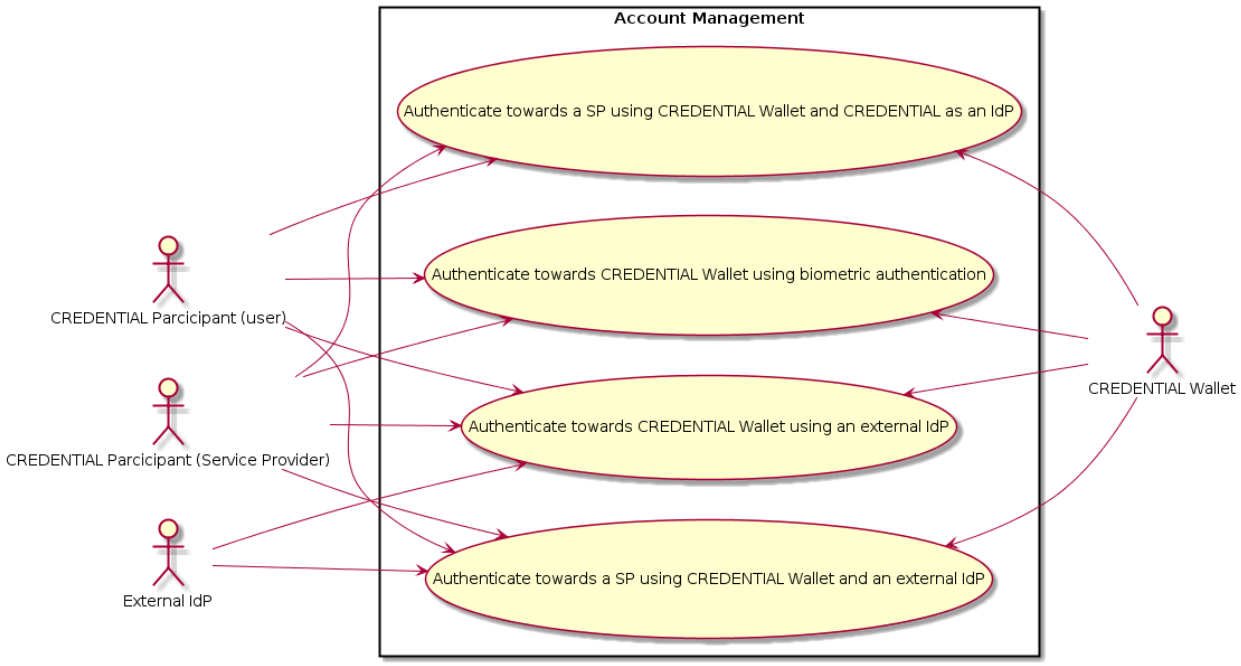


Figure 6: Identity Management Related Use Cases

Figure 6 gives a short overview of the Identity Management related use cases in which CREDENTIAL technology can be used. They introduce the concept of CREDENTIAL as an IDP which authenticates users against the Service Provider. CREDENTIAL as an IDP includes technology to re-encrypt Identity Attributes that are encrypted and provided by the CREDENTIAL Wallet.

The Data Sharing use cases focus on storing, reading and sharing of user data that is assigned to the CREDENTIAL Wallet. The user data is protected by encryption and during sharing of the user data it is never disclosed to the CREDENTIAL Wallet itself.

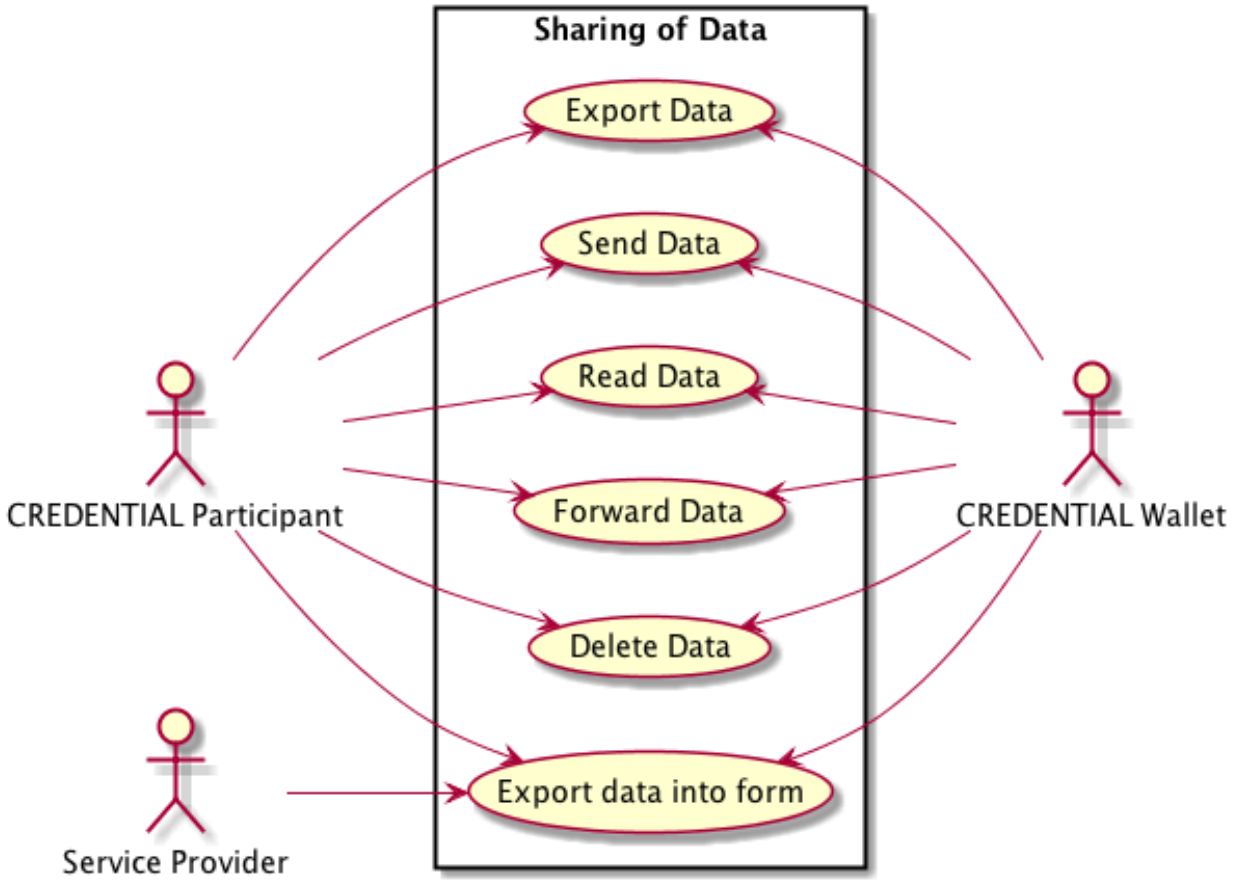


Figure 7: Data Sharing related Use Cases

Figure 7 shows the use cases that are related to data sharing. The first five use cases starting from "Export Data" until "Delete Data" involve end users and how they can use the CREDENTIAL Wallet's functionality as a Data Sharing Platform. By using proxy re-encryption technology the data can be re-encrypted by the CREDENTIAL Wallet afterwards without disclosing any sensitive information towards the CREDENTIAL Wallet. The "Export data into form" use case shows how a CREDENTIAL Participant can use the CREDENTIAL Wallet in order to fill a web form that will be posted to the service provider.

The Account Management Use Cases focus on the whole account life-cycle and access management. A user can create a new account which involves the creation of his proxy re-encryption enabled key material and an account association on the CREDENTIAL Wallet. Furthermore, the user can perform various management functionalities like showing an activity protocol on his data or delegate access rights to his data.

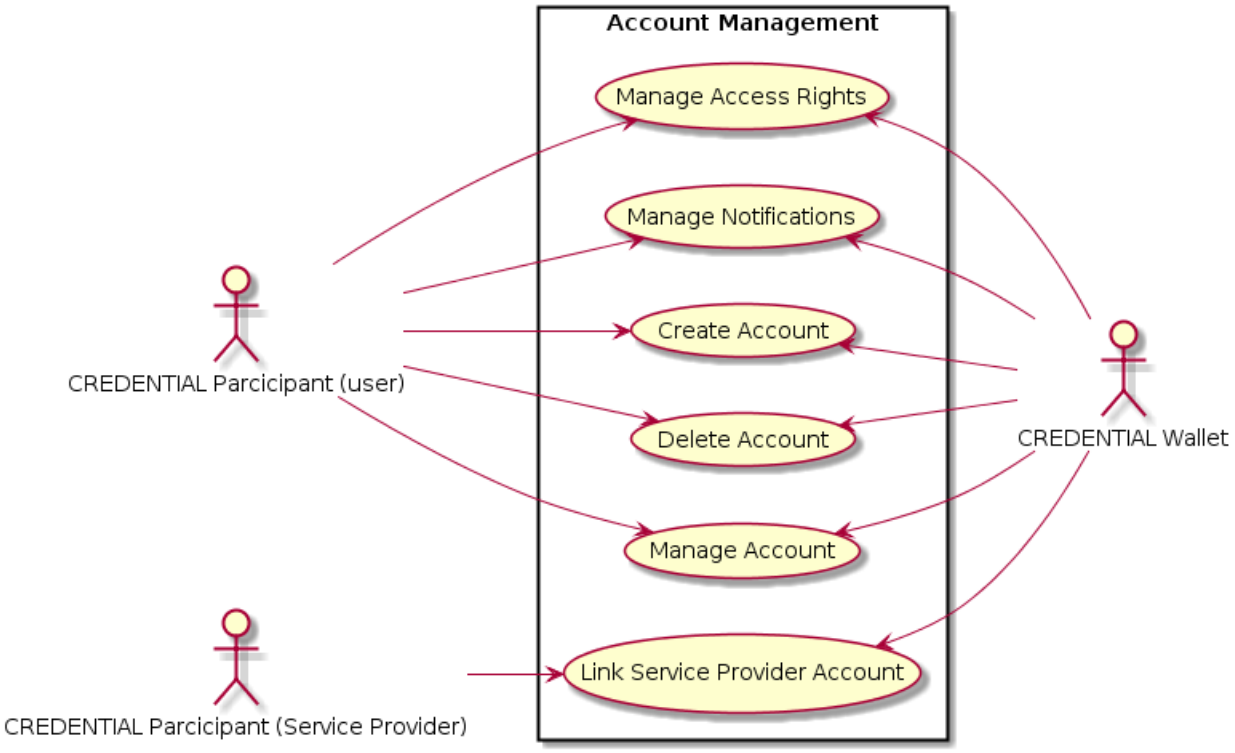


Figure 8: Account Management

Figure 8 shows the Account Management use cases and how they are related to the actors. “Manage Access Rights” bundles all the use cases that are related to requesting, delegating, granting and revoking of access rights. A Notification mechanism is included by the CREDENTIAL Wallet which allows CREDENTIAL Participants with proper access rights to subscribe to specific events on the data stored in the CREDENTIAL Wallet. E.g. update of a data set where a CREDENTIAL Participant has read access rights. The Link Service Provider Account use case shows how a CREDENTIAL Participant can link his CREDENTIAL Account with another Service Provider which allows him to use the Service Provider’s functionality with his CREDENTIAL identity.

5 Logical Architecture

Based on the use case view presented in Section 4 and the list of generic logical actors, as identified by CREDENTIAL stakeholders, are listed and detailed in this section. First, and as an introduction, in Figure 9 all the logical components and the main relationships are presented.

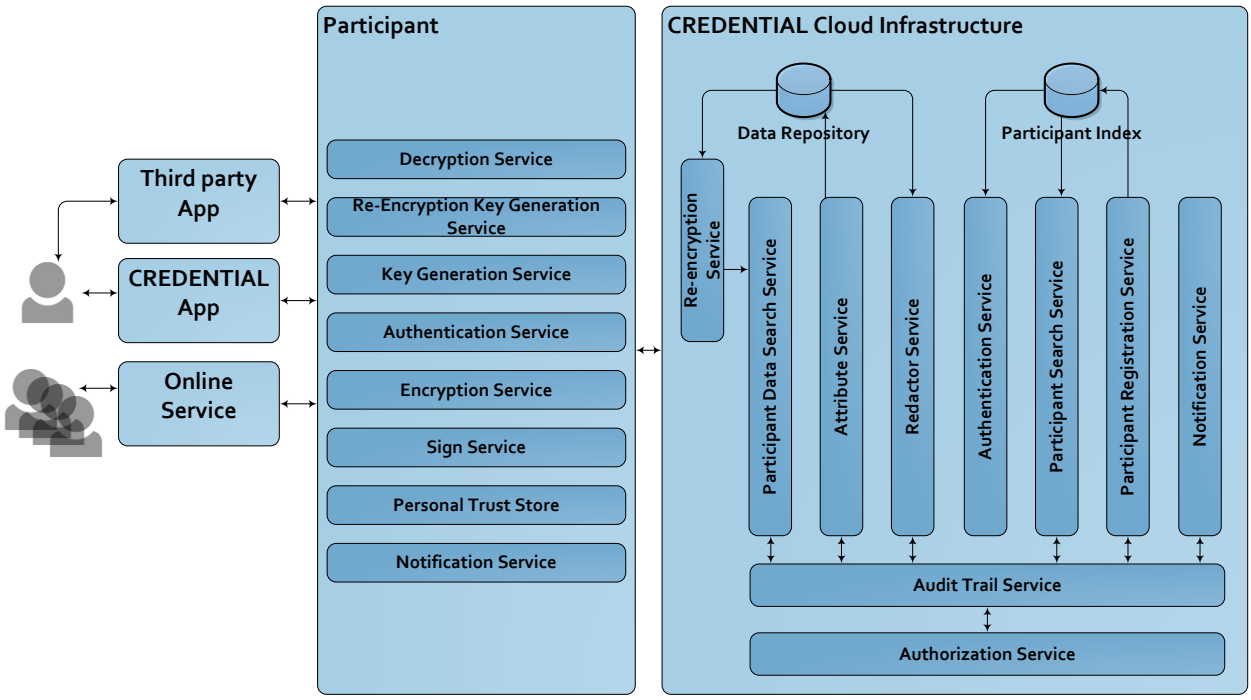


Figure 9: CREDENTIAL Logical Architecture

Next, each of the logical components is described according to the functionalities expressed in the use cases collected in D2.1 and summarized in Section 4. The template used to describe each of the components includes a title, a description and a list of high-level functions, with its own description and parameters characterization. Finally, it also includes a list of constraints and observations for the component as well as its relationship with specific use cases (use cases described in D2.1). An initial classification has been made in order to group related services:

- Cryptographic Services: comprise all services that are related to the management of cryptographic material and its usage to protect data;
- Data Management Services: includes everything dealing with the management of the data and the policies to access it;
- Account and Identity Management Services: deals with credentials, accounts and identity and attributes management;
- Auditing and Notification Services: horizontal services that are not specifically related to the management of data, accounts or cryptographic material.

5.1 Cryptographic Services

5.1.1 Re-Encryption Key Generation Service

Re-Encryption Key Generation Service		
This logical component is an integral part of the innovative and secure data sharing mechanism presented by CREDENTIAL. It generates re-encryption keys that will be used by the Wallet or independent re-encryption proxies to transform encrypted data so it can be decrypted by the data receiver. By generating this re-encryption key from a private key and distributing it to an authorized proxy, the participant owning encrypted data enables sharing and thereby provides some form of consent. After this key is generated, user's data can be re-encrypted and shared with the intended participant without further involvement of a user. Besides this strong enforcement mechanism an additional layer of enforcement is provided through sharing policy mechanisms that may, e.g. logically invalid a key.		
Interface	**generateReEncryptionKey**(data provider private key, data receiver public key): reEncryptionKey	
	Description	This function generates a re-encryption key that can be used to transform ciphertext for one participant into ciphertext for another participant. In order to generate this key it is necessary to use the private key that was used to encrypt the original data and the public key of the data receiver.
	Inputs	• data provider private key • data receiver public key
	Output	reEncryptionKey: the key required to transform ciphertext of one participant for another participant
Constraints / Observations		• As this component requires to operate with a plain version of the data provider private key, this component must be "trusted" by the data provider • The reEncryptionKey sensitiveness is somewhere between public and private. While it is not as sensitive as a private key, it can be used in conjunction with the data receiver's key in order to decrypt ciphertext of the data provider.

5.1.2 Personal Trust Store

Personal Trust Store		
The component Personal Trust Store is responsible to provide a secure place for storing the high confidential CREDENTIAL participant's (users or service providers) private key next to the public key. Only authorized participants are permitted to gain access to this private key. While this component key, is essential to clarify that this element will reside on the participant's physical devices (e.g. an Android Trusted Execution Environment (TEE))		
Interface	**getPrivateKey** (accessToken): privateKey	
	Description	This function returns the participant's private key if a valid accessToken is provided.
	Inputs	• accessToken: Information provided by the CREDENTIAL participant to gain access to the private key.
	Output	privateKey: Asymmetric private key usable for encryption, signatures and respectively re-encryption operations.
	getPublicKey (): publicKey	
	Description	This function returns the participant's public key.
	Inputs	
	Output	Public Key: Asymmetric public key usable for decryption, signatures and respectively re-

		encryption operations.
Constraints / Observations		• The private key must be stored in a secure (i.e. encrypted) way in order to prevent access by attackers. • Independent keypairs are necessary for different operations (Signature/Encryption). • It has to be defined if the trust store itself is responsible for generating/encrypting the keys.

5.1.3 Encryption Service

Encryption Service		
In order to ensure the confidentiality of the user's possibly sensitive data in untrusted environments, these attributes are protected by cryptography. CREDENTIAL uses proxy re-encryption instead of traditional public key encryption to support flexible sharing of the participants' ciphertexts. This component performs the encryption process, which requires public key material of the sender. Users may encrypt data for themselves and upload the resulting ciphertext.		
Interface	**encrypt** (participant's public key, plain data): ciphertext	
	Description	The algorithm offered by this interface encrypts the provided plain data for a participant, who is specified by the given public key. As proxy re-encryption is used, only this participant is able to decrypt the resulting ciphertext or to grant decryption right to another entity through the issuance of re-encryption keys
	Inputs	• participant's public key: The public key of the participant for which the plain data should be encrypted. Only this recipient is able to decrypt the ciphertext or to delegate decryption rights. • plain data: Plaintext provided by the user herself or a third party, which potentially includes sensitive information.
	Output	ciphertext: Data encrypted for the owner of the involved public key.
Constraints / Observations		• As public key material is required, these keys have to be available. • Since only public key material is involved, this component may be operated wherever the plain data's sensitivity allows. • Only the participant designated by the used public key is able to decrypt the ciphertext or to delegate decryption rights.

5.1.4 Decryption Service

Decryption Service		
In order to ensure the confidentiality of the user's data the CREDENTIAL Wallet encrypts all data. All encrypted data have to be decrypted as well at a specific point. That's why this component is one of the fundamental parts of the CREDENTIAL Wallet. This component performs a service which decrypts a given ciphertext utilizing the related private key. A possible scenario is that the user wants to see his data and therefore the decrypt service has to be performed. A second scenario is when re-encrypted data is received from the Wallet, in this case, the user must also use his private key to decrypt received data.		
Interface	**decrypt** (private key of participant A, ciphertext for A): plain data	
	Description	The decrypt function is utilized to decrypt a given ciphertext using a given private key related to a specific participant. The resulting plain text can be read by the decrypting entity such as the user.
	Inputs	• private key of participant A: This private key corresponds to a public key that was used in two scenarios. Firstly, the public key was used to encrypt the ciphertext. Secondly, the public key was used to generate a re-encryption key, and the ciphertext was re-encrypted with that key.

		• ciphertext for A: Data encrypted for the owner (A) of the involved private key.
	Output	plain data: The decrypted content of the provided ciphertext.
Constraints / Observations		• As this component requires access to a participant's private key, it can only be deployed in an environment that enjoys that participant's trust.

5.1.5 Re-Encryption Service

Re-Encryption Service		
The CREDENTIAL Wallet offers novel cryptographic mechanism to ensure user's data confidentiality. One mechanism is the so-called re-encryption which is utilized when sharing data to keep up the data confidentiality. The re-encryption mechanism performs a decryption of a ciphertext together with a new encryption using the re-encryption key. The advantage of this re-encryption mechanism is that the decryption party is not able to see the plain text but is able to re-encrypt the ciphertext. This ciphertext which is encrypted for participant A is re-encrypted for participant B. This mechanism is used for sharing data without sharing the key to decrypt the data.		
Interface	**re-encrypt** (e-encryption key from A to B, ciphertext for participant A): ciphertext for participant B	
	Description	This method takes a re-encryption key together with a ciphertext and re-encrypts it. The re-encryption key is generated using the related re-encryption key generation service. This re-encryption key is used together with the ciphertext, which is encrypted for participant A, and re-encrypts the ciphertext so that participant B is now able to decrypt it.
	Inputs	• re-encryption key from A to B: A re-encryption key generated from A's private key and B's public key. With this key A (delegator) delegates her decryption rights to B (delegate). • ciphertext for participant A: Data that was encrypted for A.
	Output	ciphertext for participant B: If and only if a re-encryption key from A to B was used, the resulting valid ciphertext can be decrypted with B's private key.
Constraints / Observations		• As the re-encryption key is not public information, this component has to be operated at a proxy that is trusted to operate correctly. However, as this proxy neither learn anything about the underlying plaintexts of the processed ciphertexts nor sees private key material, the proxy may be curious to inspect intermediate steps without violating the confidentiality of the handled data.

5.1.6 Redactor Service

CREDENTIAL Redactor Service		
A service which is able to redact fields in a "document" but maintains the validity of the document's signature using malleable signature techniques.		
Interface	**redact** (data, redactable-signature, parts to redact): redacted-data	
	Description	The CREDENTIAL Wallet offers the so-called redactable signatures. This logical component is utilized to redact predefined parts of a document and after redacting the document's signature will still be valid. This is possible using malleable signature techniques. Utilizing conventional signatures does not allow changing (redact) parts in a document after signing it.
	Inputs	• data: This data represents the given input data where the redact operation should be applied. • parts to redact: The parts which should be redacted in the given document are described by this parameter.
	Output	redacted-data: The output of the redact operation is a valid signed redacted document.
Constraints / Observations		• The signature can be different after redacting. This depends on the malleable signature technique utilized.

5.1.7 Key Generation Service

Key Generation Service		
This component generates public/private key-pairs, compatible with the re-encryption schemes chosen for CREDENTIAL. The public key will act as the public Wallet account identifier, while the generated private key must be safely stored in the participant's IT systems and will be required to perform many actions within the Wallet, e.g.: • Encrypt data previously to send it to the Wallet • Sign identity and attribute assertions • Sign authorization requests and responses The cryptographic keys are randomly generated without any specific input		
Interface	**generateKeyPair():** public/private key-pair	
	Description	This function generates a public and a private key that serve as a public account identifier, to transform plaintext to ciphertext and/or to perform signing operations.
	Inputs	
	Output	public/private key-pair: A key-pair with a public and a private component
Constraints / Observations	• This component is crucial as it is the one providing the public/private keys which are the anchor of the trust chain	

5.2 Data Management Services

5.2.1 Participant Data Repository

Participant Data Repository		
This is the service where the actual data blobs of a user get stored along with corresponding metadata related to the account that is linked to, related to the application that generates or triggers its storage and other contextual information such as creation date, etc.		
Interface	**store**(data, metadata): blobId, status	
	Description	This function stores the data in the repository, if the blobId is specified in the metadata it may replace it
	Inputs	data: data/file to be stored. metadata: could, e.g., contain file name or other data based on which the CREDENTIAL Participant Data Search Service (note that full text search is not possible by definition).
	Output	blobID, a unique handle/identifier for the stored data status indicating, e.g., successful/non-successful write operations.
	retrieve (blobId**):** data, metadata, status	
	Description	This function retrieves a previously stored blob, accessing it by its unique identifier
	Inputs	blobId: A handle to the file to be retrieved
	Output	The requested file/data. The stored metadata. A status indicating whether file was found, could not be read, invalid access rights, file size, etc.
	delete (blobId**):** status	
	Description	This function deletes a previously stored blob, referencing it by its unique identifier
	Inputs	blobId: A handle to the file to be deleted
	Output	Status indicating success/failure/missing rights/non-existing file.

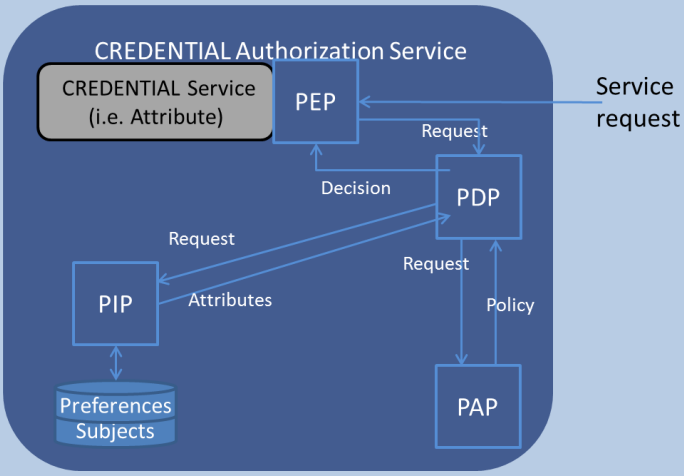
Constraints / Observations	• It should be guaranteed that the metadata does not yield a privacy problem.

5.2.2 Participant Data Search Service

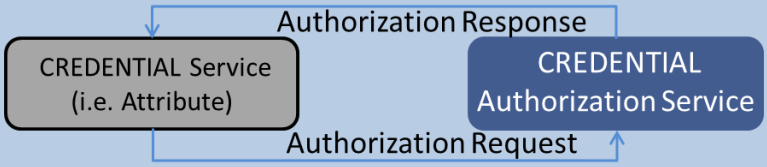
Participant Data Search Service		
This is the service that provides participants to find or access data by specifying some metadata (date, keywords, etc.)		
Interface	**search**(searchParameters): searchResult throws SearchError	
	Description	This logical interface is used in different CREDENTIAL use cases to search for information stored within the CREDENTIAL Participant Data Repository
	Inputs	searchParameters: Defines a value or a range of values for (multiple) meta-information that were assigned to CREDENTIAL data and that should be searched for e.g. data that refers to a concrete participant, data that was created within the last 5 days, data with the type of "PIN"...
	Output	searchResult: Contains pointers to information contained within the CREDENTIAL Data Repository. The pointers correspond to blobIDs and can be used to retrieve the data
Constraints / Observations	• Set of Meta data must be defined in a way that, on the one side is usable and on the other does not threaten data privacy by containing too many details. • Data Search Service and Directory are closely coupled maybe even integrated as search operation will always use some meta-information of the stored data to discover it. If meta information might be stored in an independent registry the coupling might become looser but additional operations to register the meta-information will have to be integrated. • It should be discussed if an implementation of this logical interface must be provided within the project or arbitrary services that realise the search functionality might be used e.g. cloud storages/database etc.	

5.2.3 Authorization Service

Authorization Service
The authorization service is responsible for giving or denying permission to requests to services, data or resources in general. There is a close relationship between the Identity Provider and the authorization service, as the authorization policies may be directly linked to specific users (i.e. “user a” wishes to give “user b” full access to all his data). Traditionally, in an authorization component there are different subcomponents which collaborate to fulfil its objective: • Policy Enforcement Point: intercepts requests to services or resources and requires an authorization before letting them proceed. • Policy Decision Point: is responsible to approve or deny authorization requests, taking into account the established policies and the available information • Policy Administration Point: manages the authorization policies. In the context of CREDENTIAL we will talk about delegation and authorization preferences • Policy Information Point: is responsible for collecting all the information that is necessary for the authorization (e.g. what are the resources related to the request, the identity of the requester, who is the owner of the resource and any other relevant information such as time, origin IP address…)



In the context of CREDENTIAL, there are different services which will require the authorization service (Attribute Service, Sign service, etc.). From a logical point of view, the authentication service will receive authorization requests and issue adequate responses:



Interface	**authorize** (authorizationRequest): authorizationResponse	
	Description	This function analyzes service request and answers by granting or denying the operation following the applicable policies/preferences
	Inputs	authorizationRequest: information related to the original service request (i.e. service called, operations and affected resources, related parameters, security tokens…)
	Output	authorizationResponse: if the authorizationRequest is authorized or not
Constraints / Observations	• In CREDENTIAL, the PIP may interact with other services and components in order to retrieve all relevant information (e.g. with Identity Providers or with the Attribute Service) • The authorization policies will be mostly managed by participants and related to user preferences which decide who can access what (i.e. read e-Health data) • Some of these authorization policies/preferences will have a re-encryption key associated (i.e. to re-encrypt owner's data so it can be decrypted by the data receiver)	

5.3 Account and Identity Management Services

5.3.1 Participant Registration Service

Participant Registration Service	
This component is responsible for registering a new, or deleting an existing account of the CREDENTIAL Wallet. Upon de-registration, also all data related to the specific user should be deleted from the Wallet.	
Inte	**register**(username, aux): status
	Description	This function creates a new account in the CREDENTIAL Wallet

	Inputs	username: a unique identifier for the account to be created aux: any other data aux which is required for a successful registration. This could include, e.g., authenticated identity data in case of legal applications.
	Output	status: status specifying the result of the operation, e.g., successful/not successful registration, missing information, username already exists in the system, etc.
	deregister (username)	
	Description	This function deletes an existing account and all its related data (authorizations, stored data…)
	Inputs	username: a unique identifier for the account to be deleted
	Output	status: status specifying the result of the operation, e.g., successful/not successful registration , user not found, invalid rights, etc.
Constraints / Observations		• Deleting an account is a critical operation which requires a clear understanding by the user what it is about to do and must only be executable after strong authentication. • Error messages (like "user already exists") could be a privacy problem.

5.3.2 Participant Index

Participant Index		
This index contains identifiable information about users. It is written to by the CREDENTIAL Participant Registration Service, and is queried by the CREDENTIAL Participant Search Service.		
Interface	**write**(username, userdata): status	
	Description	This logical interface is used to store users' "searchable" data
	Inputs	Username: The unique identity of the user for which information is to be written. Userdata: The actual identifiable data (e.g., email address, social security number, etc.) to be written
	Output	Status indicating success/failure/...
	deleteUser (username): status	
	Description	This logical interface is used to delete all users' "searchable" data
	Inputs	Username: The unique identifier oft the user for who we want to remove all data
	Output	Status indicating success/Failure/insufficient rights/...
	Search (pattern) [(username, userdata)]	
	Description	This interface is useful to find users matching some pattern
	Inputs	A pattern which the index should be searched for.
	Output	An array containing all pairs of usernames and userdata that match the given pattern.
Constraints / Observations		• It needs to be sure that the search interface does not pose a privacy problem. E.g., in the e-Health pilot, the pure fact that I can find a person's public key in a diabetes service might reveal that the participant suffers from diabetics---an information which should be considered private. • If the CREDENTIAL Participant Index accesses the SAME data as the Registration Service, all but the Search Interface might be redundant.

5.3.3 Participant Search Service

Participant Search Service

This service allows searching for the public key of a participant, which is needed for generating re-encryption keys or defining data sharing policies. The key might, e.g., be found given the user's email-address as input.		
Interface	**SearchUser** (attributes): public_key, status	
	Description	This logical interface is used to search over users' "searchable" data
	Inputs	The user can be found given some unique identifier (social security number, email address, random uid, etc.) as an input. The precise type of this uid might depend on the concrete use case
	Output	The user's public key. A status indicating success/user-not-found/.

5.3.4 Identity Provider

Identity Provider		
In the context of CREDENTIAL, an Identity Provider (IDP) is a component which is responsible for creating, maintaining and managing identity information of data subjects (citizens or organizations). It provides data subjects authentication to other stakeholders (i.e. online services). How the IDP interacts with the data subject in order to create manages or maintain its data is out of CREDENTIAL's scope. The main purpose of the IDPs in CREDENTIAL's ecosystem is to provide identity information to other actors. In order to make CREDENTIAL a truly innovative system, IDPs main features could be improved in order to provide all stakeholders a privacy-enhanced experience: • Identifiers provided to each service should be different to avoid tracking users among services; • Personal data should be provided by the IDP in a re-encryptable form that can be re-encrypted so the only the data receiver can decrypt it; • Issued assertions should be self-contained, meaning that, assertion receivers should not require (or require minimum) additional interaction to verify the assertion. The Identity or attribute assertions issued by CREDENTIAL Identity Providers can be stored in the Wallet (for later forwarding) or directly forwarded to other CREDENTIAL participants. Identity Providers will not provide identity assertions to any requester unless the data subject consents. This means that the data subject must authenticate towards the IDP and then consent to provide the identity (explicit list of attributes) to the data receiver. This authentication process is of key importance as it is part of the foundation of the trust that a data receiver can deposit on the data subject (weak authentication process implies low level of trust on the received identity).		
Interface	**authenticationRequest**(requestedAttributes): identityAssertion	
	Description	This function returns an identity assertion, containing the requested attributes, after the data subject authenticates towards the IDP.
	Inputs	requestedAttributes
	Output	identityAssertion: an assertion containing identity information and the authentication mechanism that the user used to authenticate towards the IDP
	userAuthentication()	
	Description	This function interacts with the data subject, in order to determine that it is the truly owner of one identity
	Inputs	
	Output	The output of this process is that the IDP has, with a certain level of assurance, that the referred data subject is the one interacting with the IDP and that holds the necessary credentials to prove it.
Constraints / Observations	• "Traditional" Identity Providers and Identity protocols are not "directly" compatible with Re-Encryption technology. • There are several authentication algorithms and techniques with different pros and	

	cons. Each of them grant different levels of assurance

5.3.5 Authentication Service

CREDENTIAL Authentication Service		
The CREDENTIAL Authentication Service authenticates participants against the CREDENTIAL Wallet. It consumes credentials provided by the participant, verifies them and in a successful case issues an Identity Assertion with which a participant is able to use CREDENTIAL services.		
Interface	**getChallenge():** Challenge	
	Description	This logical interface is used by participants to retrieve a server challenge that can be used for registration and authentication purposes
	Inputs	
	Output	Challenge: a randomized string
	authenticate(credentials): IdentityAssertion	
	Description	This logical interface is used by participant to authenticate towards the Wallet. The credentials may be directly verifiable by CREDENTIAL or could require a verification with the identity assertion issuer.
	Inputs	credentials: a set of credentials such as signed challenges, identity assertions from accepted identity providers, etc.
	Output	IdentityAssertion: an identity assertion, issued by CREDENTIAL, which can be used by the participant to access CREDENTIAL services or to other service providers
	Errors	AuthNError - e.g. Not valid credentials, expired credentials....

5.4 Auditing and Notification Services

5.4.1 Audit Trail Service

Audit Trail Service		
In general, an audit trail is a full historic list of all actions that are relevant for a certain service or resource. This service/component will be responsible to keep reference and allow querying all the relevant actions within CREDENTIAL i.e. whenever participants different from the owner access and/or modifies some data. The objective of this service is twofold: • On one hand, it works as an accountability and transparency tool, making the owners aware of who and when is accessing its data, so they can detect unexpected access and modify their preferences accordingly. • On the other hand, it works as a forensic tool in the eventual case that the system is subject to an attack or to a malfunction. Audit trail services have two major approaches: • Centralized: audit logs from different components are collected, normalized, aggregated and stored in a central component • Decentralized: audit logs are kept in each component. Whenever there is a log search request, this is distributed to the different logging components. Then the responses are normalized and aggregated, providing a unified response.		
Inte	**logEvent** (eventInformation)	
	Description	This function stores new events in the Audit Trail Service

	Inputs	eventInformation: information related to the event. I.e. related component, time, origin IP, action, related participant…
	Output	
	queryEvents(query): events	
	Description	This function queries the audit logs and returns the events that match the query (i.e. all login events related to the actual user)
	Inputs	Query: similar to SQL queries, a list of fields and associated operations and values (>, <, =…)
	Output	List of relevant events
Constraints / Observations		• The events can be logged individually or in batch. Pull strategies where the audit trail component asks for new events should be studied, as opposite to push strategies where components proactively inform regarding new events. • Audit trails can reveal sensitive information, it has to be carefully studied which data is logged, how long and who can access it.

5.4.2 Notification Service

Notification Service		
The CREDENTIAL Notification Service is responsible for storing participants' notification preferences and for notifying participants whenever there is an event that matches such preferences.		
Interfaces	**getPreferences(**Account**):** List<NotificationPreferences>	
	Description	This logical interface is used by participants to retrieve their actual notification preferences
	Inputs	Account: The CREDENTIAL account to which the preferences are related to
	Output	List<NotificationPreferences>: A list of NotificationPreferences, which include the affected data and the type of operation for those cases when the participant should get notified
	addPreference(Account, NotificationPreference**)**	
	Description	This logical interface is used by participant to authenticate towards the Wallet. The credentials may be directly verifiable by CREDENTIAL or could require a verification with the identity assertion issuer.
	Inputs	• Account: The CREDENTIAL account to which the preferences should be related to • NotificationPreferences: the NotificationPreferences to add
	Output	
	deletePreference(Account, notificationPreferenceId**)**	
	Description	This logical interface is used by participant to delete a notification preference by referencing by its id
	Inputs	• Account: The CREDENTIAL account to which the preferences are related to • notificationPreferenceId the id of the notification to delete
	Output	
	notify(Event)	
	Description	This logical interface is used by other components that want to make aware participants of specific events. The component will decide, based on applicable notification preferences if the user must be notified
	Inputs	• Event: An event occurred in CREDENTIAL (someone accesses some data, someone

		tries to access your data bus is not authorized...)
	Output	

6 Development Architecture

In order to follow the methodology detailed in Section 2.4 this section contains a list of the main SW components that will be part of CREDENTIAL systems. And includes description provides details on their generic functionality, technology, dependencies, license, APIs and interfaces.

Before presenting CREDENTIAL's general data architecture (Section 6.1) and the list of individual components, two different views of the general design of CREDENTIAL are introduced.

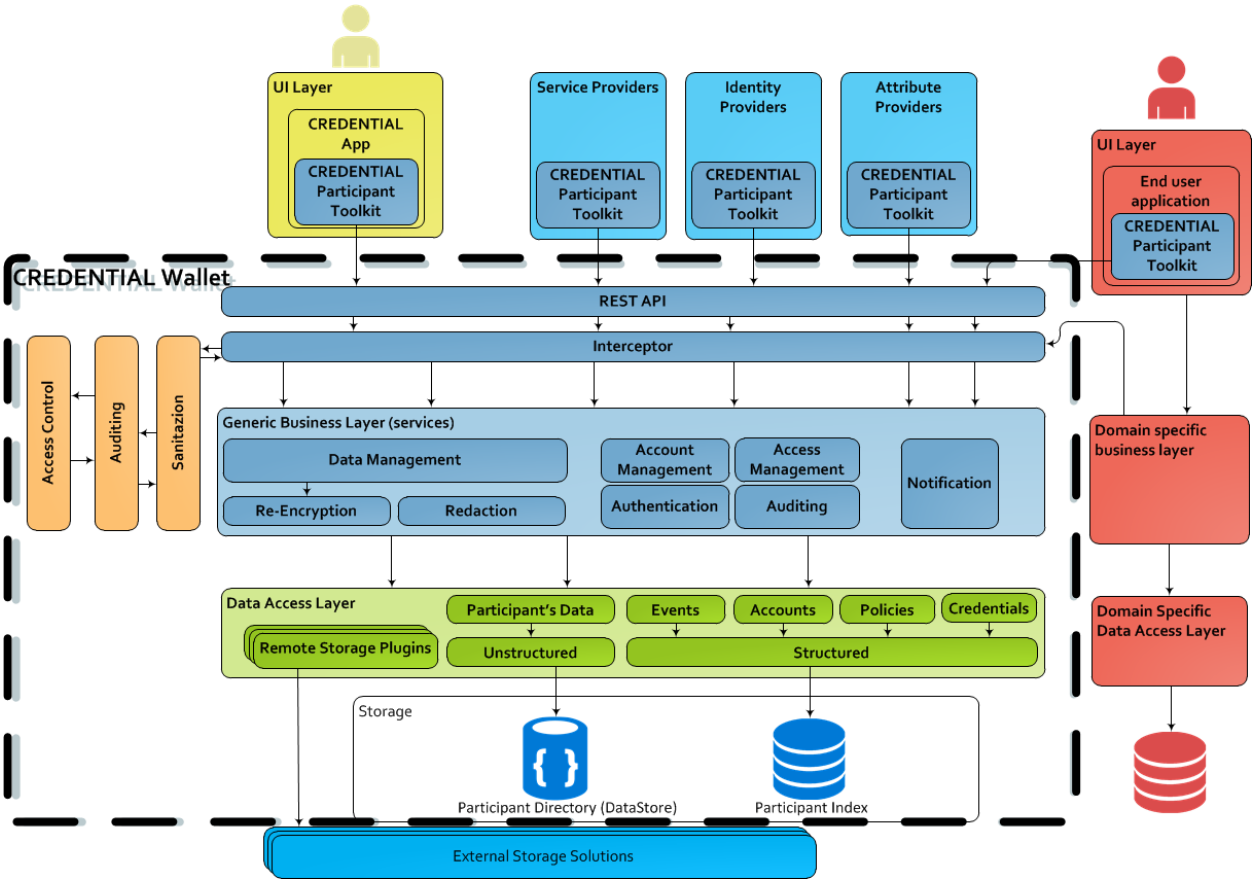


Figure 10: General Layered Architecture

Figure 10 shows a three layers architecture. The first layer is related to the interface provided to all relevant stakeholders. While CREDENTIAL consortium will present a CREDENTIAL specific application with its own UI, a Participant Toolkit will be provided for its integration in other participant's systems and interfaces. E.g. an e-Government service provider may provide their own service for citizens but be connected to CREDENTIAL systems in order to request access to citizens' data. The government would use the toolkit to decrypt attributes coming from the Wallet. The CREDENTIAL Mobile Application is not a core part of the design of the architecture and hence the detailed description is not fully described in the main sections of the deliverable. However, notions on how CREDENTIAL Mobile App should be developed are hinted in Appendix B – CREDENTIAL Generic Mobile Application.

At a second level, the business layer can be divided in two main groups. On the one hand, horizontal functionality such as access control, auditing and data sanitization, that are applicable to all other services

(e.g. account management, notification, etc.). Horizontal functionality will be applied to all requests by using interception mechanisms specific for the selected technology (e.g. JAVA's filters). On the other hand, specific services dealing with specific functionalities will rely on a data access layer that will be used to store personal data and operative data. Personal data will use some No-SQL technology so applications connected to CREDENTIAL may specify their own schema for data or metadata. Data related to the operation of the platform and which can be defined beforehand (e.g. policies, accounts, etc.) will be stored in structured databases.

It is important to notice that the CREDENTIAL architecture is flexible enough to allow many levels and approaches of integration. I.e. Participants can just provide an application and use CREDENTIAL as a backend or they can also have a more complex integration having their own full stack (mobile app, business layer and data layer) and connecting the business layer to the CREDENTIAL Wallet.

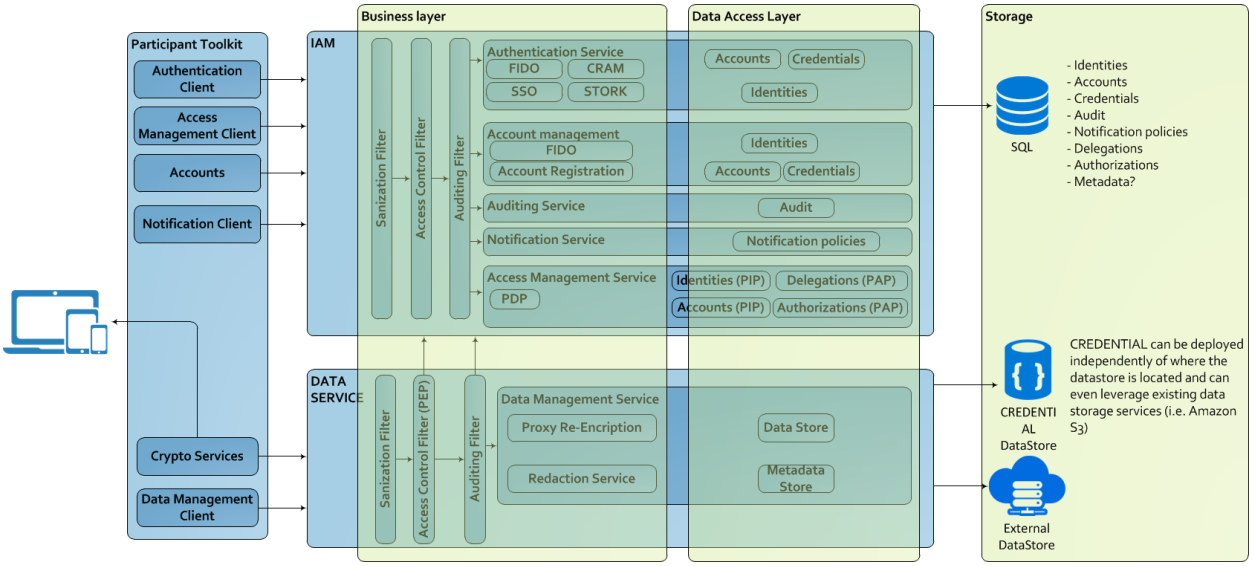


Figure 11: The CREDENTIAL Wallet Sub Components

Figure 11 show a more detailed view of some of the components. E.g. The authentication service must include modules dealing with Single Sign On functionality towards service providers, integration with FIDO standard, STORK/eIDAS infrastructure and a basic Challenge Response Authentication Mechanism (CRAM).

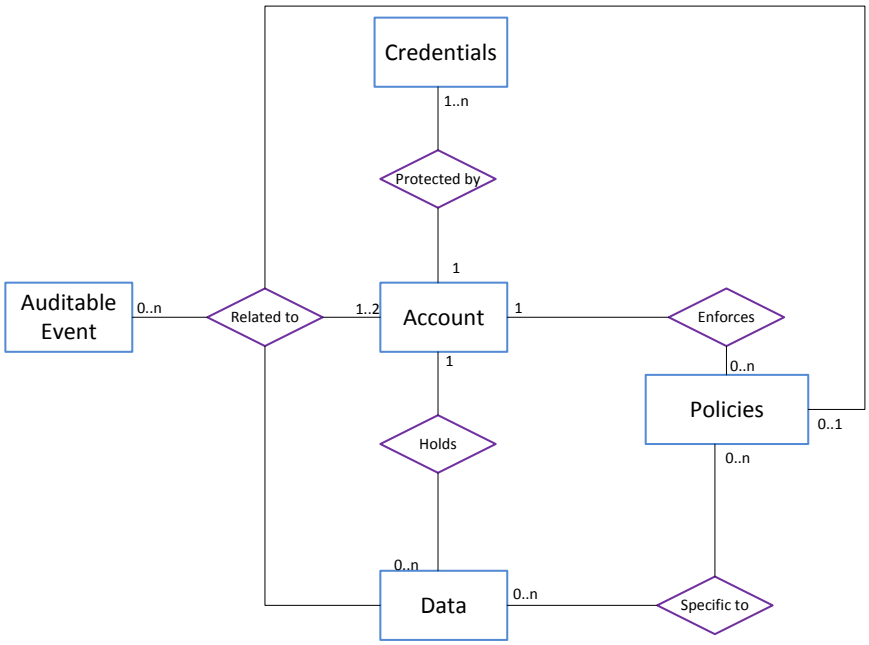
6.1 Data Architecture

The data architecture of the CREDENTIAL Wallet presents a number of challenges as it has to be flexible enough to allow the creation of services on top of the selected data structure, to support the re-encryption and redactable signatures technologies and to disclose minimum information in order to maximize privacy preservations.

CREDENTIAL's data architecture, as the development architecture, can be divided in two high-level components. On the one hand, the data structure related to the IAM operation, comprised of accounts, credentials, authorization and delegation policies, and auditing data. On the other hand, the personal data

store, where only data, provided by participants, and not related to the functionality of the Wallet itself, is stored.

For the IAM related data, it is proposed to use a standard SQL database with the following high-level data model:



For the personal data store, and in order to cope with the flexibility challenges, the data structure has to be very flexible to accommodate generic and domain specific scenarios requirements. Hence, a NoSQL document based database where the document properties can vary from one document to another provides a very good starting point (e.g. MongoDB[2]).

In the following paragraphs it is proposed NoSQL data structure that could be used to address the requirements of any of CREDENTIAL's Scenarios regarding the storage of, e.g. patients' blood measures. The presented data in the following block is shown in plain and it is just a draft to demonstrate the main ideas. Final document specification will be provided in further tasks. The encryption scheme to be applied to such structure is further described in Table 3.

```
{
    'document-id': 'f28e1d5d-237e-4cb4-9d09-83b41eff19c1',
    'general-metadata': {
        'account-id': '577b2c00-8215-44ac-bf3d-ed0dac31fffc',
        'document-name': '/medical/bloodpresure/lastWeek.data',
        'creation-date': '18-02-2016',
        'modification-date': '19-02-2016',
        'application-id': '06ae0686-2fe5-4db9-b892-af502bdf4702',
        'embedded': true,
```

[2] https://www.mongodb.com

```
    },
    'application-metadata': {
        'app-property-1': 'value1',
        'app-property-2': 'value2',
        'app-property-3': 'value3',
    },
    'data': [{
        'key': 'key1',
        'value': 'value1'
    },
    {
        'key': 'key2',
        'value': 'value2'
    },
    {
        'key': 'key3',
        'value': 'value3'
    }]
}
```

Figure 12: Example of a Generic NoSQL Data Structure Compatible with Pilot Requirements

Table 2 lists and describes the different elements of the data structure.

Property	Description and Rationale
Account-id	This element links documents/data to accounts. An example of an account ID could be a public key. It can also be a random unique number generated by the Wallet during the account creation process and linked to a public key.
Document-id	Unique document identifier. It can be some random number generated by the Wallet or some kind of hash generated in participant's system. Uniqueness must be guaranteed as this is the primary key to access the document.
Document-name	Optional, in some cases it would be useful for participants organizing their Wallet. On data automatically generated by i.e. the health application is irrelevant. One option would be to treat this as application-metadata. In some cases it could leak some personal data.
Creation-date	Document creation date. This info may be useful for sorting documents in participants' applications and/or for the signature of the values/data
Modification-date	Document modification date. This info may be useful for sorting documents in end-user application and/or for the signature of the values/data
Application-id	A unique participant application id. This field may be useful in participants' applications. I.e. ask to the Wallet for all files created from the application. Alternatively use also a end-user application-specific index, the application holds a list of all documents it has created
Embedded	This field distinguish NoSQL documents which will have the actual data stored within the same document and those which are stored in an remote or separate object store/filesystem
Application-metadata	This field accommodates application-specific metadata that may be useful to understand the data itself. This will vary from one document to another.
Data	The data itself

Table 2: Data Structure properties

After presenting the basic data structure, the next step is to deal with its encryption. While the minimum requirements must guarantee the encryption of the data itself, the document structure and metadata may

reveal information of the data and its owner. Hence it has to be agreed which and how metadata has to be encrypted. Table 3 presents proposition on what metadata structures are to be encrypted:

Property	Description and Rationale
Account-id	Not encrypted.
Document-id	Not encrypted
Document-name	Encrypted. Currently researching suitable ways to search on encrypted data to minimize the metadata leakage to the Wallet while still retaining search functionality.
Creation-date	Encrypted. Currently researching suitable ways to search on encrypted data to minimize the metadata leakage to the Wallet while still retaining search functionality
Modification-date	Encrypted. Currently researching suitable ways to search on encrypted data to minimize the metadata leakage to the Wallet while still retaining search functionality
Application-id	Encrypted. Currently researching suitable ways to search on encrypted data to minimize the metadata leakage to the Wallet while still retaining search functionality
Application-metadata	Encrypted. Three options. Encrypt whole application-specific metadata structure which would only allow querying it in the client-side, encrypt property-value or encrypt property-name and property-value. Second and third options would enable exact match filters. Second and third option would reveal metadata structure which could lead to identifying unique structures and leading to matching accounts with end-user applications. Second and third options reveals more info regarding the data but allows more complex queries (filters by date, etc..). Second option directly may reveal the kind of app the user is using (e.g. measure-device-id)
Embedded	This field distinguishes NoSQL documents which will have the actual data stored within the same document and those which are stored in an object store/filesystem. If non-encrypted it could reveal some info regarding the size of data (larger documents are usually not embedded)
Value	Encrypted

Table 3: Encryption of data structure

Next, each of the components identified in Figure 10 is detailed. For the sake of clarity, the components have been grouped according to where will be deployed. The presence in the architecture of each of the development components is linked to the logical services described in Section 5 and the use cases defined in D2.1. The detail of the mapping between use cases and components is presented in Appendix C.

6.2 CREDENTIAL Wallet Services

The CREDENTIAL Wallet Services includes all those services that have to be deployed and provided online to the different stakeholders. These are the core services of the whole CREDENTIAL system and provide functionality related to account, identity, notifications and data management services and other essential services such as auditing.

6.2.1 Authentication Service

The authentication service in the server side is responsible to, in base to some pre-established credentials such as a key-pair, determine if the end-user trying to access the Wallet is the owner of such credentials.

6.2.1.1 Component Description

The authentication service attends authentication requests from any CREDENTIAL participants and is closely related to the account management service (see Section 0). This service has three main objectives:

- Enroll authenticators and corresponding credentials (besides main key pair) and associate them to an account in order to provide authentication alternatives with variable level of assurance, usability, convenience, etc.
- Verify that the credentials that the user provides in order to access the Wallet correspond to the owner of the account created through the account management service.
- Allow other participants to use the Wallet as an authentication provider towards their own services.

The authentication component is built as a single endpoint that supports multiple authentication protocols. The endpoint analyzes the received authentication requests and responses and instantiates, according to the requested URL,the appropriate protocol-specific component. Then, the received message is handled taking into account the actual step of the authentication process. With this model, the concept of authentication requests and responses is very wide as it can comprehend from SAML messages in XML to JWT tokens or FIDO-specific messages. However, for the sake of the design of the component, all protocol-specific messages are considered as AuthenticationRequests or AuthenticationResponses.

As depicted in Figure 13, the two main sub-components are:

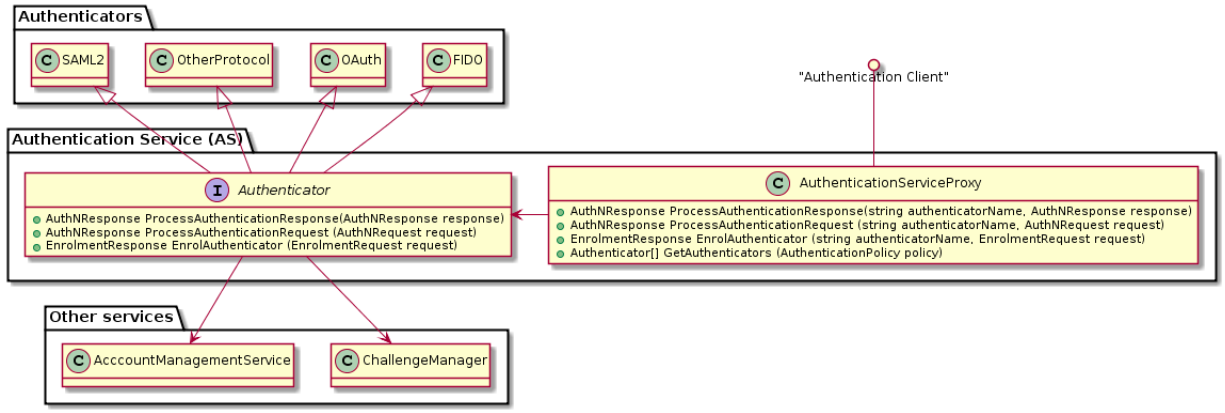


Figure 13: Authentication Service Component

- The authentication service proxy which is responsible for listing available authenticators, for enrolling authenticators with accounts and for forwarding messages to the appropriate authenticators
- The authenticators that implement a common interface that should address common protocol requirements. These authenticators will use CREDENTIAL-specific and other means to perform the authentication. E.g. they may use CREDENTIAL's challenge creation and verification service, they may access existing account information or they may connect to users' devices to perform biometric local authentication (like expected in FIDO protocol). These authenticators will also act as translators/mediators that are able to understand CREDENTIAL accepted protocols (e.g. SAML or OAuth) and to generate protocol-specific requests (e.g. FIDO, STORK-SAML).

The four main functions implemented by this service are:

- GetAuthenticators: allows participants to retrieve a list of available authenticators that match a certain security policy (e.g. biometric authentications, certain level of assurance)
- ProcessAuthenticationRequest: this functionality allows third parties and the CREDENTIAL Authentication client to initiate the authentication process.
- ProcessAuthenticationResponse: this functionality allows third parties and the CREDENTIAL Authentication client to continue with the authentication process.
- EnrolAuthenticator: this functionality allows owners of CREDENTIAL accounts to register as many authenticators as they desire to the accounts they own.

With this approach, the concept of the AuthNResponse is very wide. It can directly hold information regarding the identity of the end user or information to carry out the next authentication step. I.e. it may be a security token (e.g. JWT) with adequate expiration and revocation policies that can be further exchanged to maintain a session between the user and the Wallet or Third party service.

6.2.1.2 Component Justification

This component is related to the logical components:

- CREDENTIAL Identity Provider
- CREDENTIAL Authentication Service

6.2.2 Account Management Service

The account management service is responsible for handling the life cycle of CREDENTIAL accounts.

6.2.2.1 Component Description

The account management service, which allows managing the full life cycle of CREDENTIAL, is built upon three main components:

The service itself, which will map a public REST interface and its URLs to a set of functionalities which have been extracted from the list of use cases outlined in Section 4 (Use case view).

This component has to (at least) implement the following functions.

- **Challenge createAccount(byte[] publicKey):** creates a non-confirmed CREDENTIAL account, makes it persistent and returns a challenge to be answered by the call originator
- **Account confirmCreateAccount(ChallengeResponse challengeResponse):** as part of the creation account process, the account creator must answer the server-generated challenge to assure the ownership of the private key that correspond with the public key that was initially presented
- **Account deleteAccount(Account account):** it deletes the account at a logical level and all data related to the account. The account is not fully deleted in order to allow the multi-account master key mechanism.
- **Challenge recoverAccount(byte[]recoverAccountPublicKey):** CREDENTIAL accounts may have a recovery key associated which will allow access to the account even in the eventual case where the original key is lost. This mechanism will generate a challenge to be answered by the

recovery process initiator to provide a proof of the ownership of the private counterpart of the recovery key.

- **Challenge confirmRecoverAccount(ChallengeResponse challengeResponse):** once the recovery initiator confirms the ownership of the key pair, the recovery process is initiated including: associating a new public key to the account and a re-encryption of all data associated to the account. Whenever an account is recovered, a new set of re-encryption keys must be generated. New keys will be generated for the existing policies managed by the participant while other participants will be requested to re-generate the keys associated to the "old" account.
- **Account[] [] getAccounts (byte[][] publicKeys):** this function enables clients to query accounts by an array of public keys. This is useful when importing a master key, the client may generate a number of derived keys and check if they've been used in the server, to restore links to them in the client application. If the publicKey is related to an account, it returns, in the same position in the array the corresponding UUID. If not, it will return an empty string.
- **Account saveAccount(Account account):** updates the account information and makes it persistent in the data base
- **Account[] searchAccount(String[] key, String[] value):** searches in the attributes associated to the account for matching the key value provided. It will only return full matches. E.g. searchAcccount(["email"], [john.doe@acme.coml);

There will be a distinction between some special account properties that are not updatable by the account owner but by CREDENTIAL system administrators (e.g. a banned attribute indicating if the account is banned or not).

A second subcomponent, the account manager data access object will provide all the data persistence logic that allows separating the business from the data management logic and will only have 3 main functions:

- **Account getAccount(byte[] publicKey):** retrieves a stored account using the byte[] as a key. Due to performance issues or database limitations. It may be the case that key fingerprints are used.
- **Account[] searchAccountByAttributes(String[] keys, String[] values):** search for matching key values among the attributes associated to accounts
- **Account saveAccount(Account account):** persists the account in the storage.

Finally, the third component is related to the challenge-response mechanism and it is responsible for the secure generation of challenges and the verification of the associated responses.

It is important to highlight how there is a distinction between accounts and identities which have a relation of "one to one or none". Accounts without an associated identity will remain anonymous, while accounts meant to be public will have identity (attributes) information which will allow other participants to search for these public accounts. At a UI level it will be necessary to inform the user that, whenever it provides identity info, these immediately will become public and publicly searchable.

Figure 14 shows the component's structure and its relation to other components:

- Data Management Service: has to be invoke for the deletion of all the data related to the account being deleted;

- Access Management Service: upon an account deletion, credentials should be registered;
- Auditing Service: it is essential to securely log the account deletion request and related operations.

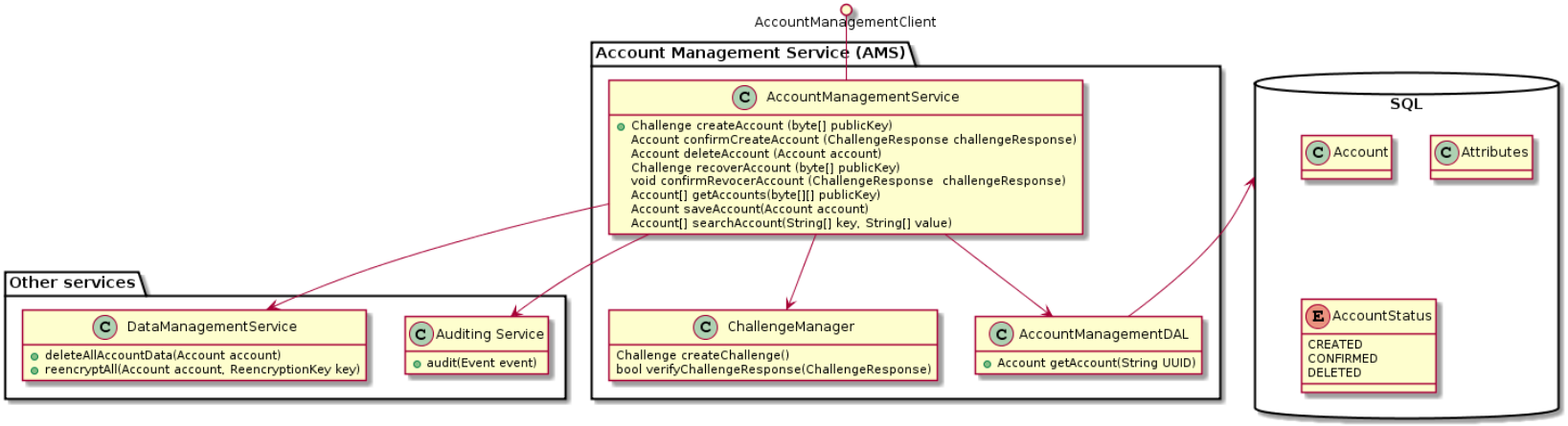


Figure 14: Account Management Service Component

6.2.2.2 Component Justification

The need of an account management service component is tightly linked to the logical components identified as Participant Registration Service (Section 5.3.1) and as Participant Index (Section 5.3.2).

6.2.3 Access Management Service

This service controls the access to the users' data by managing and evaluating requests against user-defined policies.

6.2.3.1 Component Description

The Access Management Service is a core component of the CREDENTIAL Wallet, responsible for controlling who is authorized to access the users' data and to use the Wallet's services. In this section, it is first described the interconnection with related actors, given an overview of the used technologies, explained how these technologies are applied in two approaches, and finally the subcomponents are enumerated.

The Access Management Service mainly communicates with two other top-level components, namely the Access Management Client and the Access Control Filter. Firstly, the client-side of the Access Management consists of libraries that communicate with the server-side to manage policies and to acquire tokens with sufficient power to access the desired resources. Secondly, the Access Control Filter is deployed in front of the Resource Server as well as in front of the Authorization Management Service. This filter receives requests to access resources or services, for which it obtains an authorization decision from the Access Management Service.

Three core technologies are suggested to be used by the Access Management Service: First, OAuth is used to grant a third-party application access to the CREDENTIAL Wallet. Second, User Managed Access (UMA) extends OAuth so that a resource owner is able to securely share his/her data. Moreover, UMA allows the resource owner to protect his/her resources stored on the Data Service. Third, the eXtensible Access Control Markup Language (XACML) is used to describe complex access policies. XACML is mainly used in combination with UMA to cover sophisticated data sharing permission between users.

We apply two different approaches to access control: Firstly, to protect the user's data stored on the Data Management Service, we employ OAuth's User Managed Access (UMA) Profile and policies in the eXtensible Access Control Markup Language (XACML). A user is able to protect and share her resources on the Data Service by defining access policies. Additionally, requesters are able to request access to data by suggesting policies. Secondly, plain OAuth is used to protect the remaining services offered by the CREDENTIAL Wallet. In comparison, the rules to access those services are simple, as only a limited and pre-defined number of actors should be able to use the services.

The subcomponents of the Access Management Service, shown in Figure 15, are listed and detailed below.

The **OAuth Authorization Server** issues access tokens to involved communication partners after ensuring they are authorized to access the APIs. The involved communication partners can be end users but also resource servers and software clients. Since users are authenticated by the Authentication Service,

the issued authentication data carrying sufficient information (SAML Assertion, JWT id token) can be exchanged for access tokens. The OAuth Authorization Server offers the following endpoints:

- **Authorization Endpoint**
- **Token Endpoint**
- **Dynamic Registration**

In contrast to a plain OAuth Authorization Server, the **UMA Authorization Server** makes it possible to perform more fine-grained access control at resource-level. Resource Servers register their resources at the UMA Authorization Server. When a requester tries to access such protected resources, a Requesting Party Token, sufficient to authorize the current request is required. This token is issued by the UMA Authorization Server, after the requester supplied all necessary data and a policy defined by the data owner was satisfied. The UMA Authorization Server can be split into the following subcomponents:

- **Configuration Endpoint**: This endpoint describes the configuration of the UMA Authorization Server, which is necessary to correctly interoperate with the service.
- **Protection API**: This API handles the communication with the resource server.
    - Resource Set Registration Endpoints
    - Permission Registration Endpoint
    - Token Introspection Endpoint
- **Authorization API**: This API handles the communication with the client.
    - RPT (Relying Party Token) Endpoint

The **Policy Engine (XACML)** evaluates incoming requests against policies defined by the data owners. Authorization requests coming from the RPT Endpoint carry an association to the requester as well as to the requested resource. In order for a Policy Decision Point to reach an authorization decision, the incoming data might have to be augmented with additional data obtained from a Policy Information Point. The two main components of the Policy Engine are:

- **Policy Decision Point (PDP)**: Evaluates policies defined by the data owner.
- **Policy Information Point (PIP):** Retrieves additional information required for the decision process from various data sources.

The **Policy Management** component allows users to define policies for their data. The endpoints of this component have to be protected, so that only authorized users, namely the data owners, are able to set and change policies for their data. This protection can be achieved with OAuth. The endpoints to manage the users' policy are mainly:

- **Create, Read, Update, and Delete**

The **Request Permission Service** is used by a requester to request access to files or their parts from a data owner if no suitable policy was found. The access request is sent by the Access Control client to the Request Permission Service providing a suggested access policy. As this suggested policy has to be approved by the resource owner, the policy is forwarded via the Notification Service. Once approved, the owner generates a re-encryption key, which is installed in the "**Re-Encryption Key Store**" along with the

policy at the CREDENTIAL Wallet. Finally, the requesting party is being notified that access to the required data is now possible.

The **Data Layer** handles the storage and retrieval of all required data. APIs for different data types decouple the storage location from the actual code.

- Policies
- UMA Configuration
- Registration data of resource servers
- Registration data of clients
- Permission Tokens (for pending authorization requests)
- Access Tokens and Relying Party Tokens
- Re-Encryption Keys

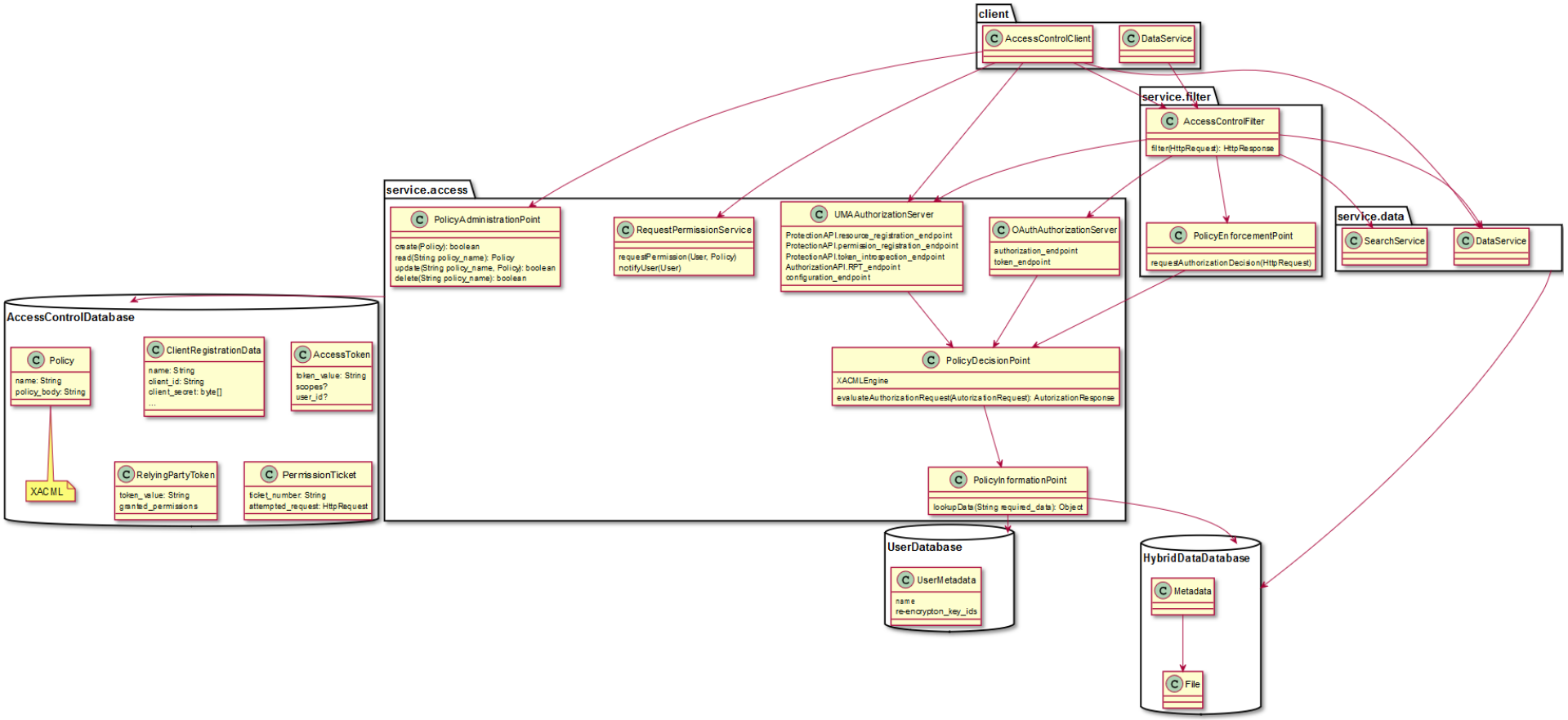


Figure 15: Components of the Access Management Service

6.2.3.2 Component Justification

This component is related to the Authorization Service logical component

The Access Control Service mainly tackles use cases for requesting and granting access rights. Furthermore, this component also provides crucial internal communication with other components deployed in the cloud-based Wallet, in order to support them in reaching authorization decisions.

6.2.4 Data Management Service

This service provides access to the users' data. When another user's data is requested, this service transforms the ciphertext for the data owner into ciphertext for the requester.

6.2.4.1 Component Description

The Data Management Service manages and provides access to the users' data. This component offers two externally reachable services, the data service to access, store, and modify data, and the search service to efficiently find data. Internally these services are built on top of the data layer, which handles the data's persistence without being limited to one storage technology. Also, the resources set registration component notifies the access control service about available resources. Figure 16 provides an overview of the individual subcomponents. The following paragraphs go into details about these subcomponents.

The **Data Service** offers the primary interfaces to access, store, and modify a user's data. In general, we distinguish between the actual payload data and accompanying metadata. The metadata contains for example, the id of the data's owner, information about the structure, algorithms, used key material of the encrypted data, a hierarchy reference, as well as encrypted search keywords. The payload is usually the encrypted document. This data service also performs the re-encryption operation when data is requested by a user which is not the data owner, assuming this requester is authorized. This data service offers the following operations:

- **Upload, Download, Update, and Delete** Data: These operations may not only affect the whole document and its attached metadata but may also be applied to subsets. Based on the description of the encrypted payload's structure, only a desired subset of the whole payload can be retrieved or updated.
- **Retrieve Hierarchy**: Data items are organized as a hierarchy, in order to structure them in a user-friendly way. The metadata of a document specifies where in the hierarchy it should be placed. Metadata for a single document and its children can be retrieved from this component.

In addition, the **Search Service** makes it possible to find specific data items based on a query. This service exposes the following operation:

- **Search (query): the query allows to only extract documents, whose associated metadata satisfies a tree of logical AND/OR statements.**

The **Resource Set Registration component** represents the interface with the UMA Authorization Server of the Access Management Service. Through these interfaces, new resources are registered and removed resources are deregistered at the OAuth Authorization Server.

The **Data Layer** is responsible for the persistence of the user's data. In general, we distinguish between the actual data and accompanying metadata. Depending on the size or application's preferences, these two types of data can be stored together or separate. This layer is designed to abstract the internally used API from the actual storage technologies. Such an abstraction makes it possible to implement storage functionality that makes use of a cloud's preferred database technology.

- **Document-Store**: stores metadata and small, structured payloads or references to a blob-store.
- **Blob-Store**: the blob-store might be more suitable to store payload data with large size.

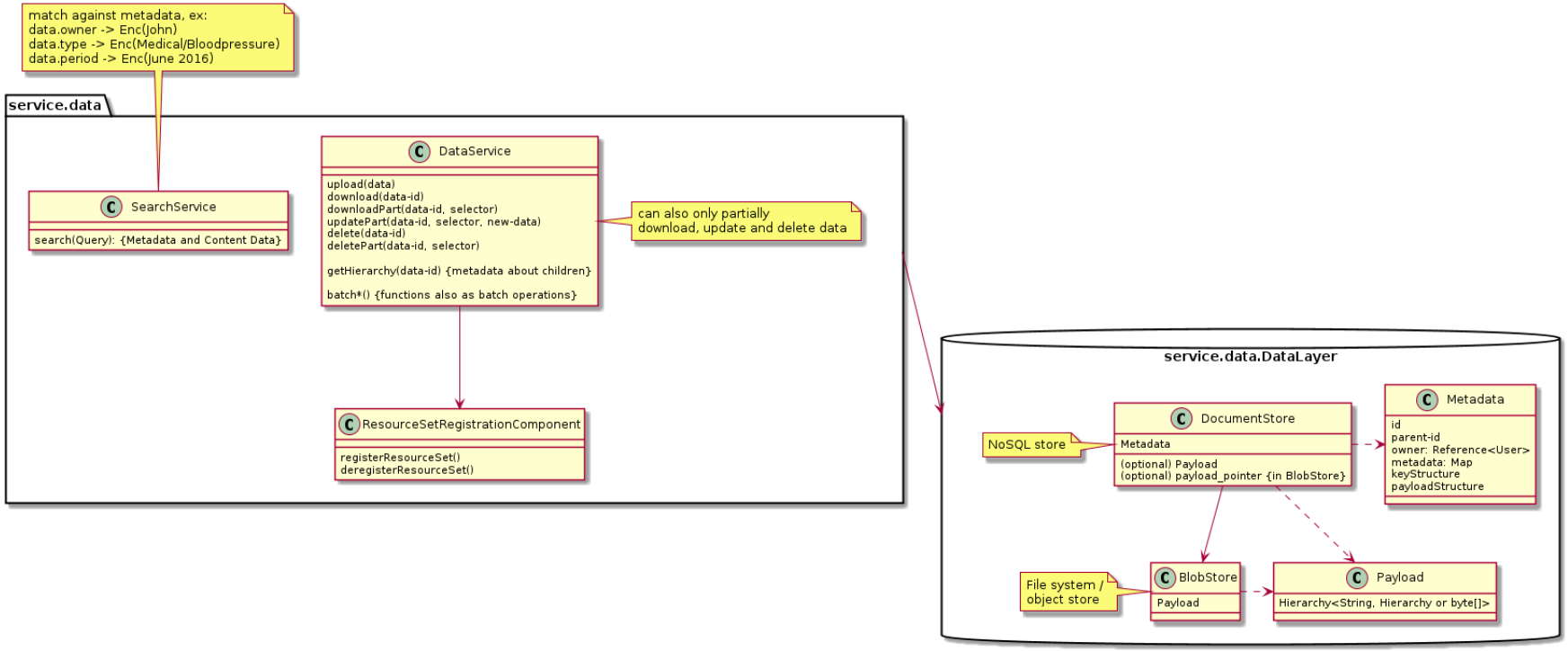


Figure 16: Data Management Service

6.2.4.2 Component Justification

This component is related to the logical components:

- CREDENTIAL Participant Data Repository
- CREDENTIAL Participant Data Search Service

6.2.5 Auditing Service

This component will store, in compliance with current legislation framework and in consonance with privacy requirements, information regarding attempted access and authorizations to access stakeholders' data

6.2.5.1 Component Description

The auditing service, while having a simple design, it is of key importance and helps to fulfil three objectives:

- Compliance with the legal requirements regarding the monitoring of accesses to sensitive data;
- Support security tasks by keeping logs that will help to detect security breaches or for forensic analysis, after such breaches have occurred;
- Transparency principle of the GDPR, empowering users with information regarding who and when has accessed their data raising awareness of the scope of their data sharing

This server-side component is intertwined with the audit filter (see Section 6.4.2) and also with the client-side which are the only other components that will interact with this service. Assuming that all data stored in the Wallet is encrypted, personal data may be inferred just by looking to data access patterns, hence, there is a strong authentication and authorization requirement placed in this component which can be summarized as: "Only owners of data (and to some extent administrators) will be able to access auditing information related to their data". A second requirement will be implemented by this component and is that only components authenticated as part of the Wallet infrastructure will be allowed to add events to the auditing service. The server side of the auditing service is basically made by one component service which stores and queries a SQL database (green packages in Figure 17), following the previously mentioned security constraints.

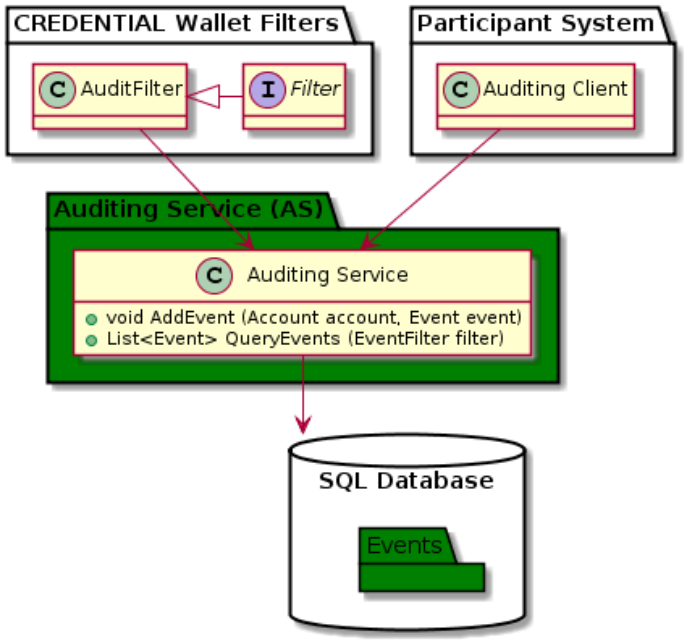


Figure 17: Auditing Service Structure

The main two functions implemented by this service are

- AddEvent: stores an event which is always related to an account. Examples of what kind of events can be added and stored are: accesses to data by different participants, participant's authentication events, successfully authorizations or denials of access. Events will also hold information regarding the date and other relevant information that will be agreed to find the best tradeoff for security, privacy and privacy awareness.
- QueryEvents: all participants can query the events related for any data they own. The event filter, object will allow participants establish some filters for the query such as an identifier of the data, date/time minimum and maximum dates, participants' identifiers, types of events, etc.

The exact fields or data to be stored for each event will have to be determined in collaboration with the three CREDENTIAL scenarios. A potential improvement of this component would involve having two separate logging databases. The first would store, without any encryption measure, only the minimal data required for legal purposes and would only be available to the Wallet operator, while the second would store more interesting and privacy-sensitive data, but encrypted so that only the notification recipient can read it.

6.2.5.2 Component Justification

The need of an auditing service component is tightly linked to the logical components identified in the Audit Trail Service and to its associated logical use cases.

6.2.6 Notification Service

The Notification Service is responsible for recognizing events that happens on the CREDENTIAL Wallet and notify users according to their preferences. Users can customize various notification settings and get notifications on their devices.

6.2.6.1 Component Description

The Notification Service manages the event notification mechanism of the CREDENTIAL Wallet. It fulfills mainly three roles:

- Manage account specific notification mechanism properties
- Register events that occur within the CREDENTIAL Wallet
- Send push notifications to the participant's devices about certain events they want to be informed

The server-side component offers a REST-interface for client-side components which offer the functionality to change the account notification preferences for an individual and to register events which may result in further notifications. By using a REST-interface the server-side component is flexible enough to recognize events within the CREDENTIAL Wallet as well from components that are not part of the CREDENTIAL Wallet. Since account related can be changed by this service it has to collaborate with the Authentication Service, the Access Control Management Service and the Audit Trail Service.

The main task of the Notification Service is to process incoming Events and match them against the notification preferences of the users. Thus, the solution contains a Notification Dispatcher component which is responsible to continuously check the availability of newly added Events to the Event Queue.

A user subscribes to certain type of events by creating a Notification Preferences object. It is linked to his account and describes for what type of events and in which conditions the user should receive a notification. The Notification System offers the Notification Management interface with the three basic operations:

- getPreferences – Returns the list of Notification Preferences that are currently active for a given account.
- addPreference – Adds a Notification Preference to the given account
- deletePreference – Deletes a given preference for a given account

Events are published to the Notification Service through the notify interface and added to the Event Queue. The backbone of the Notification System is the Notification Dispatcher. It runs continuously and checks the Event Queue for new and unprocessed events. The events are checked against the stored Notification Preferences. If a Notification Preference matches an event the Notification Dispatcher creates a new notification in the Notification Database. It contains at least the recipients URI and a document describing the notification. The notifications are processed by a different process of the Notification Dispatcher which tries to deliver the notifications to the desired recipient. It does this by using a Notification Broker component. The Notification Broker is responsible for receiving messages from the Notification Service components or Notification Client components, identifying recipients, and transmitting the messages to the recipients. In addition, it keeps track of all the connected devices. It can be realized by standards like MQTT [14] or solutions like GCM [15] or Pushy [16]. Special privacy and

security considerations have to be met in the latter solutions because notifications are delegated to external services. Therefore, the MQTT solution is the preferred way to deliver notifications to end user devices since the Notification Broker can be executed on the CREDENTIAL Wallet provider side. Figure 18 describes the components of the Notification System.

Requests to Notification Service and to the Notification Management sub components, as all requests towards the CREDENTIAL Wallet services, filtered through data sanitization and authorization filters that ensure that only "clean" and authorized requests are received. Requests to the methods offered by these two components may change account settings and allow registering events which may result in notifications send to the users.

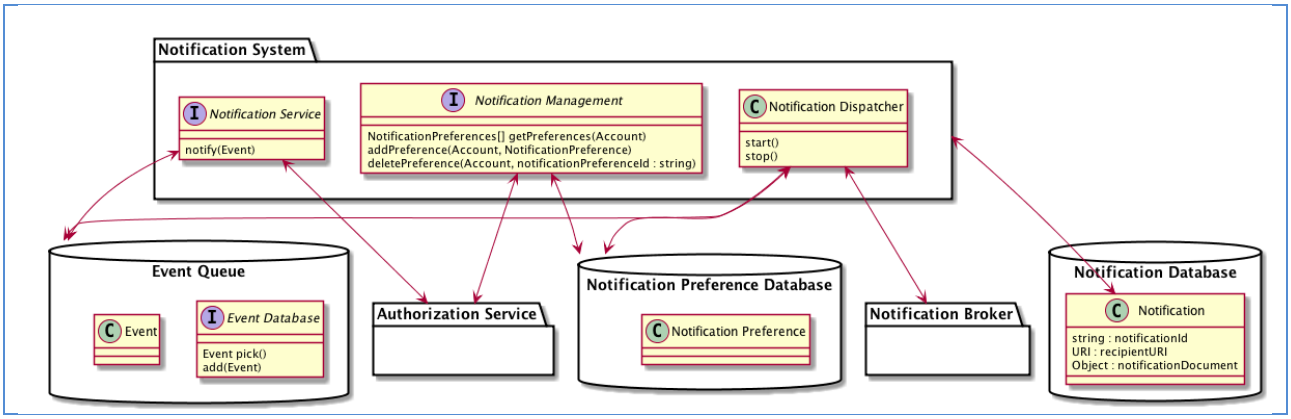


Figure 18: Notification Service (server-side) Structure

6.2.6.2 Component Justification

The need of a notification service component is linked to the corresponding logical component (Section 5.4.2) and its associated logical use cases and business use cases.

6.3 Participant Toolkit

The Participant Toolkit is built of components that have to be integrated directly in the participant's IT systems in order to consume CREDENTIAL Wallet's services. A special case is the cryptographic service, which does not consume any online service and is necessary in both, the Wallet and in the participant's IT systems.

6.3.1 Authentication Client

The authentication client provides the means to integrate with the CREDENTIAL's authentication services. On one hand it provides a client that allows consuming all the services described in the Authentication service (Section 6.2). On the other hand, it must provide access to native libraries or functionalities. E.g. In order to have the authentication client in an Android mobile to use FIDO standard, the authentication client must interface with FIDO's native UAF client. In other cases, i.e. the CRAM authenticator, the client may include additional logic in order to access local services or hardware to unlock, store or use local credentials (e.g. sign a challenge using a private key stored in a secure element). This authentication client should be presented to the end user through very usable UI to be provided by the integrator of the Participant Toolkit. The main objective of this component is to abstract developers of the client application of the complexity of communications with the server side, error handling, etc.

6.3.2 Access Management Client

The Access Management client is basically a technology-specific direct implementation of a library to access the Access Management Service functionalities. Please see the description of the Access Management Service for relationships of this development component with logical components as well as related use cases.

6.3.3 Account Management Client

The Account Management client is basically a technology-specific direct implementation of a library to access the Account Management Service functionalities. Additional functionality will be added to interact with the cryptographic services which will allow the secure management and operation of all cryptographic material related to the account management (e.g. account key pairs, recovery keys, etc.)

6.3.4 Data Management Client

This component provides a conceptually unified interface for client-side application developers. We focus upon the "conceptual" part as neither the concrete application programming languages nor execution environments are fixed but are subject to change. What this conceptual model achieves is that application developers will be able to reuse the same concepts through a well-known and documented API on the different target platforms.

Delivery in library form hides implementation details form the client-side application logic. This allows preventing common security problems, e.g., insufficient confidentiality and integrity protection on the transport layer, by enforcing suitable protection mechanisms within the library. In addition, the high abstraction level allows exchanging implementation details, e.g., the used network transport mechanism such as TCP/IP, HTTP or WebSockets, while minimizing application-level changes. In a certain sense, the library allows for parallel development of end-user applications and back-end applications.

Initial versions of the client-side library can be automatically generated using automated tools and relying on an API definition which enables to generate multiple client stubs for different platforms. Additionally, added features might mandate the move to manually written client-side libraries. In this case, the automatically generated libraries and API will be utilized as an initial starting point. API changes will maintain compatibility with this initial (automatically generated) version.

6.3.4.1 Component Justification

This component provides client applications with a unified interface and thus streamlines the clients' implementation effort. It is not strictly necessary but will prevent redundant functionality within the different clients and thus reduce development effort.

6.3.5 Notification Client

The Notification Client component has the purpose to enable components within the CREDENTIAL Wallet and user devices from outside to communicate with the Notification System and to receive notifications. The client component is split into two separate submodules: one to be used by other CREDENTIAL Wallet Services (internal client) and one that is expected to be deployed and operated in a client environment (e.g. Smartphone). The internal client component has an interface to propagate events towards the Notification Service. It propagates events by using the Notification Broker as introduced in Section 6.2.6. The client creates the event and publishes it through the notification service interface.

The Notification Client however, is also supposed to be operated in the user's client device. It has the interfaces to render, add new or delete existing notifications and notification preferences for his account. The render operation calls basically the getPreferences endpoint of the Notification System and renders the received notification preferences and notifications in an appropriate format on the user's device.

Requests directed to the notification service will be previously processed by the request filters, ensuring that all requests received are authenticated, authorized and sanitized.

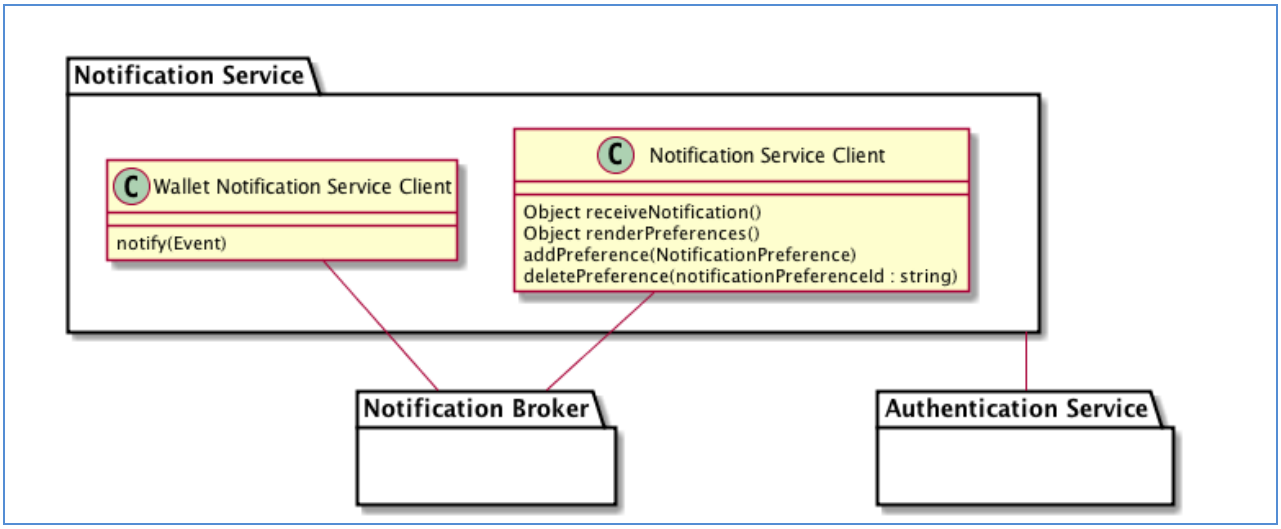


Figure 19: Notification Service (client-side) Structure

6.3.6 Cryptographic Service

The **Cryptographic Service** consists of 4 major components responsible for providing the encryption primitives, advanced cryptography, signatures and key management. As this library will be used by different types of participants, the CREDENTIAL Wallet centric system, the clients and the service providers, a core library will be implemented as well as a library for the CREDENTIAL Participant Toolkit extending the core library with system specific code (e.g. Android). The library on the client side thereby will make use of the Trusted Execution Environment (TEE) provided by Android, the ARM TrustZone.

6.3.6.1.1 Encryption

The encryption component provides primitives ensuring the confidentiality of the user data. It implements proxy re-encryption representing a key feature in CREDENTIAL context. In order to provide more fine grained delegation permissions, conditional proxy re-encryption is supported. For obtaining efficient encryption performances, hybrid encryption is applied.

- **Key Generation:** $KeyGen(\lambda) \rightarrow (pubKey,\ privKey)$
  The $KeyGen$ operation is an algorithm to generate an asymmetric proxy re-encryption key pair consisting of the private key $privKey$ and the corresponding public key $pubKey$ while providing a security parameter λ defining cryptographic primitive attributes like key length or curve type as input.

- **ReEncryption Key Generation:** $ReKeyGen(privKey, pubKey) \rightarrow (rk)$
  A re-encryption key for delegating a ciphertext can be generated using the *ReKeyGen* method. Attributes necessary for generating this key might depend from the scheme to be implemented, but usually the delegators private key and the delegates public key are required.
- **Conditional Key Generation**: $CKeyGen(privKey, cond) \rightarrow (ck)$
  Using the *CKeyGen* operation a conditional key *ck* can be generated requiring the users private key *privKey* and the condition *cond* as input. This conditional key generation can be combined with the re-encryption key.
- **Encryption**: $Enc(p, pk, cond) \rightarrow (c, ek)$
  The Enc operation is used to transform a plaintext p into a ciphertext c using a hybrid encryption scheme. As parameter a public key pk generated by the $KeyGen$operation is necessary next to the message to be encrypted. Optionally a condition *cond* can be provided in order to limit the delegation of the ciphertext by third parties to specific conditions. As result the encrypted message *c* next to the encapsulated symmetric encryption key *ek* is returned.
- **Decryption:** $Dec(c, privKey, ek) \rightarrow (p)$
  In order to regain a plaintext p from a ciphertext the Dec operation is applied. Therefore the private key $privKey$ corresponding to the public key used for encryption has to be provided as parameter next to the ciphertext c and the encapsulated symmetric key *ek*.
- **Re-Enryption:** $ReEnc(c, rk, ck) \rightarrow (c^*)$
  Delegating a ciphertext can be achieved using the *Re-Encryption* operation. Therefore the original ciphertext c and the generated re-encryption key rk have to be provided as attributes resulting in a new ciphertext c^* decryptable by the delegate. Additionally, it is possible to pass a conditional key *ck*.

6.3.6.1.2 Digital Signatures

The Signature Service component ensures the integrity and authenticity of exchanged data. For example, a user or issuer of data signs the message before uploading it to the cloud-based CREDENTIAL Wallet. While sharing such a signed message, the CREDENTIAL Wallet might redact (or black-out) unnecessary parts of the message that the user does not want to expose to realize selective disclosure. Once this signed message was shared with a recipient, such as a service provider or doctor, the authenticity and integrity of the received message can be verified. This signature service consists of two sub-components, which are described in the following paragraphs.

The **Traditional Signature Service** provides functionality to digitally sign documents as well as to verify the digital signature for a document. The main operations provided by this service are:

- **Key Generation:** $KeyGen(\lambda) \rightarrow (sk, pk)$
  The $KeyGen$ operation is an algorithm to generate a key pair consisting of the private (secret) key sk and the corresponding public key pk with utilizing a security level parameter λ as input. The key pair is generated for the signer.
- **Sign:** $Sign(sk, msg) \rightarrow (\sigma)$
  The $Sign$ operation takes as input a message msg and the private key sk. From the message's content, a signature σ is created, which is the operation's output.
- **Verification:** $Verify(msg, \sigma, pk) \rightarrow val$

The verification operation $Verify$ takes as input the message msg, the signature σ and the public key pk of the signer. The return value val of this algorithm shows if the signature for the given message is valid.

The **Redactable Signature Service** extends the Traditional Signature Service, as it also allows to redact (black-out) parts of an already signed document. The resulting modified document and signature can still be verified against each other. The main operations of this service are:

- **Key Generation:** $KeyGen(\lambda) \rightarrow (sk, pk)$
  The $KeyGen$ operation is an algorithm to generate a key pair consisting of the private (secret) key sk and the corresponding public key pk with utilizing a security parameter λ as input. The key pair is generated for the signer.
- **Sign:** $Sign(sk, msg) \rightarrow (\sigma)$
  The $Sign$ operation takes as input a message msg and the private key sk. Each message block of the message is used to create the signature σ, which is the operation's output.
- **Redact:** $Redact(msg, \sigma, pk, mod) \rightarrow (msg', \sigma')$
  The redact operation $Redact$ takes as input the message msg, the signature σ, the public key pk of the signer and an instruction mod for modifying the message. A modification instruction, basically, describes which message blocks from the message have to be redacted (removed). After the message has been modified according to the instruction the algorithm updates the signature accordingly. The modified message together with the modified signature is returned.
- **Verification:** $Verify(msg', \sigma', pk) \rightarrow val$
  The verification operation $Verify$ takes as input the message msg, the signature σ and the public key pk of the signer. Depending on the scheme it can also require the hash value of the redacted message block. The return value val of this algorithm shows if the signature for the given message is valid.

6.3.6.1.3 Advanced Cryptography

Encrypted attribute-based credential (eABC) systems consist of a set of algorithms for generating parameters and key material for all involved parties, as well as interactive protocols for issuing credentials to users as well as for presenting those credentials to service providers.

- **System parameter generation:** $ParGen(1^{\lambda}) \rightarrow (syspar)$
  This algorithm generates the necessary system parameters. Those could include the used key sizes for the required security parameter, the number of supported attributes per credential, or the attribute space A. If needed by the deployed cryptographic schemes, the system parameters could also include common reference strings (i.e., potentially cryptographic values that are available to all parties). The system parameter generation algorithm is usually (assumed to be) executed by a trusted party, and only needs to be executed once when setting up the system. Depending on the precise parameters (e.g., whether or not they contain cryptographic keys that need to be well-formed), this algorithm may in practice also be executed by the software vendor who then ships the parameters to use with the software.
- **Issuer Key Generation:** $KGen_I(syspar) \rightarrow (sk_I, pk_I)$

Each issuer (i.e., party giving credentials certifying certain attributes to users) executes this algorithm to compute a secret/public key pair. In particular, the issuer will generate a key pair for a (potentially but not necessarily redactable) signature scheme, and use the signing key when certifying attributes for users.

- **User Key Generation:** $KGen_U(syspar) \rightarrow (sk_U, pk_U)$
  Each user in the system runs this algorithm to compute its own secret/public key pair. In particular, this includes the computation of a proxy re-encryption key pair as specified in Section 6.3.6.1.1.

- **Service Provider Key Generation:** $KGen_{SP}(syspar) \rightarrow (sk_{SP}, pk_{SP})$
  Each service provider in the system runs this algorithm to compute its own secret/public key pair. In particular, this includes the computation of a proxy re-encryption key pair as specified in Section 6.3.6.1.1.

- **Issuance:**
  $<U.Issue\big(syspar, (a_i)_{i=1}^{l}, A, D, pk_I, pk_U, sk_U\big),\ \ I.Issue(syspar, (a_i)_{i\in D}, A, D, sk_I, pk_I) >$

  $\rightarrow cred$
  Issuance is an interactive protocol between a user and an issuer. Depending on the concrete eABC system this may be a two-round protocol (i.e., one message being sent by each party) or a multi-round protocol.
  Both parties take as inputs the system parameters, the supported attribute space A, the indices D of the disclosed attributes (i.e., those that the issuer is allowed to see and from which it might decide whether or not to issue a credential; in practice, D will often be pre-defined by the issuer who needs, e.g., to check the user's nationality or age), and the public key of the issuer pk_I. In addition to that, the user takes as inputs the values of all his attributes $(a_i)_{i=1}^{l}$ (where the number l must not exceed a potential limit in the system parameters), as well as his own public and secret key. The issuer additionally takes the attribute values of the disclosed attributes and its own secret key as inputs.
  At the end of the (interactive) protocol, the user receives as an output the credential $cred$ which is then stored on the Wallet, together with a re-encryption key from the user to the service provider. In particular, $cred$ consists of: encryptions $(c_i)_{i=1}^{l}$ of the attributes under his own public key, a signature sig (among others) on those encrypted attributes, as well as potentially other auxiliary information depending on the concrete instantiation of the protocol. In case of an error, both parties receive a respective message.

- **Presentation:**
  $< W.present(syspar, cred, pk_U, pk_{SP}, rk_{U\rightarrow SP}, pk_I, R),$

  $SP.Present(syspar, pk_{SP}, sk_{SP}, pk_I, R) >\rightarrow (a_i)_{i\in R}$
  Presentation is again a protocol potentially including multiple communication rounds between the Wallet and the service provider. Both parties take as inputs the system parameters, the service provider's public key pk_{SP}, the issuer's public key pk_I, as well as the indices R of the attributes to be revealed to the service provider (similar to before, these indices will typically be requested by the service provider, and the user is asked for his consent before the protocol continues). The Wallet additionally takes the credential $cred$, the user's public key pk_U, as well as the re-

encryption key from the user to the service provider, $pk_{U \to SP}$. The service provider also takes its secret key sk_{SP} as input.

At the end of a successful interaction, the service provider obtains attributes $(a_i)_{i \in R}$ for further processing. In case of an error, both parties receive a respective message.

We want to stress that redaction is an implicit part of this protocol. Namely, all un-revealed attributes (i.e., those not in R) remain hidden from the service provider, i.e., they are "redacted" during the presentation. However, depending on the concrete instantiation, there might not be an explicit call to the $Redact$ interface of the signature scheme, as in eABCs re-encryption and redaction are sometimes very intertwined and cannot be separated from each other. Even more, certain instantiations do not even use a redactable signature scheme as underlying building block but achieve this functionality through zero-knowledge proofs of knowledge or similar primitives also for standard signature schemes.

The **Combination of Proxy Re-Encryption and Redactable Signatures** consists of the following set of algorithms:

- **Signature Key Generation:** $SignKeyGen(1^\lambda) \to (sk_{sign}, pk_{sign})$: On input of a security parameter λ, this probabilistic algorithm outputs a signature keypair (sk_{sign}, pk_{sign}).
- **Encryption Key Generation:** $EncKeyGen(1^\lambda) \to (sk_{Enc}, pk_{Enc})$: On input of a security parameter λ, this probabilistic algorithm outputs an encryption keypair (sk_{sign}, pk_{sign}).
- **Re-Encryption Key Generation:** $ReKeyGen(sk_{Enc,A}, pk_{Enc,B}) \to (rk_{A \to B})$: On input of a private encryption key $sk_{Enc,A}$ belonging to user A and a public encryption key $pk_{Enc,B}$ of user B, this (probabilistic) algorithm outputs a re-encryption key $rk_{A \to B}$.
- **Redactable Signature:** $Sign(sk_{sign}, M, \boldsymbol{ADM}) \to (M, \sigma, red)$: On input of a private signature key sk_{sign}, a message M, and ADM, this (probabilistic) algorithm outputs a message signature pair (M, σ) together with some auxiliary redaction information red.
- **Encrypt:** $Encrypt(pk_{Enc,A}, M) \to C_A$: On input of a public encryption key $pk_{Enc,A}$ and a message M, this (probabilistic) algorithm outputs a ciphertext C_A.
- **Redact:** $Redact(\sigma, C_A, MOD, red) \to (C'_A, \sigma', red')$: This (probablistic) algorithm takes a valid signature σ_A for a message M encrypted as ciphertext C_A, modification instructions MOD and auxiliary redaction information red as input. It returns a redact encrypted message-signature pair (C'_A, σ'_A) and an updated auxiliary redaction information red'.
- **ReEcnrypt:** $ReEncrypt(rk_{A \to B}, C'_A) \to C'_B$: On input of a re-encryption key $rk_{A \to B}$ and a ciphertext C'_A, this (probablistic) algorithm returns a transformed ciphertext C'_B, of the same underlying message.
- **Decryption:** $Decrypt(sk_{Enc}, B, C'_B) \to M'$: On input of a private decryption key $sk_{Enc,B}$ and a ciphertext C'_B, this deterministic algorithm outputs the underlying plain message M'.
- **Signature Verification:** $Verify(pk_{sign}, M', \sigma') \to valid \in \{0,1\}$: On input of a public key pk_{sign}, a signature σ' and a message M, this deterministic algorithm outputs a bit $valid \in \{0,1\}$.

6.3.6.1.4 Key Management

The **Key Management Service** on the other hand is responsible for the persistent storage of user keys as well as providing access to those keys.

- **Store Keypair:** $storeKeyPair(kp, \lambda)$ -> ()

The *Store Keypair* operation is used for storing a keypair in a secure manner on the client device. A key parameter λ containing information for identifying and securing the keys has to be provided as additional input parameter.

- **Get Private Key:** $getPrivateKey(\lambda)$ -> (k)
  Returns a private key. A key parameter λ containing information for identifying and securing the key has to be provided as input parameter.
- **Get Public Key:** $getPublicKey(\lambda)$ -> (k)
  Returns the public signature key. A key parameter λ containing information for identifying the key has to be provided as input parameter.
- **Store Re-Encryption Key:** *storeReEncKey(k,,* λ) -> ()
  The *Store Re-Ecnryption* operation is used for storing re-encryption keys in a secure manner on the client device. A key parameter λ containing information for identifying and securing the keys has to be provided as additional input parameter.
- **Get Re-Encryption Key:** $getReEncKey(\lambda)$ -> (k)
  Returns a re-encryption key. A key parameter λ containing information for identifying the key has to be provided as input parameter.

6.3.6.2 Component Justification

This component joins several logical components (Re-Encryption Key Generation Service, Personal Trust Store, Encryption Service, Decryption Service and Re-Encryption Service) in one unique development component and is related to cryptographic logical use cases defined in D2.1.

6.4 Request Filters

The last group of development components are general filters that provide some horizontal functionality. They are mostly related to securing and auditing the system. E.g. components required ensure all accesses are legitimate or to collect necessary auditing information.

6.4.1 Access Control Filter

The filter automatically intercepts incoming requests and verifies the user's authorization for the requested operation. To achieve that it communicates with the authentication and authorization services. If authorization fails and or error/exception is generated, then the operation is not executed. The control filter also interacts with the auditing service in order to store ACL decisions.

The authorization decision can be retrieved through multiple means, depending on the to-be-protected service:

- UMA to protect the Data Management Service: This filter protects the CREDENTIAL endpoints though OAuth's UMA profile. Upon request to an endpoint, the filter establishes a permission ticket through the Protection API of the UMA Authorization Server offered by the Access Management Service. After the requester obtained a Relying Party Token (RPT) from the UMA Authorization Server, the filter inspects this token. Given sufficient permissions included in the token, the filter allows access to the requested endpoint.
- OAuth to protect other services: The Filter inspects access tokens attached to the incoming requests. If no such token is provided or this token represents insufficient permissions, the

requester has to obtain an access token with the required permissions for the OAuth Authorization Server, which is part of the Access Management Service.

6.4.1.1 Interface

There is no external interface of this component. Instead it could extend standard language or technology-specific interfaces (e.g. ContainerRequestFilter for JAX-RS based implementations) to intercept all service requests and responses and acts upon user data extracted from those operations.

The filter operation should include data about the incoming request that would provide enough information to match the operation against the access control policies.

In detail, the following steps will be performed:

1. Extract user credentials, i.e. session id, from the incoming request and query User Authentication Service to verify its validity
2. Extract the operation's target (object), query the Access Management Service if the combination of user, operation and operation target is allowed to proceed.
3. If acknowledgement was received, allow the operation to pass; otherwise return error message.
4. If configured, forwarded high-level logging information to the server-side Audit Subsystem. The logged data will be sanitized according to our privacy guidelines prior to submission. This minimizes the potential privacy implications of too detailed audit trails.

6.4.1.2 Component Justification

As all the components are performing operations upon sensitive data, each of those operations is subject to access control. This implies that it is related or part of all components. To prevent code redundancy access control is implemented as a filter which is transparently executed before each operation. This component is related to Authorization Service and all authorization related logical use cases, but especially to the Authorization logical use case.

6.4.2 Auditing Filter

The filter automatically intercepts all incoming service requests and forwards the extracted information to the server-side auditing service.

Summarizing, the following steps will take place during the Audit Filter:

1. The incoming HTTP request is assigned a unique identifier.
2. Information regarding the request is forwarded to the server-side audit system, where it will be persisted
3. The request is forwarded to the next filter in the filter chain;
4. The filter remains in an awaiting status until the response is routed back through the same filter chain.
5. When the corresponding response from the application server to the client is received, the data is matched to the already stored incoming request. The unique identifier is added to the record and again the resulting data is forwarded to the server-side auditing subsystem for storage.

Please note, that the potentially captured information can be highly sensitive. While implementing the capture filter we will heed no follow Privacy-by-Design principles (e.g. data minimization) to mitigate the data collection's privacy impact. Access to the stored log information itself is subject to access control and logging as defined in deliverable D2.5.

6.4.2.1 Interface

There is no external interface of this component. Instead it is proposed to follow technology specific best practices (e.g. J2EE filters) to intercept all service requests and responses. This is very similar to the Access Control filter; the main difference is that the Audit Filter is passive, i.e., cannot prevent an operation from occurring, while the Access Control Filter can prevent an operation from being called.

The filter operation should include data about the incoming request. The Filter will extract information from this data and forward this information to the Auditing subsystem. While the filter can theoretically throw an exception, due to the passive nature of the prototype this is not expected behaviour..

6.4.2.2 Component Justification

Logging and Auditing are cross-cutting concerns; the audit component can thus be seen as an aspect of the program. An aspect if a feature linked to many other parts of the program, but which is not related to the program's primary function. In this case, auditing will be performed for all incoming operations --- thus all logical use cases implicitly use this aspect. It is also linked to CREDENTIAL Audit Trail Service logical component and its related use cases.

This component is a low-level data gathering part of the Audit Trail Service. This service will compact, aggregate and analyse audit trail information from multiple sources and thus provide a comprehensive audit trail for the whole system.

6.4.3 Data Sanitization Filter

According to the OWASP Top 10 Vulnerabilities list[3], two of the three most common attack vector are injection attacks. These attacks are performed by inclusion of malicious code within data passed on to services. The malicious code may ether be executed within the service or be executed within a client displaying the malicious code.

The CREDENTIAL system will mitigate this threat by performing data sanitization as early as possible. This filter automatically intercepts all incoming service requests, sanitizes input data and forwards the resulting requests to the next input filter. The sanitization step will be performed by State-of-the-Art third-party libraries specializing into detecting malicious source code fragments[4].

Summarizing the following steps will take place during the Data Sanitization Filter:

[3] https://www.owasp.org/images/f/f8/OWASP_Top_10_-_2013.pdf
[4] https://github.com/OWASP/owasp-java-encoder/

1. Perform an input sanitization to prevent malicious code from entering the system. This is done through usage of existing data verification libraries. The initial version will perform basic security checks, more advanced versions of this filter might use different filter libraries depending upon the called service and method.
2. If malicious code was detected, this information is forwarded to the server-side audit system as this is an indication of an ongoing attack. The filter can be configured to either pass-on the unsanitized data ("intrusion detection mode") or to automatically filter malicious requests ("intrusion prevention mode").
3. If configured, the resulting "secure" request is forwarded to the next incoming operation filter.

Please note, that the potentially captured information can be highly sensitive. While designing the capture filter we will heed Privacy-by-Design principles to mitigate the data collection's privacy impact. We assume that all access to the stored log files itself is subject to access control and logging as defined in deliverable D2.5.

6.4.3.1 Interface

There is no external interface of this component. Instead it could extend standard language or technology-specific interfaces (e.g. ContainerRequestFilter for JAX-RS based implementations) to intercept all service requests and responses and acts upon user data extracted from those operations. The component will ensure that all components operating after this filter will only access sanitized request parameters

6.4.3.2 Component Justification

Injection attacks will happen during the lifetime of a deployed CREDENTIAL system. Each server component of CREDENTIAL is required (as defined within D2.5) to cope with this class of attacks. The incoming Data Sanitization Filter is an additional (and optional) security mechanism that prevents malicious requests from entering the system at all. In addition, new attack patterns will be identified after a CREDENTIAL system has been installed, the global Data Sanitization Filter allows CREDENTIAL to add support for filtering new malicious code by updating a single (non-functional) component. The alternative would be to change the input processing of all components that might process data with the new malicious code pattern. The former is less invasive.

7 Process Architecture

As described in the methodology description (see Section 2.5) the process view of the architecture *shows its* dynamic aspects and how each of the system's components interacts with each other towards a common goal. Architectural significant business and logical use cases will be decomposed in terms of development components, resulting in a set of **architectural significant generic technical use cases.**

In order to achieve this objective a two-step approach has been followed. The first one has been to merge the individual logical use cases that appear in D2.1 in a reduced number of coherent stories that clearly map the use cases to real world goals. The second step has been to map the story to the development components designed in Section 6 and their services and APIs.

The following subsections include the resulting stories interleaved with references to logical use cases described in D2.1and sequence diagrams showing technical uses cases, or, how the development

components collaborate towards the user's objectives. Use cases referred in the story use a [use case id] notation; these ids can be used to find the complete use case definition in D2.1.

7.1 Notation and assumptions

In order to correctly understand the following UML sequence diagrams some clarifications are included:

- Green lanes or lanes with green background correspond to components or actors that are in the domain of the participant (end-user);
- Yellow lanes or lanes with pale yellow background correspond to components or services that are deployed within the CREDENTIAL Wallet;
- A special case is filters denoted with the circle arrow symbol. These filters will be always be applied to all requests going to the Wallet. For simplicity, these are only shown in some diagrams were its presence is relevant;
- Service provider entity is denoted with a dark yellow colour;
- Cylinder icon is used to denote data stores;

7.2 Alice Creates and Tests her CREDENTIAL Account

Alice heard about the CREDENTIAL tool, and wants to use it, e.g., as an identity provider (IDP) for various service providers (SP) such as her favorite movie streaming service. She thus downloads the CREDENTIAL software. By running the software, Alice generates a public and private key pair [G-LUC-GENKEYPAIR] as well as a recovery key. Further, a re-encryption key from the Alice' public key and the recovery key is generated [G-LUC-CREATECREDACCREQ]. The software sends all keys (except the private key) and necessary user information to the server where the CREDENTIAL Wallet is located and an account within this Wallet is created [G-LUC-SUBMITCREDACCREQ, G-LUC-CREATECREDACC].

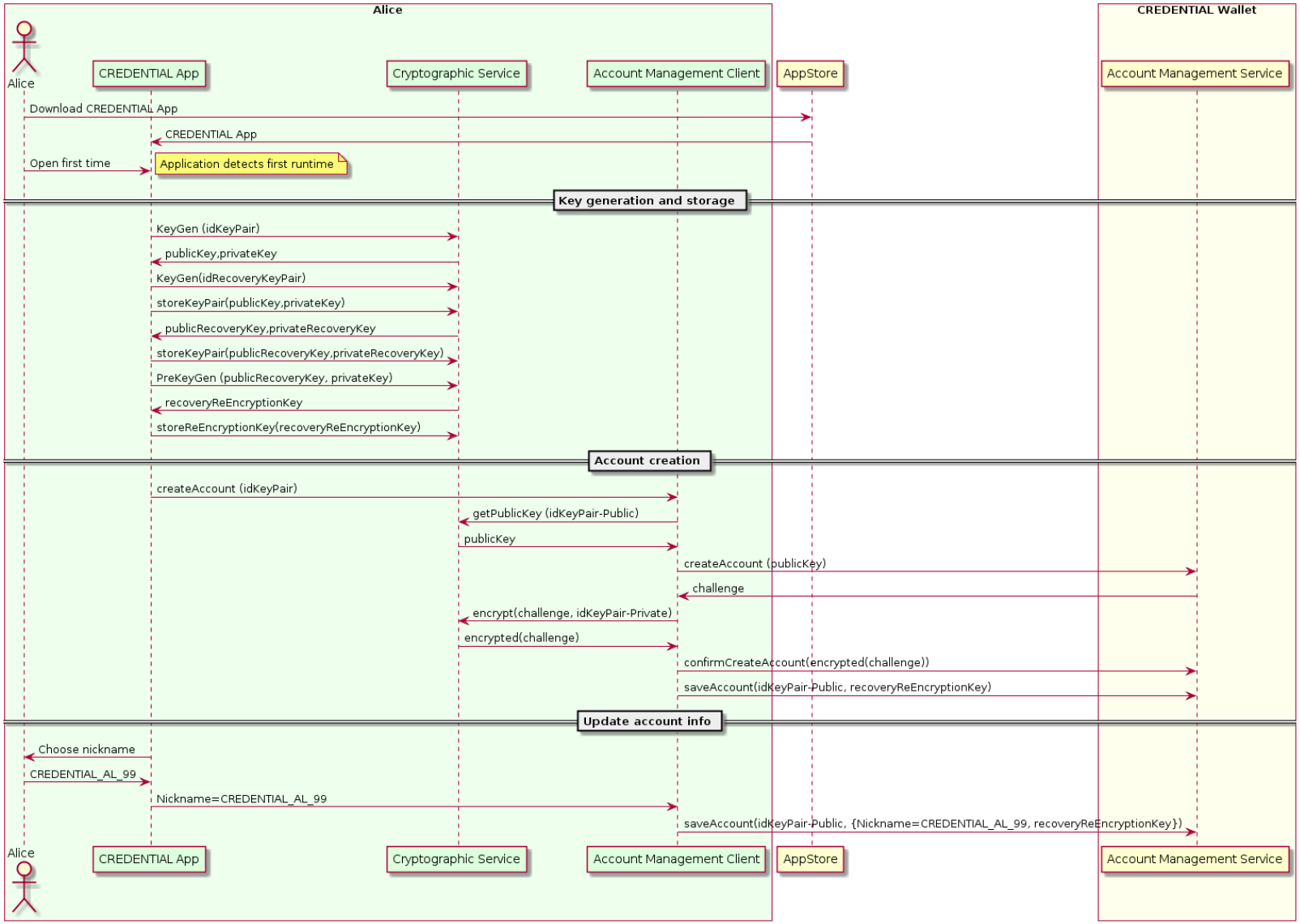


Figure 20: Sequence Diagram of Account Creation

Furthermore, Alice follows best practices (and the recommendations on the CREDENTIAL webpage) and creates a backup of her private key material. She uses the CREDENTIAL software to securely export the recovery key by storing it in a personal USB stick or it prints it as a QR image [G-LUC-EXPPRIVKEY], which she stores in a secure place, e.g., a bank vault. To double-check and verify if storing her private key was successful, Alice imports the key by scanning the QR image with her mobile using the CREDENTIAL app [G-LUC-IMPPRIVKEY].

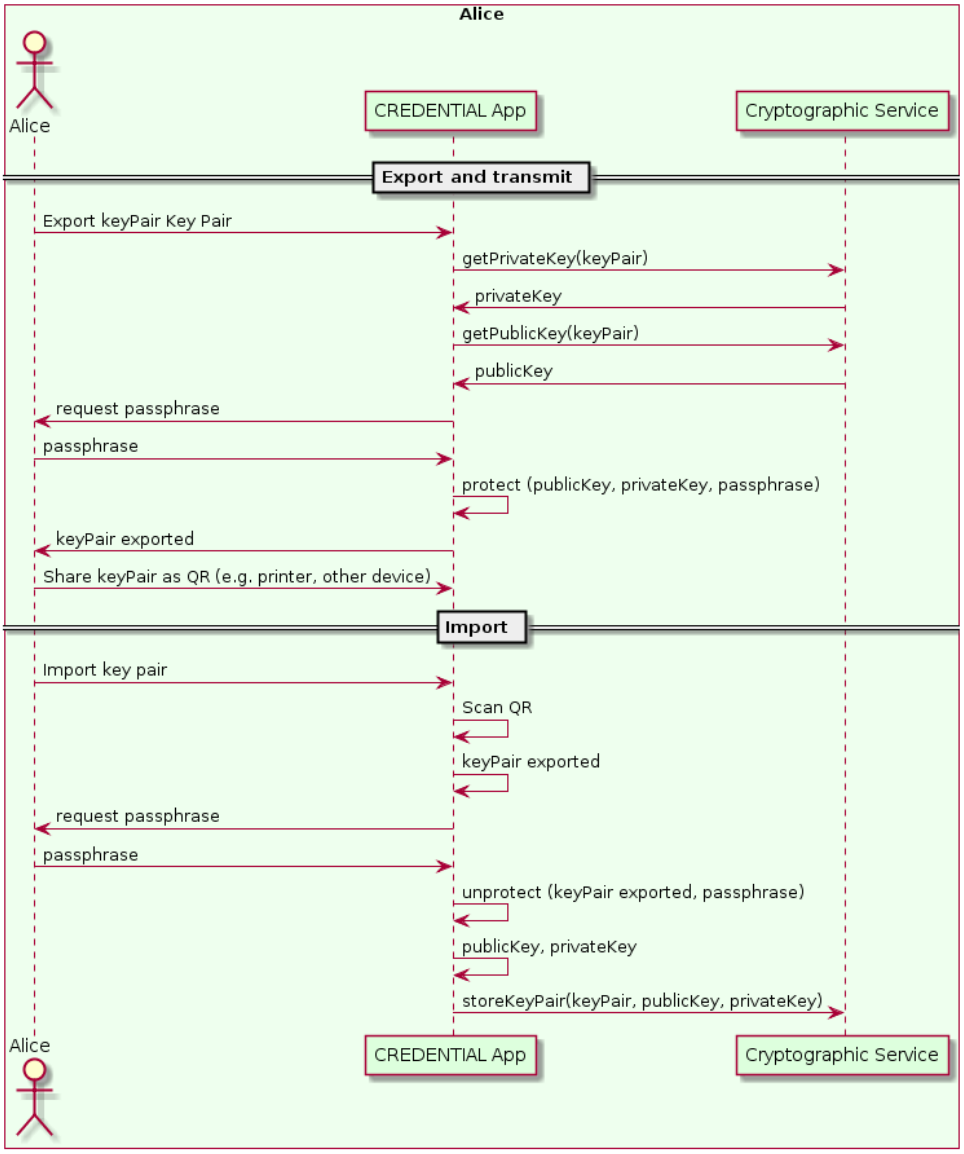


Figure 21: Sequence Diagram Corresponding to Inporting and Exporting Cryptographic Keys

Next, Alice starts to fill her data into the Wallet. She provides, e.g., her name, her birthdate, and her billing information to the CREDENTIAL software which is encrypted internally, send to the Wallet and successfully registered [G-LUC-ENCDATACREDENTIAL, G-LUC-SENDENCDATA, G-LUC-REGDATA, G-LUC-DATAREGCONF].

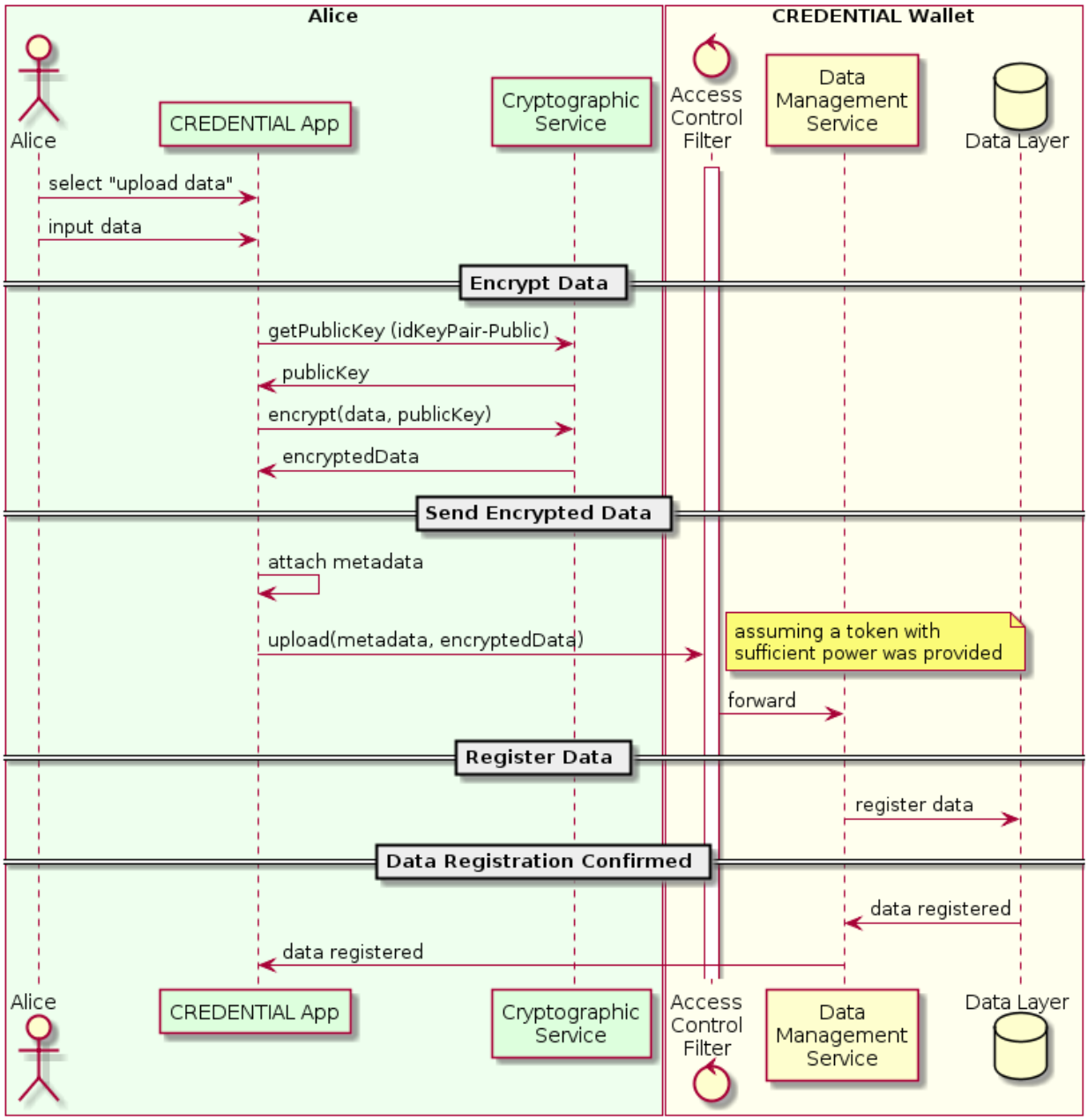


Figure 22: Uploading Data to the Wallet

To benefit from all the features of CREDENTIAL, in particular the IDP functionality, Alice would now like to link the service provider for streaming videos to her CREDENTIAL account. She therefore logs into the streaming service [G-BUC-AUTHWALLET], and links her existing account there to her new CREDENTIAL Wallet [G-LUC-LINKACCREQ]. Partially involving Alice to confirm that certain data may be sent, the following steps are now performed in the background: [G-LUC-REDIRECTUSERCREDAUTH, G-LUC-PROVLINKACCREQ, G-LUC-AUTHORIZATION, G-LUC-REGLINKACCREQ].

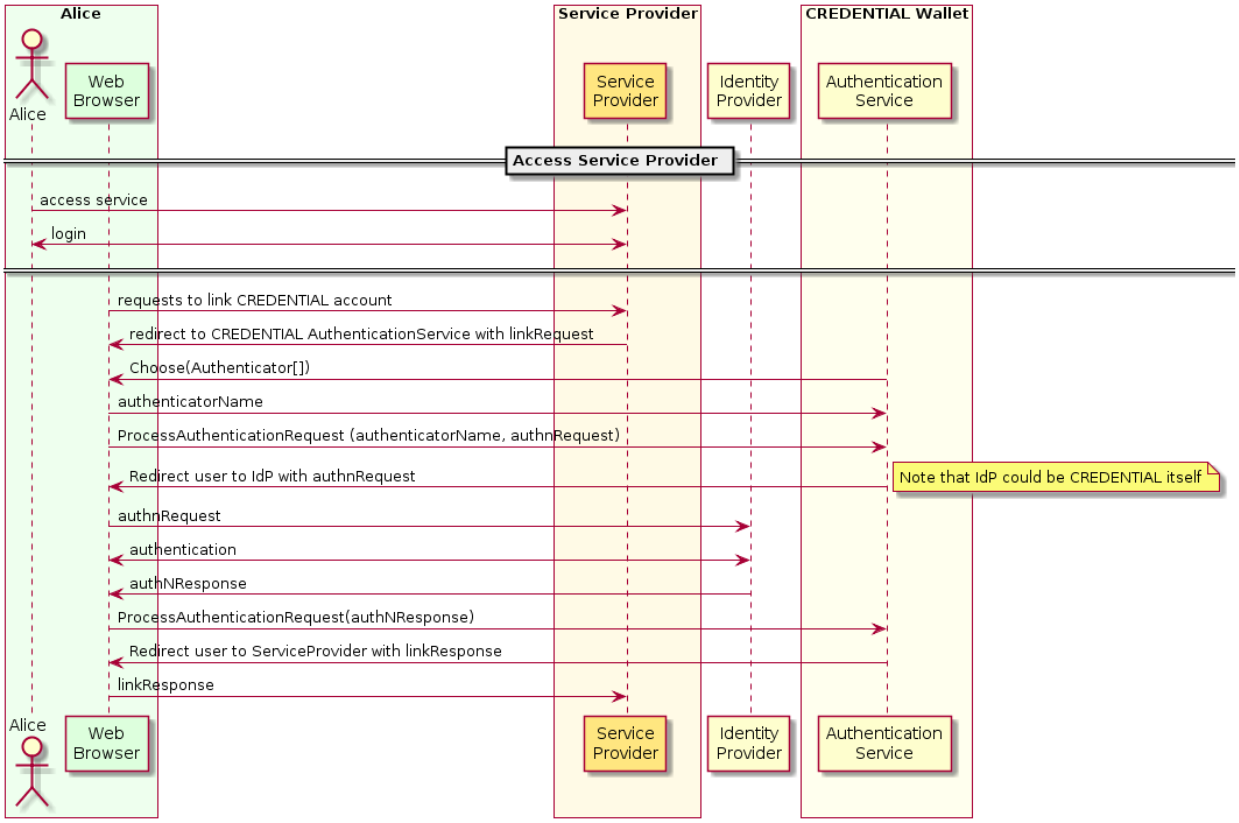


Figure 23: Linking SP and CREDENTIAL Accounts

To guarantee that the correct accounts are linked, Alice is now informed by CREDENTIAL that a linking request has been received [G-LUC-RECEXTTRIGEVENT, G-LUC-EVALNOTIFYCONF, G-LUC-CREATENOTIFYLIST, G-LUC-SENDNOTIFY, G-LUC-PROCNOTIFY]. If Alice does not abort after receiving the notification, the linking process continues where Alice can choose which of her attributes she wants to provide [G-LUC-REQIDENTIYASSERTION, G-LUC-ATTCOLLECTION, G-LUC-SELECTATTRDISCLOSE, G-LUC-REDACTIDASSERTION, G-LUC-PROVIDASSERTION, G-LUC-ENCATTRIBUTES, G-LUC-AUTHORIZATION, G-LUC-ISSIDENTITYASSERTION, G-LUC-AUDITING, G-LUC-RECIDENTITYASSERTION, G-LUC-DECIDENTITYASSERTION, G-LUC-VERIDASSERTION].

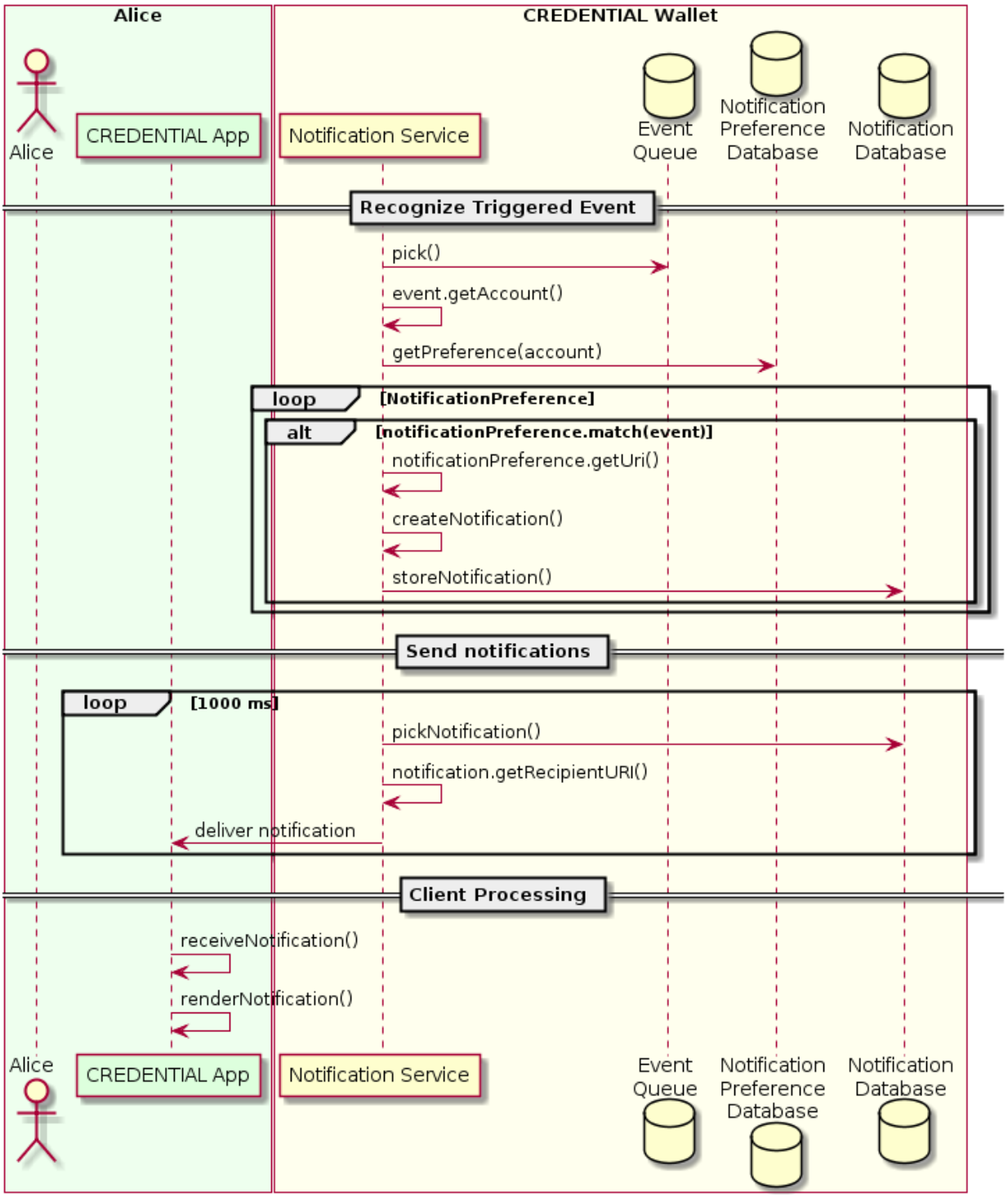


Figure 24: Sequence Diagram Showing the Notification Mechanism

As Alice's friend Bob, who encouraged her to use CREDENTIAL, is still online in some chat program, she wants to give him a treat by sending him a simple file proving that she is now using CREDENTIAL. She therefore signs the file [G-LUC-REQSIGN, G-LUC-CREATESIGNREQ, G-LUC-PROVSIGNREQ, G-LUC-PROVDATASIGNREQ, G-LUC-RECSIGNREQ, G-LUC-PROVSIGNREQ, G-LUC-

SIGNDOC, G-LUC-RECSIGNDOC], encrypts the file and the signature [G-LUC-ENCDATACREDENTIAL], and stores both in the Wallet [G-LUC-SENDENCDATA, G-LUC-REGDATA, G-LUC-DATAREGCONF]. The Wallet will then internally perform some processing [G-LUC-AUTHORIZATION, G-LUC-REGDATA, G-LUC-AUDITING, G-LUC-DATAREGCONF].

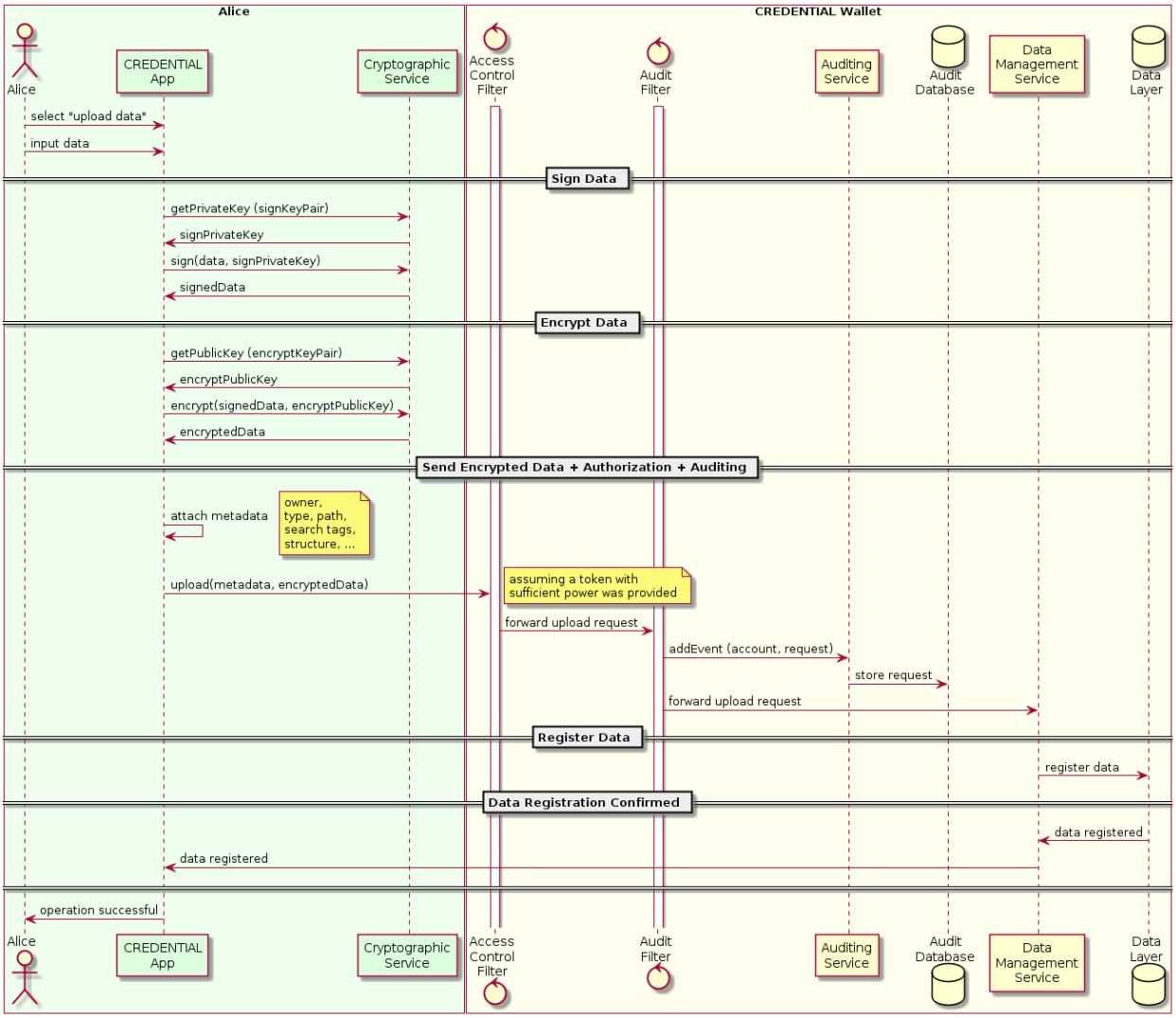


Figure 25: Storing Data in the Wallet

Alice wants to share a file with Bob and sends him a link to that file, but she forgot to grant the required access rights to Bob before. After receiving the link, Bob tries to access the file and gets a notification that no access was granted for that file. Therefore, Bob requests those rights [G-LUC-PROCEXC , G-LUC-REQACCRIGHTSLIST, G-LUC-AUTHORIZATION, G-LUC-REGDATA, G-LUC-AUDITING, G-LUC-PROVACCESSRIGHTSREQ], which is now accepted by Alice [G-LUC-GRANTACCESSRIGHTS, G-LUC-PROVIDEACCESSRIGHTS, G-LUC-REGACCESSRIGHTS]. When Bob now tries to access the file, the following steps are performed: [G-LUC-DEFREQPARAMS, G-LUC-REQDATA, G-LUC-AUTHORIZATION, G-LUC-SEARCHDATA, G-LUC-REENCDATA, G-LUC-AUDITING, G-LUC-RECDATA, G-LUC-DECDATA, G-BUC-READDATA], where the Wallet internally also requests the re-encryption key [G-LUC-REQREK] from its authorization service. After this

successful test of how to share data, Alice deletes the dummy file again [G-LUC-CREATEDELDATAREQ, G-LUC-SUBMITDELDATAREQ, G-LUC-AUDITING, G-LUC-DELDATA].

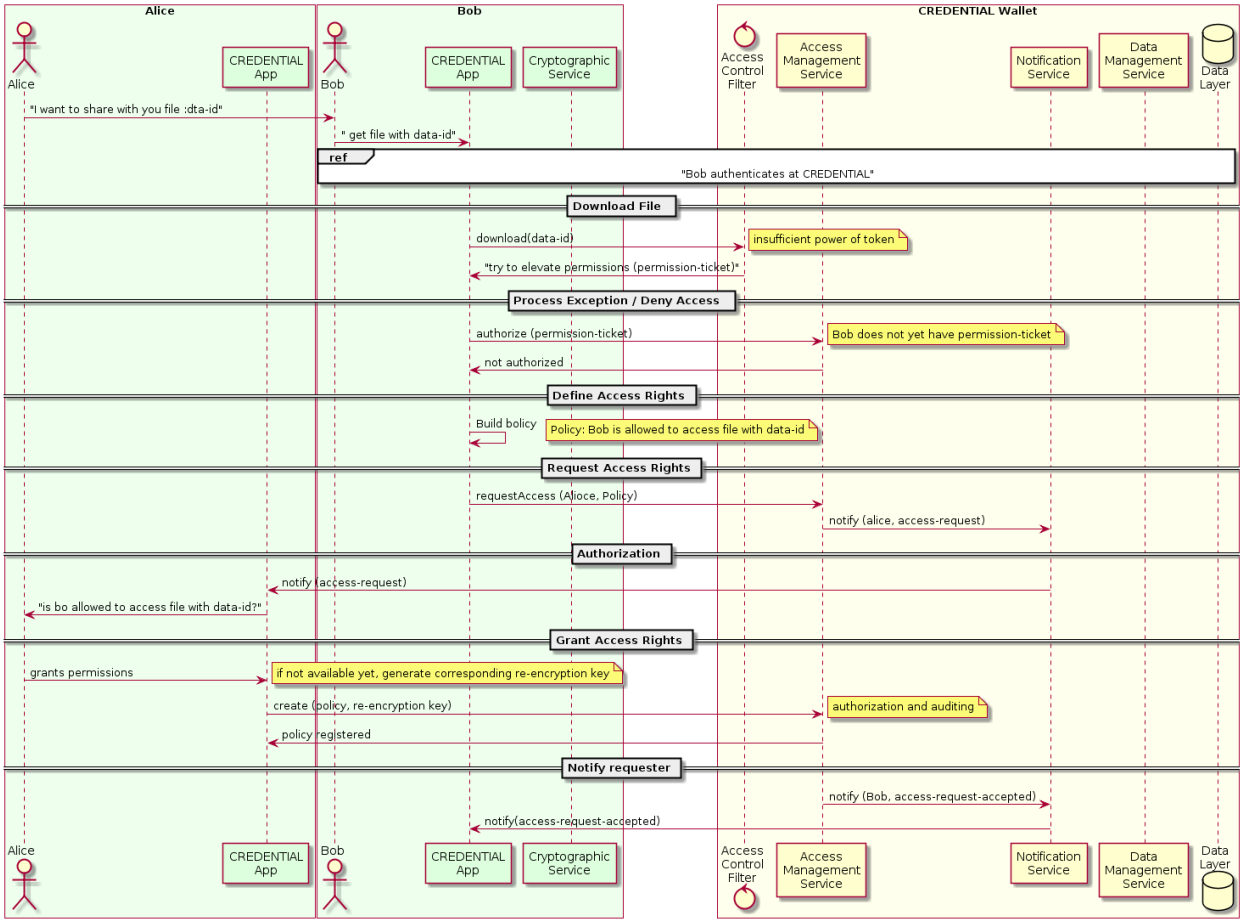


Figure 26: Alice Sharing a File with Bob (1/2)

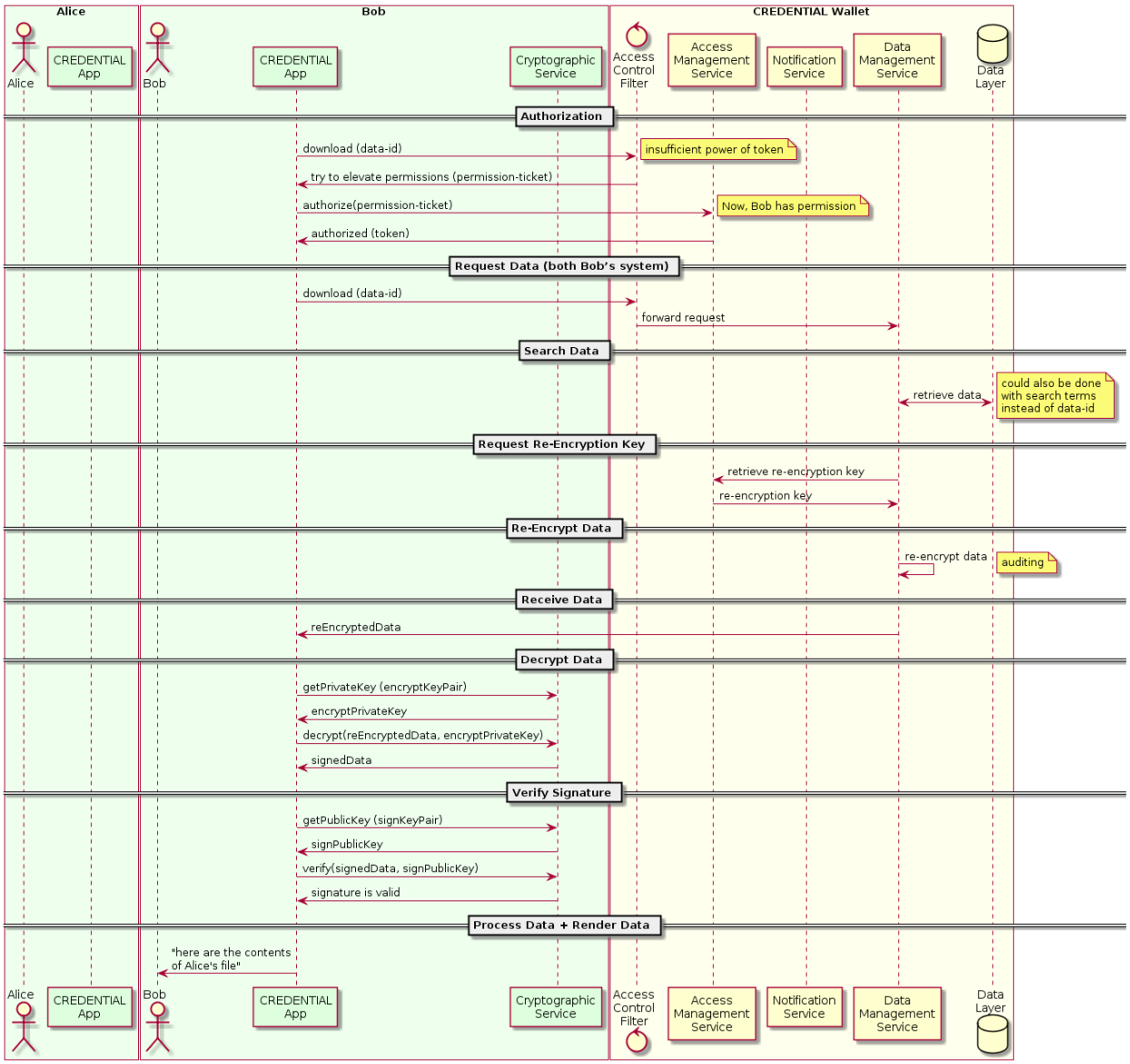


Figure 27: Alice Sharing a File with Bob (2/2)

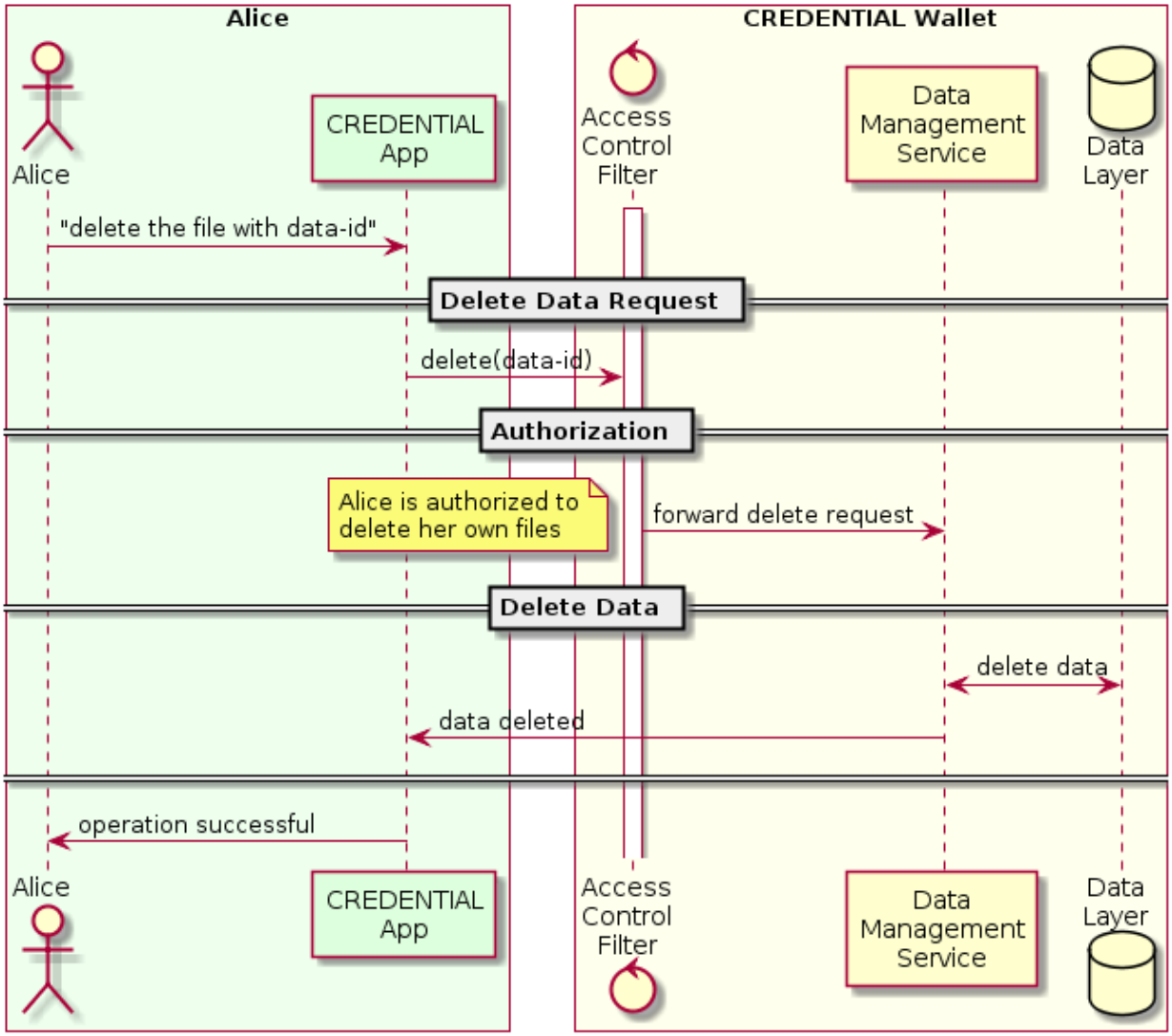


Figure 28: Document Deletion

To relax after a hard work day, Alice finally wants to stream a movie using the account that she linked earlier. She therefore visits the web presence of the movie streaming service [G-LUC-ACCESSSP] and wishes to login via CREDENTIAL. After receiving the login request by Alice, the streaming service asks for a list of IDPs via the CREDENTIAL technology. Alice can then select an IDP and is able to login [G-LUC-ASKIDPS, G-LUC-ASKIDPFROMLIST, G-LUC-SELECTIDP, G-LUC-REDIRECTUSER, G-LUC-XYZ]. (Note that, depending on the IDP, it might also possible for Alice to login using a smart card [G-LUC-AUTHSMARTCARD].)

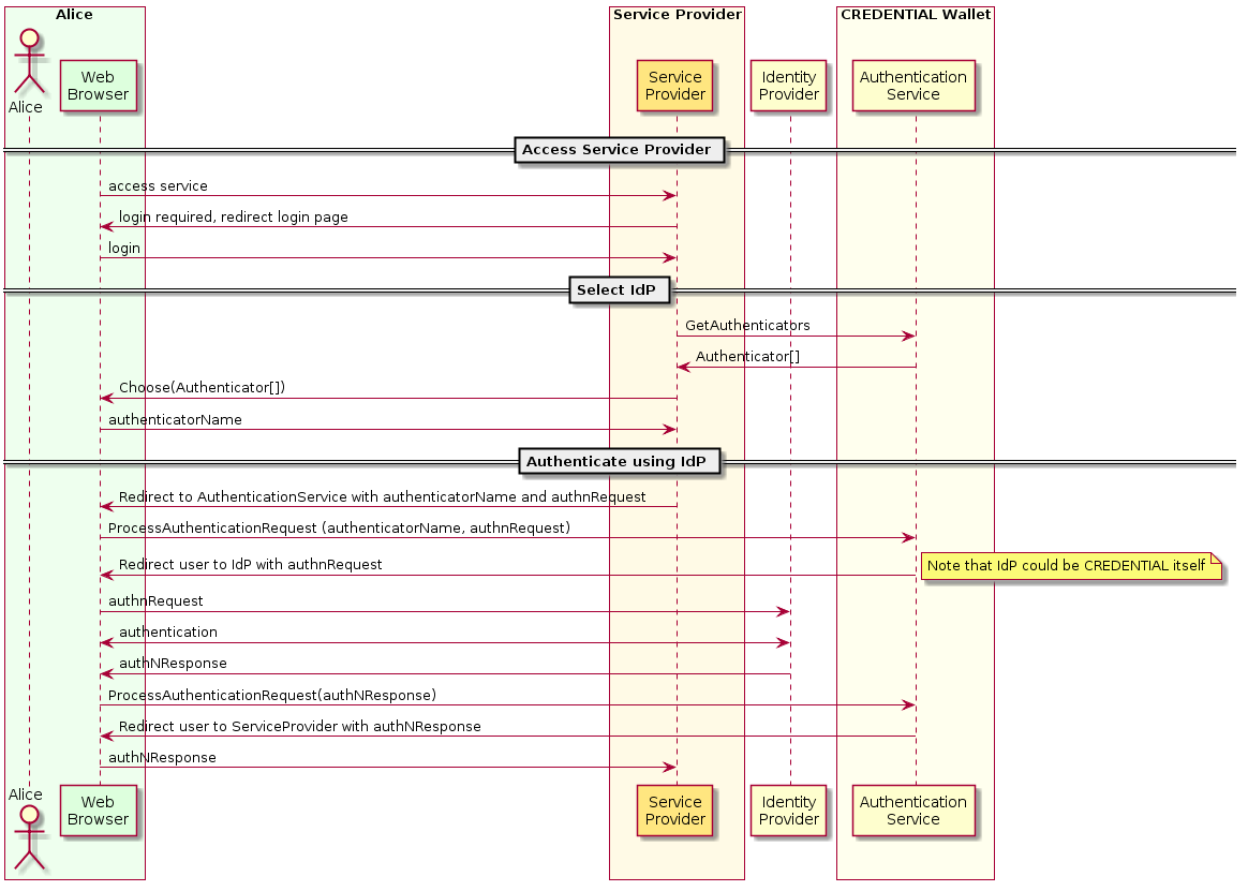


Figure 29: User Authenticating towards a SP Using a CREDENTIAL-accepted IDP

The service shows her that she, as a reward for her loyalty, receives a free premium subscription for a month, if she is willing to provide some additional information to the service. She is therefore shown a form, where the information received from CREDENTIAL is already pre-filled [G-LUC-FILLREGFORM], and Alice is asked to fill in some more fields, which she then submits to the provider [G-LUC-RENDERREGFORM, G-LUC-ADDATTRIBUTSREGFORM, G-LUC-SUBMITREGFORM]. Finally, Alice can now relax and watch her favorite TV show.

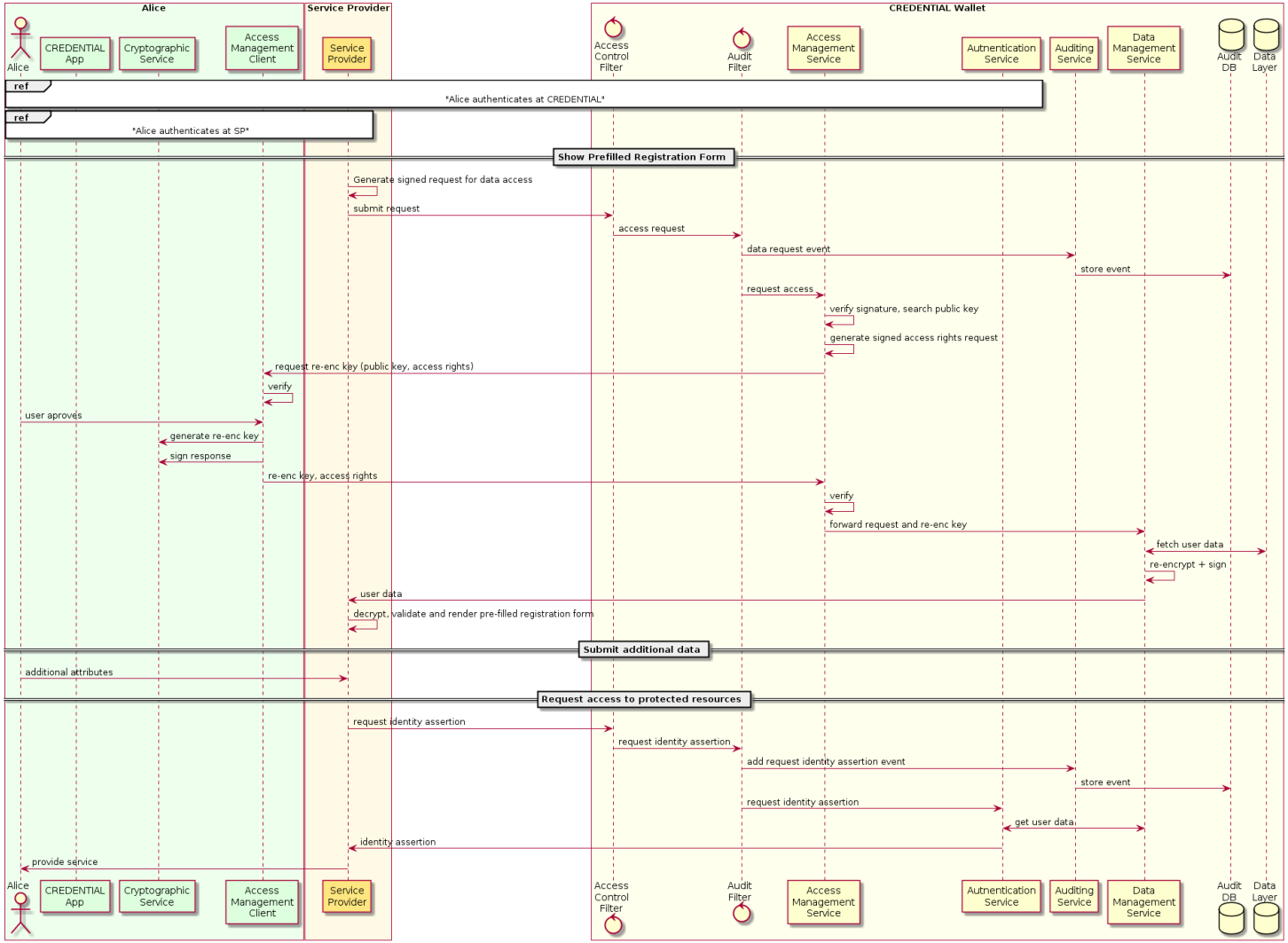


Figure 30: Form Filling with CREDENTIAL

Because she needs to get up for work early the next day, Alice finally decides to logout and continue the next day [G-LUC-CREATELOGOUTREQ, G-LUC-LOGOUTCREDWALLET (Alice's system), G-LUC-AUTHORIZATION, G-LUC-INVSESSION, G-LUC-AUDITING, G-LUC-RESPSUCCLOGOUT].

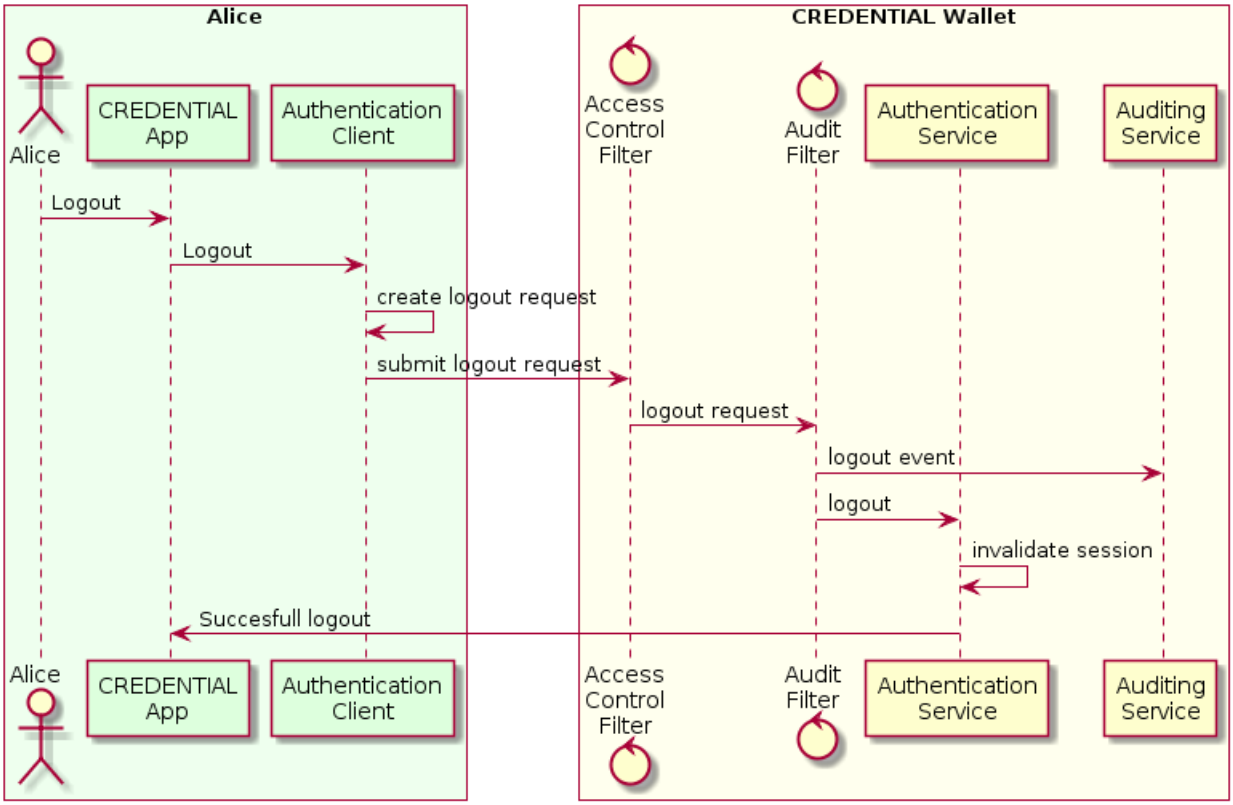


Figure 31: User Logout Process

7.3 Court-Order Requirement

A court-order requirement (e.g. because of sharing copyrighted data) was issued to ban one of Alice's account from using CREDENTIAL. Subsequently, Alice's account gets blocked [G-LUC-CREATEBANUSERREQ, G-LUC-SUBMITBANUSERREQ, G-LUC-AUTHORIZATION, G-LUC-SEARCHUSER, G-LUC-BANUSER]. Furthermore, Alice receives a notification [G-LUC-RECEXTTRIGEVENT, G-LUC-EVALNOTIFYCONF, G-LUC-SENDNOTIFY, G-LUC-PROCNOTIFY].

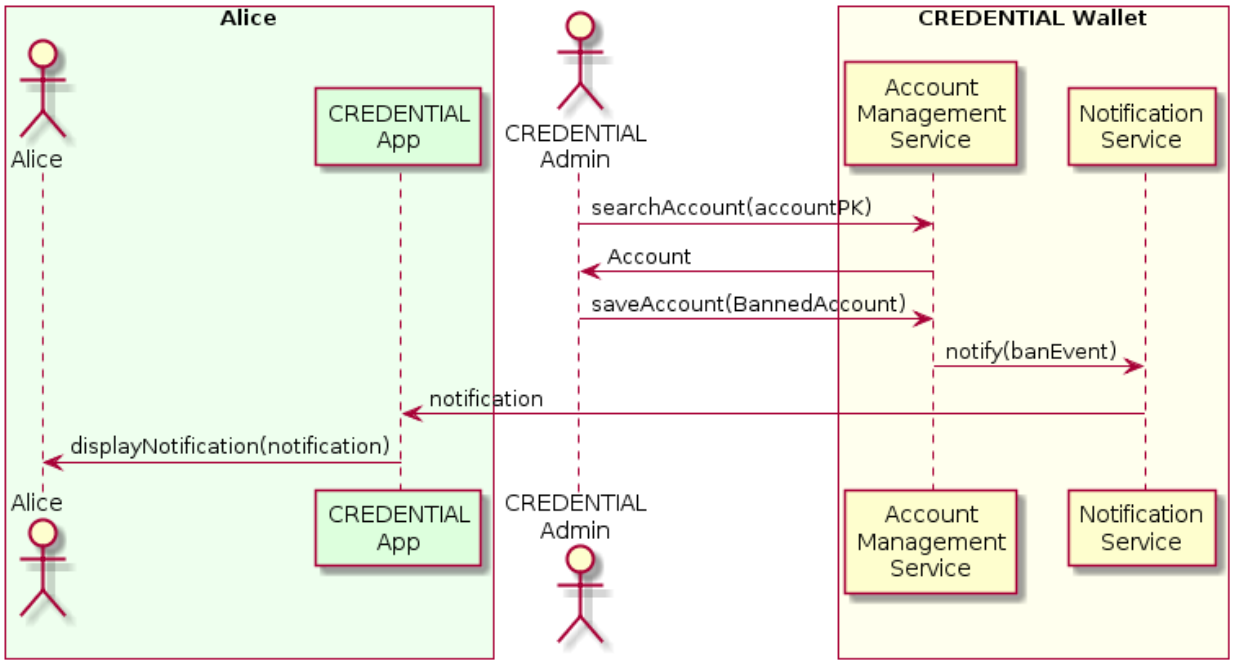


Figure 32: Banning an Account

Now, when Alice tries to login to her account, the access is denied [G-LUC-ACCESSDENIED]. After the court conflict is solved five months later, Alice is allowed to access her data again (e.g., since data within CREDENTIAL only gets deleted 6 months after notification) [G-LUC-SUBMITUNBANUSERREQ, G-LUC-AUTHORIZATION, G-LUC-SEARCHUSER, G-LUC-UNBANUSER, G-LUC-RECEXTTRIGEVENT, G-LUC-EVALNOTIFYCONF, G-LUC-SENDNOTIFY, G-LUC-PROCNOTIFY].

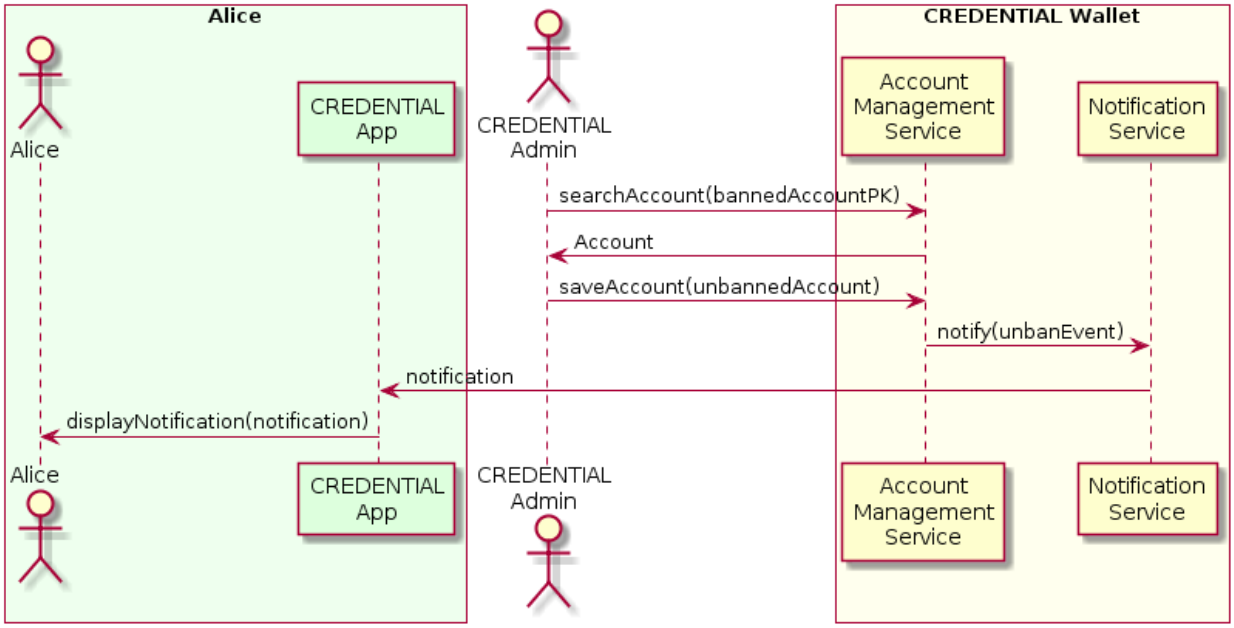


Figure 33: Unbanning an Account

7.4 Alice Wants to Check on her Shared Data

After some time of using CREDENTIAL, Alice forgot about what data she has uploaded to the Wallet since her registration to the CREDENTIAL service and to whom she is actually sharing those data. Alice therefore checks all previous logins to her account [G-LUC-CREATELISTPREVLOGREQ, G-LUC-SUBMITLISTPREVLOGREQUEST, G-LUC-QRYLISTPREVLOGINS, G-LUC-RETLISTPREVLOGINS] to validate that only she has used her account in the meantime. Furthermore, she checks which data is being accessed by other CREDENTIAL participants and when those accesses happened [G-LUC-CREATEVIEWACCESSESREQ, G-LUC-SUBMITVIEWACCESSESREQ, G-LUC-AUTHORIZATION, G-LUC-SEARCHACCESSES, G-LUC-AUDITING, G-LUC-RETLISTALLACCESSES].

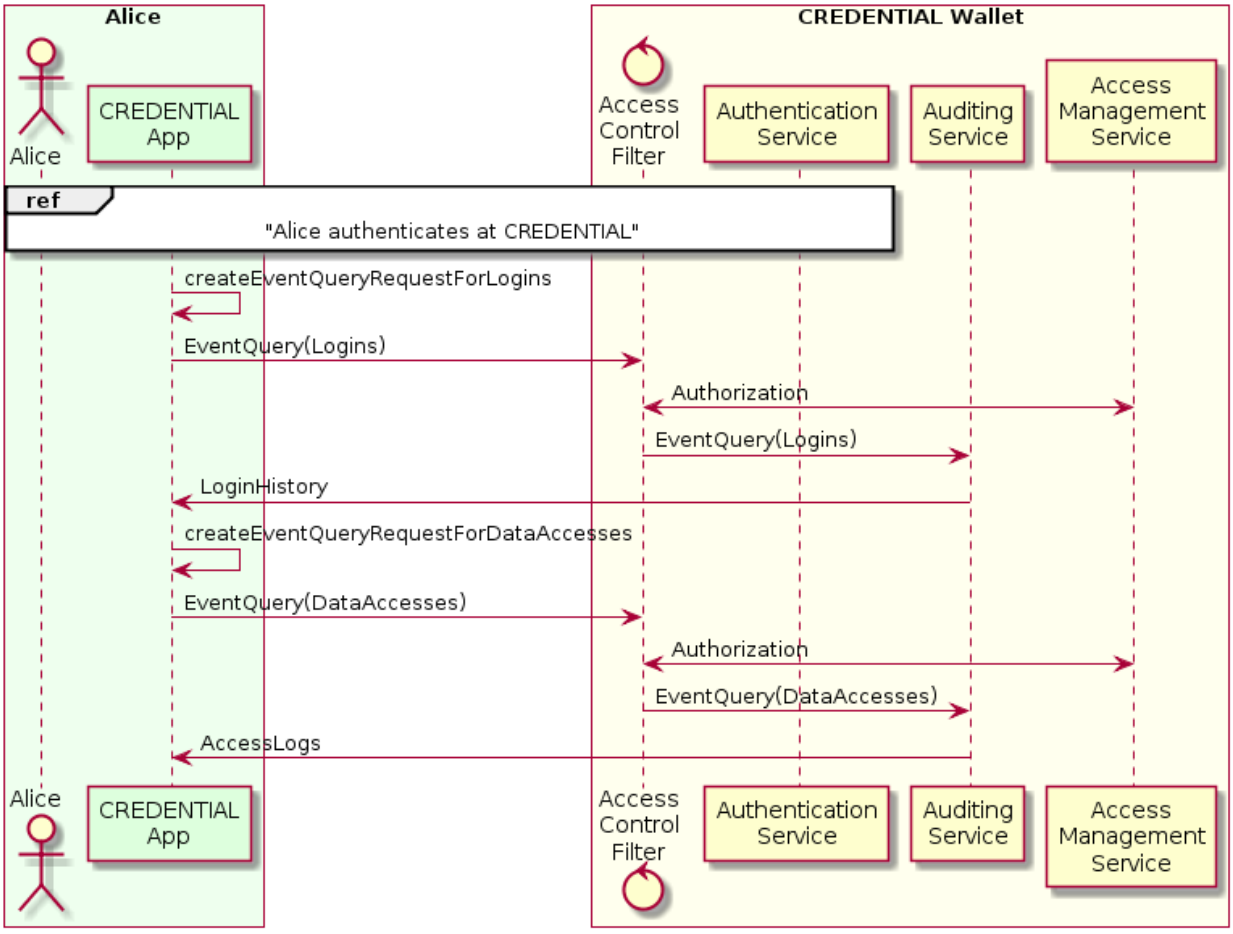


Figure 34: Checking History of Logins and Access Requests

7.5 Alice's Laptop is Stolen

An attacker stole Alice's smartphone and, hence, would not only have access to Alice's secret keys, but even worse, Alice's secret keys are destroyed. In this case, Alice would have to recover the data in her Wallet using the recovery key associated to a recovery account she exported to a USB stick when she signed up for CREDENTIAL [G-LUC-READRECOVERYPRIVKEY, G-LUC-CREATERECREQ, G-

LUC-SUBMITRECOVREQ, G-LUC-SEARCHUSER, G-LUC-VERIFYRECOVERYREQUEST, G-LUC-REENCDATA].

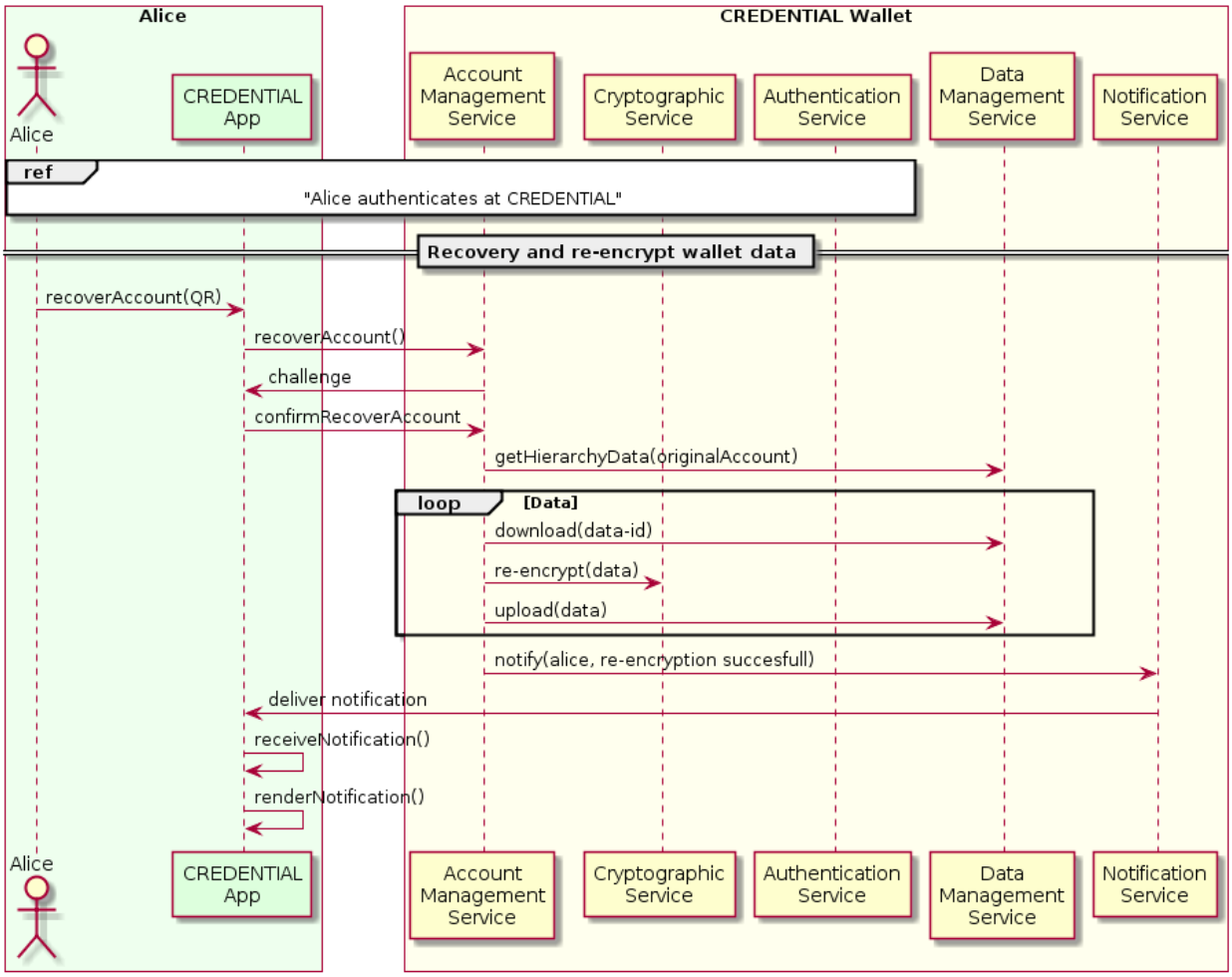


Figure 35: Recovery Process

This would then be followed by the re-generation of Alice's access rights [G-LUC-REQACCRIGHTSLIST, G-LUC-RECACCESSRIGHTSLIST, G-LUC-GRANTACCESSRIGHTS, G-LUC-PROVIDEACCESSRIGHTS] and the generation and secure storage of a new public and private key pair and its associated recovery key[G-LUC-GENKEYPAIR, G-LUC-ASSOCKEYPAIR, G-LUC-GENUPDREENCKEY, G-LUC-REQREK, G-LUC-VERIFYRECOVERYREQUEST]. The recovery mechanism is bound to, creating at a previous stage a recovery account and a re-encryption key for sharing data from the original to the recovery account. After the recovery, and in order to share again this data with other participants, depending on the final cryptographic scheme and its limitations, it may be the case that data has to be downloaded, decrypted, encrypted and uploaded again.

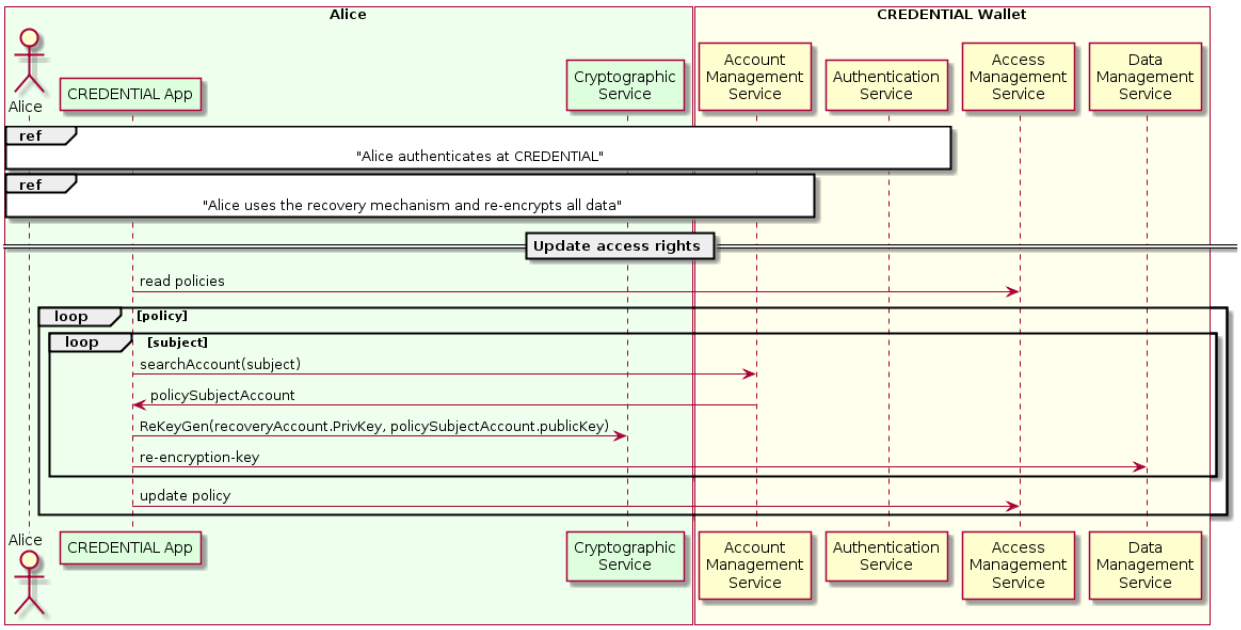


Figure 36: Updating Access Rights after Re-encryption

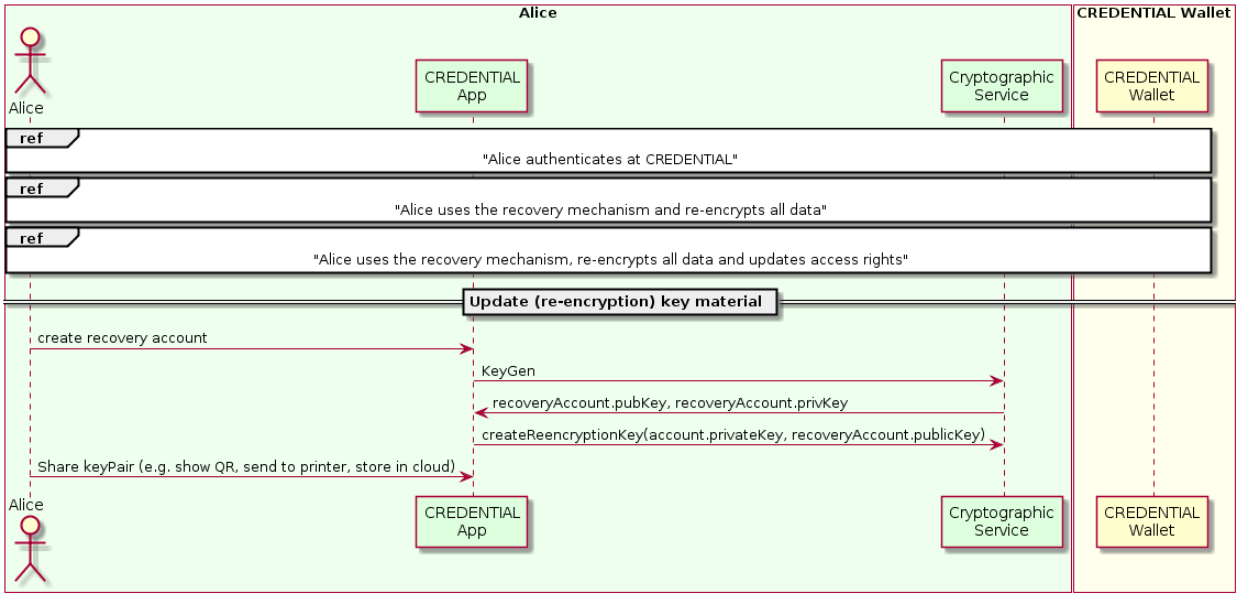


Figure 37: Update Cryptographic Material

7.6 Alice De-registers her Account

Alice wants to de-register her CREDENTIAL account. She uses the export function of CREDENTIAL to export her data from the Wallet to her CREDENTIAL software [G-LUC-CREATEEXPORTDATAREQ, G-LUC-SUBEXPDATAREQ, G-LUC-RETEXPDATA, G-LUC-RECDATA, G-LUC-DECDATA]. Next, she de-registers her account from the CREDENTIAL project [G-LUC-CREATEDEREGACCREQ, G-LUC-SUBMITDEREGACCREQ, G-LUC-DEREGACC].

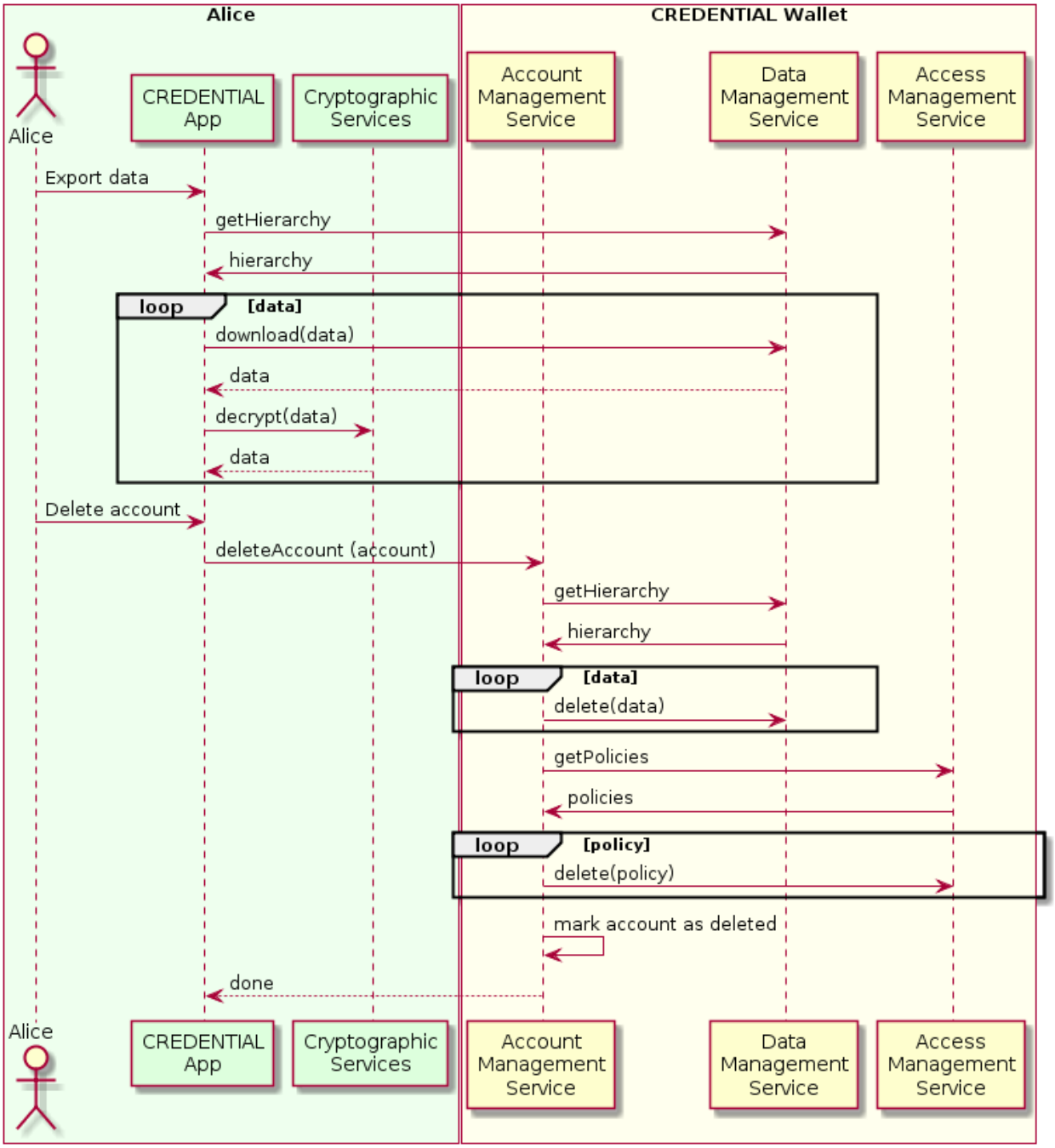


Figure 38: De-registering an Account

8 Physical Architecture

The main objective of the physical architecture section is to describe the physical nodes where CREDENTIAL technology/components will be deployed. This section will propose a reference physical architecture taking into account the principles presented in Section 2 but also general non-functional requirements such as availability, reliability (fault-tolerance), performance (throughput), and scalability. While this reference architecture is general and cloud platform independent, this section will also propose an operational environment supported by an open source solution for private and public clouds such as OpenStack that can be instantiated and adapted for pilot-specific needs.

Besides the development components specified in Section 6, there are additional "commodity" components that must be integrated in the generic physical architecture (e.g. network, physical storage, etc.). Being this the reference architecture, multiple instances/configurations of it are expected: some for development and testing, others for the deployment of the system for various sites or for different customers. Hence, the mapping of the development components to the physical nodes needs to be highly flexible and have a minimal impact on the scalability of the CREDENTIAL Wallet.

The CREDENTIAL Wallet's physical architecture takes into account the following concepts and aspects:

- The CREDENTIAL Wallet' underlying infrastructure will consist of physical, virtual, and automation components.
- The starting point for the CREDENTIAL Wallet – independently of pilot specific characteristics– is the physical data centre, control, and hardware.
- CREDENTIAL's reference architecture is required to achieve the correct level of resiliency and redundancy, including power and security aspects of the underlying infrastructure.
- The underlying platform is critical to consider as it is the foundation of the services that are built and delivered with high quality and reliability.
- The HW platform consists of physical servers to provide the underlying compute, memory, and local disk needed to support the infrastructure needs of cloud.
- Storage consists of a variety of different speeds and sizes of Serial Advanced Technology Attachment (SATA), Serial Attached SCSI (SAS), and solid-state (SSD) disks. These disks can be local to the storage infrastructure.

According to these aspects, we will deploy CREDENTIAL's Wallet main components (IAM and Data Services) in replicated virtual nodes that will support high availability requirements for CREDENTIAL Wallet's services. Beyond this additional redundancy, failover and load balancing mechanisms will be proposed to address high availability and avoid data loss in CREDENTIAL Wallet services. OTE's physical infrastructure, on which the generic Wallet and pilot-specific services will be deployed and run, will make available virtual machines with enough resources available. The data storage aspects are independent compared to the CREDENTIAL Wallet itself. Data storage follows a different way to be

reliable and high available, which is to take backup on a daily basis (even twice a day) to protect it in case of data failures.

The following figure shows an overview of OpenStack modules that can build VMs with the appropriate resources and the deployment setup of CREDENTIAL infrastructure. The OpenStack modules can be grouped or split in one or more physical nodes, depending on the requirements, use and load of the system(s) or service(s) to be hosted.

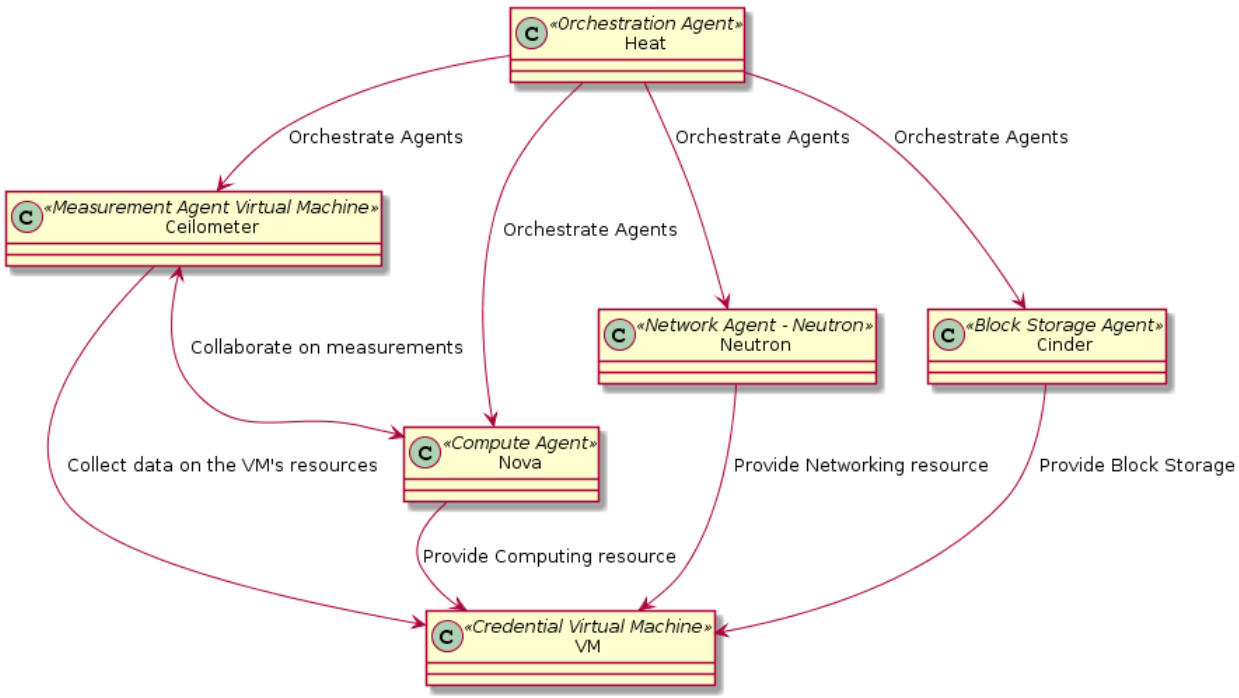


Figure 39: Cloud Provider Overview

Given the flexibility a cloud can have and give to a virtualized environment, it can facilitate both horizontal and vertical scaling. Horizontal scaling means that the scaling is performed by adding more machines into the available pool of resources whereas Vertical scaling or scaling up means that scaling is performed by adding more power (CPU, RAM) to an existing machine, noting that vertical scaling often involves downtime.

Regarding databases horizontal-scaling is often based on partitioning of the data i.e. each node contains only part of the data, in vertical-scaling the data resides on a single node and scaling is done through multi-core i.e. spreading the load between the CPU and RAM resources of that machine.

Additionally, cloud environments have developed mechanisms that can also apply auto-scaling either horizontally or vertically based on sets of rules. In the OpenStack platform this can be achieved with the utilization of HEAT and Ceilometer modules.

However, the CREDENTIAL architecture has the business logic and the data split in separate VMs (see Figure 40), which allows adding either more VMs or increase certain HW characteristics of a specific

VM, depending on the given load, number of active users, availability & security requirements etc. Given the fact that most of the functionality resides in the CREDENTIAL Wallet VM such as web server, authentication, authorization, certificate generation etc., it is expected that it may need to be more provisioned for CPU and memory, while the DB VMs more provisioned for HD space.

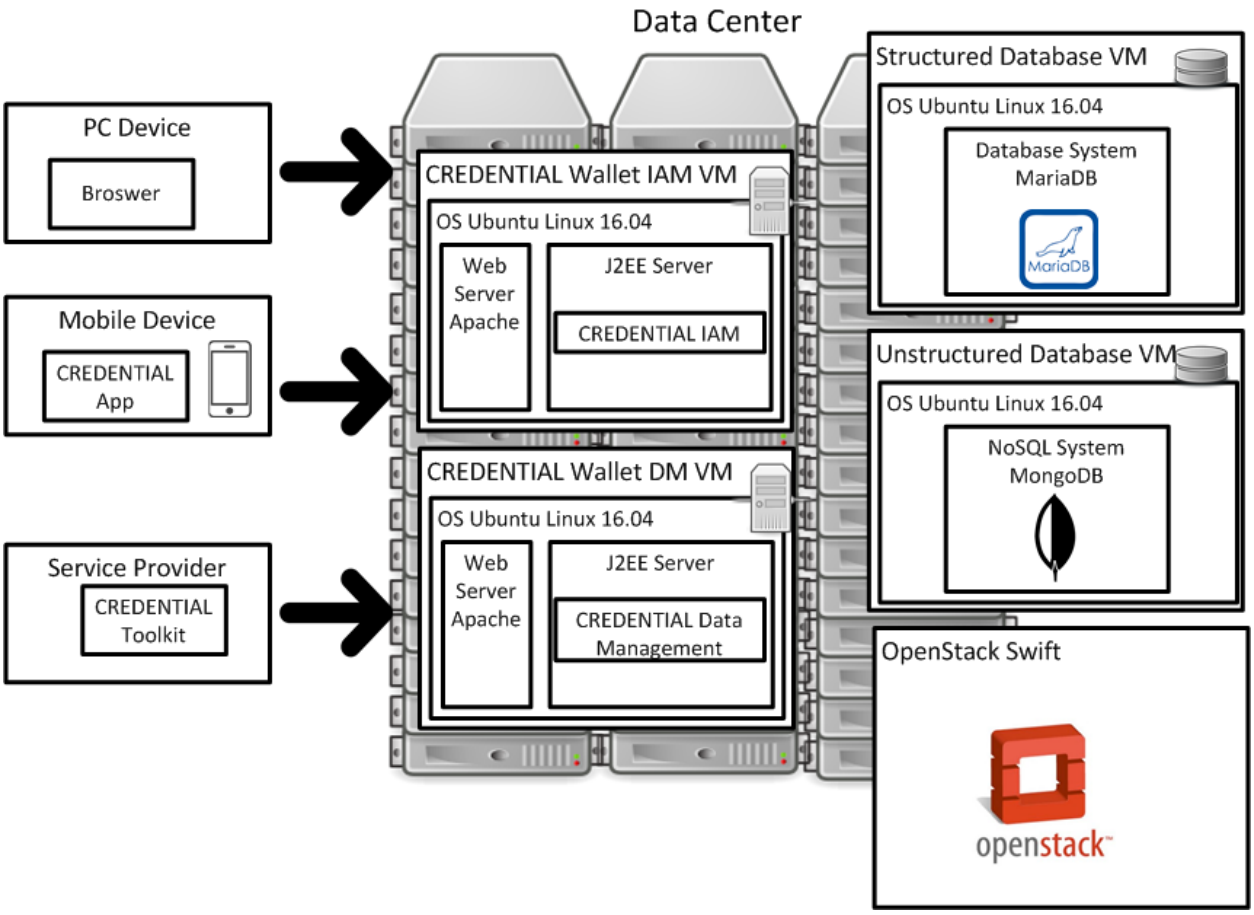


Figure 40: IAM and Data Management CREDENTIAL Components UML Diagram

Initially, given the requirements of the development teams and pilots as described in D6.1 a generic description for each node is

- PC Device & Browser (no specific requirements)
- Mobile device: minimum requirements Android 6.0
- Data centre node : CREDENTIAL Wallet VM, DATABASE VM, NOSQL VM

Regarding the Virtual Machines Software of the CREDENTIAL Wallet, this is drawn mostly from the OpenAM requirements, as they are found here[5]. It should be noted that this setup is not finalized or static and will be refined through the development and testing phases, as well as through the first and second pilot phases, where the requirements will be more refined and the performance of the whole system assessed more thoroughly. Apart from that, three environments will be setup:

- testing, which will be used for development and testing of the various components
- pre-production, where once a component has reached certain maturity and stability will be moved there and
- production environment, which will be used for the running of pilots

According to the above planning an example instantiation for pilots (e.g.) can be found below:

- Samsung S7 Mobile Phone
- Four CREDENTIAL VMs (this basic configuration will be fine-tuned according to the particular needs of the functionalities provided within the VM, e.g. Database vs IAM)
    - HW: 4CPUs, 4-8GBs of memory, 250-500GB HD
    - VM OS: Ubuntu 14.04 or 16.04 LTS
- 1 IAM VM
- 1 Data Management VM
- 1 NoSQL VM
- 1 MariaDB VM

[5] https://backstage.forgerock.com/docs/openam/13/release-notes/chap-before-you-install

9 Design Logbook

Whenever looking at the final design of a system it is not always clear what are the assumptions or rationale behind it. As a practice of transparency, aligned with PRIPARE's Privacy and Security by Design Methodology [17], a logbook section is included in this deliverable. In it, the most relevant architectural or design decisions are mentioned and accompanied with the assumptions and rationale behind them. These decisions are normally linked to the design principles, good practices, business objectives or specific requirements that drive the design. For the sake of clarity, decisions have been grouped according to its relationship with data management, account and identity management or with other horizontal aspects:

9.1 Data Management Related Decisions

9.1.1 NoSQL Databases

NoSQL databases originally refer to non-relational databases, meaning that they no longer rely on "traditional" tabular relations, which is the SQL standard. The main motivation for NoSQL approaches is to provide a higher performance and scalability (especially horizontally), which is usually at the expense of some other property (e.g. Atomicity, Consistency, Isolation, Durability). While performance and scalability are indeed a concern for the CREDENTIAL Wallet, one of the main reasons for choosing NoSQL for storing user managed data and metadata is its flexibility. While the tabular model constraints the applications and stored data to some scheme to be pre-defined for each application, a document-based NoSQL system, such as MongoDb[6] provides a higher level of flexibility, allowing applications to define their own document or metadata structure, which works as a database scheme.

9.1.2 Offline Access to Data

Within CREDENTIAL consortium it was discussed if offline access to data was to be supported and with what scope. It was proposed to allow offline versions of documents, tracking its usage and then synchronizing, at a later stage with the Wallet, so the data owner was aware of real usage of their data (even after its download to a participant IT system). While this indeed is a desirable function, it was finally considered out of the scope of the project. It is easy to understand that once the data is downloaded to a user device, the control over it is mostly fictional. Only DRM-like mechanisms are able to effectively keep track of this downloaded data. This offline tracking is not aligned with the original goal of the project or with the expertise of the Consortium. In any case, two actions have been proposed from the related discussions:

- Participants will be able to download data shared with them;
- Participants will be informed of the risk of sharing data and losing the effective control, once other participant decides to further share it.

[6] https://www.mongodb.com/

9.1.3 Client-side files / Data indexes

There was a discussion on the negative impact on privacy that would mean to have the CREDENTIAL Wallet (or any attacker with the access to the databases) to be able to link all data subject's document (even if encrypted). An alternative that was proposed to avoid this risk was to hold the link between users' documents on the client side, meaning, that just by looking at its own databases, the Wallet would not be capable of making the link between documents. The client applications would have to store an index of all the documents created by him. Additional, side-channel attacks such as observing access patterns, IPs of data subjects were also considered as a mean to link participants' documents. Finally, the costs of having clients managing these indexes added with the additional risks of users losing the index (implying losing all data) deemed the measure to be too extreme. An alternative, or a more balanced solution was agreed, the multi-account approach, which is described in next section (Section 9.3.1).

9.1.4 Data Structure

The data architecture that will govern the storage and management of data in the CREDENTIAL Wallet and presented in Section 6.1 has a very strong impact in the design decision. First, it drives the application of the "separation of concerns" (Section 3.1.4) design principle and the effective differentiation of data storage in two almost independent storage solutions: SQL-based for the storage of operational data and NoSQL-based for personal data (more flexible). Also, the concrete proposed structure allows not only protecting the data, but also the accompanying metadata (e.g. by encrypting access or modification dates)

9.2 Horizontal Decisions

9.2.1 Client-side Services or Compatible Applications

As the number of services or applications that could leverage the CREDENTIAL Wallet are multiple, it was necessary to understand how use-case or domain specific applications would interact with the Wallet or with CREDENTIAL Mobile Application. At first stage, it was suggested to use Android Services within the CREDENTIAL App to provide most of the Wallet functionality to third party apps. However, this raised the main concern that rogue or malicious applications could lure users to i.e. create a policy that provides full access to all users' data to a malicious participant. It was finally decided that all operations would be centralized through the Wallet and then approved within CREDENTIAL Application, which should be the only one with access to the account keys. Third party applications will have to request access to some particular data which will trigger pop up as a notification in the CREDENTIAL App, where the user may approve such access. For all means, third party applications will work as separate participants.

9.2.2 Notification Mechanism

Once presented the notification functionality, which is necessary to promptly inform participants of requests they need to attend or to transparently inform the user about accesses to their data, the Consortium discussed the most appropriate mechanisms to transport the notifications from the Wallet to the user (e.g. Google Cloud Messaging [15] or Pushy [16]). The usage of these public services raised some concerns as these parties could, from the content of the notification or its metadata (if the content was encrypted), somehow profile the users. The latest decision was to include a MQTT [14] compliant mechanism to which devices will subscribe to receive their related-account notifications. While the decision will imply a larger effort than integrating with mentioned public services it is in the benefit of the end-user privacy. Other alternatives that were studied were:

- Implement a proprietary mechanism which was discarded due to the envisioned effort
- Implement a notification broker that could use public multiple notification mechanism and would reduce the amount of data seen by each of the notification services.

9.2.3 Mobile-First Approach

In order to devote most of the project's effort to develop innovative aspects of the Wallet, the consortium has agreed to focus on the implementation of a mobile application that would serve as the principal mean to leverage and interact with the CREDENTIAL Wallet (authentication, authorization, notifications,,,. This means that no desktop-version of the CREDENTIAL Application is in the scope of the project. However, use case specific applications (i.e. e-Health) may be in the scope of the pilots. As mentioned in Section 9.3.2 this application will be considered to all effects as a separate participant, being necessary to request or to establish authorization policies that will allow applications to access data stored in the Wallet.

9.3 Account and Identity Management Related Decisions

9.3.1 Multiple Accounts

As a solution to avoid the "linkability" of documents from the same data subject, the concept of multiple accounts was introduced. One data subject should be capable of creating more than one CREDENTIAL account so he is not forced to accept the linkability risk of sharing all his information under one account would create. This mechanism should be as usable and as transparent to the user as possible, meaning that the complexity of the multiple account approach should be hidden by the application. The unlinkability provided by the Wallet would be twofold:

- Avoid the CREDENTIAL Wallet to store links between documents that the data subject wishes to keep unlinked
- Avoid colluding service providers to link a user across different services (e.g. using pseudonyms).

While it is acknowledged that side-channel attacks could lead to linking multiple accounts, the assumption for this case is that the attacker will only have access to the stored data and metadata (not the access logs). The way of preventing these side-channel attacks would be to integrate onion-routing (or similar) technologies. CREDENTIAL will recommend the usage of these onion-routing technologies to users and will even support it in the mobile application (if feasible according to scope and time).

The Consortium also considered the complexity of handling multiple keys for multiple accounts and agreed to explore, depending on the selected cryptographic schemes, "Deterministic Key Generation" functionalities that would reduce the complexity to the management of one unique master key.

9.3.2 Key Recovery Mechanism

The key recovery mechanism devised for CREDENTIAL is innovative in two aspects. First, it will study the application of deterministic key generation principles to proxy re-encryption schemes. The usage of a deterministic key generation algorithm would reduce the "necessary" cryptographic material for managing any number of CREDENTIAL accounts to one unique key pair. Any number of account-specific key pairs may be generated from this unique key pair and some string (e.g. account name) and the application of such deterministic algorithm. This would significantly reduce the risks and usability of managing a large

number of keys necessary when dealing with number of accounts managed by the same user (see Section 9.3.1)

Second, while backing up and restoring keys is a necessary mechanism, it is also important to handle key compromises. If no specific mechanism was planned, whenever a user detects a key compromise it would have to download all the data, decrypt it, encrypt it for a new account linked to a different key pair and upload it again. However, by allowing the creation of a recovery account and the generation of an associate re-encryption key enable the system just to re-encrypt the data, without the need of uploading large amounts of data. The recovery account functionality may be constrained by the chosen proxy re-encryption scheme, if it does not support a "multi-hop" feature, in any case, all data will have to be downloaded, decrypted, encrypted and uploaded again.

9.3.3 OpenAM

One of the most impacting decisions that the consortium has agreed is to avoid starting from scratch to build a complete IAM system. The agreement was to rely on existing, open-source, and with an acceptable licensing scheme to build the improvements proposed by CREDENTIAL while fulfilling all requirements proposed in the scenarios. Once the agreement was reached, two main projects were proposed and assessed:

- OpenAM[7]
- Gluu[8]

A small assessment was performed by CREDENTIAL partners and shared for a final decision:

	OpenAM	Gluu
Pros	Supports state of the art protocols: • UMA • OpenID • LDAP • OAuth • SAML Extensive documentation Expertise within the consortium Java-based (aligned with other components of the architecture) Active community	Supports state of the art protocols: • UMA • OpenID • LDAP • OAuth • SAML Modular and less complex Active community
Cons	More complex system	Lack of previous experience Jython and dynamic code

[7] https://forgerock.org/openam/
[8] https://www.gluu.org/

Table 4: OpenAM vs Gluu Assessment

It was finally agreed that, given that both projects would fulfil almost the same needs of CREDENTAL Wallet, it was preferable to choose OpenAM, as partners have some direct experience in working with it and with its performance. The following step is to map the development components, which were described in a generic way (with no direct relation to any particular technology) in Section 6 to OpenAM architectural components, identifying what are the functions already implemented and what are the developments to be done during the next steps of the project. This component by component assessment has been performed in Appendix A – OpenAM Instantiation.

By leveraging OpenAM, the consortium will be able to set up a functional integration of service providers and the CREDENTIAL Wallet (with basic functionality) at early stages of the development phase, and from then, step-by-step increment the functionalities provided by the Wallet, using a continuous delivery approach.

10 Conclusions

This deliverable follows a well-established methodology to provide a complete design of CREDENTIAL Wallet and the necessary components to interact with it. The process started taking as an input complete description of the scenarios and use cases which allowed identifying the generic (abstraction of scenario-specific) components from a logical perspective. Taking this logical perspective as a starting point and following system-design best practices and principles, development and deployment blueprints where created. These blueprints are key to the success of CREDENTIAL as they will serve as the main roadmap for system developers to build and pilot the CREDENTIAL Wallet.

At the beginning of the design, it was agreed among the system designers to follow a process that would serve as a design validation step. This step was defined as to create a mapping between all the logical use cases and the different development components that would have to support them. If a use case was not mapped, it would mean that it was probably not being considered. If a component could not be mapped to a use case, then it would mean that it cannot be justified due to the use cases described, and a specific justification would be necessary. The results of this mapping are present in Appendix C – Mapping between Use Cases and Development Components and show how that all use cases are covered and how all development components are justified. The only relevant point of this mapping is that it shows how the consortium decided to focus in "local" digital signatures through the cryptographic service, part of the Participant Toolkit rather than in the remote signature functionality, which would be in CREDENTIAL's Wallet roadmap for after the project duration.

There are still some decisions to be made at a research and development level, i.e. selection of adequate cryptographic implementation or the appropriate crypto scheme, but the most crucial ones have been taken in the context of the design phase. While the development architecture section started as an attempt to completely specify the components and subcomponents as they would be developed in the project, it was finally agreed as more relevant to provide a generic design architecture (technologically independent) and to complement it with a technologic-specific one. The technological-specific design (Appendix A – OpenAM Instantiation) presents all the generic development components mapped to OpenAM, translating the specifications of generic components from Section 6 to concrete implementations or approaches for OpenAM, following its architecture, plugin mechanisms and extension points. The appendix also demonstrates how the decision has a truly positive impact in the project, as it presents some of the generic development components as provided by OpenAM out of the box, allowing the CREDENTIAL consortium to focus on the relevant innovations of CREDENTIAL: proxy re-encryption, malleable signatures, strong authentication and improvement of protocols and miss no time in starting to implement from scratch a complete IAM

Other important design decisions have been made and reflected in this deliverable. These decisions do not only impact the required development effort, but most important, if positively affects the level of privacy that can be provided towards data subjects (e.g. multiple accounts, or the notification system).

References

[1] Philippe Kruchten, Architectural Blueprints—The "4+1" View Model of Software Architecture, IEEE Software 12 (6), November 1995, pp. 42-50. https://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf

[2] Grady Booch , Object-Oriented Analysis and Design with Applications, 2nd edition.

[3] FCGSS, Applying 4+1 View Architecture with UML 2, http://www.sparxsystems.com/downloads/whitepapers/FCGSS_US_WP_Applying_4+1_w_UML2.pdf

[4] Vogel, O.; Arnold, I.; Chughtai, A. & Kehrer, T., Software Architecture - A Comprehensive Framework and Guide for Practitioners. , Springer (2011)

[5] Robert C. Martin, Design Principles and Design Patterns , objectmentor.com, retrieved 20016-04-26 (http://mil-oss.org/resources/objectmentor_design-principles-and-design-patterns.pdf)

[6] European Commission, HORIZON 2020 - WORK PROGRAMME 2016-2017, General Appendixes, Appendix G - Technology readiness levels (TRL), http://ec.europa.eu/research/participants/data/ref/h2020/other/wp/2016_2017/Appendixes/h2020-wp1617-Appendix-g-trl_en.pdf

[7] NASA, Technology Readiness Level Definitions, http://www.nasa.gov/pdf/458490main_TRL_Definitions.pdf

[8] Cavoukian, A., 2006. Privacy by design: The 7 foundational principles. implementation and mapping of fair information practices. *en ligne: https://www. privacyassociation. org/media/presentations/11Summit/RealitiesHO1. pdf*.

[9] Gürses, S., Troncoso, C. and Diaz, C., 2011. Engineering privacy by design. *Computers, Privacy & Data Protection*, *14*, p.3.

[10] A. Cavoukian, "Privacy by design: Origins, meaning, and prospects," Privacy Protection Measures and Technologies in Business Organizations: Aspects and Standards: Aspects and Standards, vol. 170, 2011.

[11] The United Nations. Universal Declaration Of Human Rights. 1948, Paris. http://www.un.org/en/universal-declaration-human-rights/index.html

[12] The European Parliament and the Council of the European Union. Regulation (EU) 2016/679 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

[13] Carliss Y. Baldwin and Kim B. Clark. 1999. *Design Rules: The Power of Modularity Volume 1*. MIT Press, Cambridge, MA, USA.

[14] MQTT Version 3.1.1. specification http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html

[15] Google Cloud Messaging: https://developers.google.com/cloud-messaging/gcm

[16] Pushy https://pushy.me/

[17] PRIPARE Privacy- and Security-by-Design Methodology Handbook, December 2015. http://pripareproject.eu/wp-content/uploads/2013/11/PRIPARE-Methodology-Handbook-Final-Feb-24-2016.pdf

11 Appendix A – OpenAM Instantiation

As explained in the design logbook (see Section 9) the CREDENTIAL Consortium agreed to build its Wallet leveraging existing open source IAM systems in order to minimize development efforts in non-innovative areas, to maximize the scope of the resulting system (e.g. open source IAM systems are already compatible with a number of authenticators which could not be developed in the scope of CREDENTIAL). The final decision was to base the Wallet developments in OpenAM, as partners of the consortium already have a positive experience with its usage in other research projects and it provides the same functionality that alternative projects also assessed.

11.1 OpenAM

OpenAM is an open source Identity and Access Management and federation server platform which was originated from Sun Microsystem's OpenSSO system which was forked after Oracle's purchase of Sun.

The complete list of OpenAM features, as published by its sponsor (ForgeRock) can be found online[9]. Next, we highlight those features that are most relevant for the CREDENTIAL Wallet:

- The modularity of its architecture will allow partners to work independently on different parts of the platform, according to their expertise and expected contributions;
- Multiple authentication modules and the option of add custom ones (e.g. FIDO UAF);
- Adaptive Risk Authentication that can be used to beter balance usability and security during authenticatio;
- OpenAM as a User-Managed Access (UMA) Provider for extensive privacy and consent capabilities;
- The federation service that will allow to interconnect with multiple identity, attribute and service providers;
- Cross-domain SSO capabilities matches the requirements coming from scenarios and minimizes friction with users;
- User self-privisoning features are required by the CREDENTIAL Wallet as the enrolment and identity lifecycles are directly managed by participants;
- The usage of open standards such as OpenID, SAML will provide out-of-the-box compatibility with a large number of identity and service providers;
- The existence of REST APIs for services such as user self-service, policy, and security token service will support the decoupling between the applications and the Wallet;

As it can be seen in Figure 41, OpenAM system relies in a plugin system which facilitates developer's work in terms of the extension of the platform.

[9] https://forgerock.org/openam/#openam-feature-overview

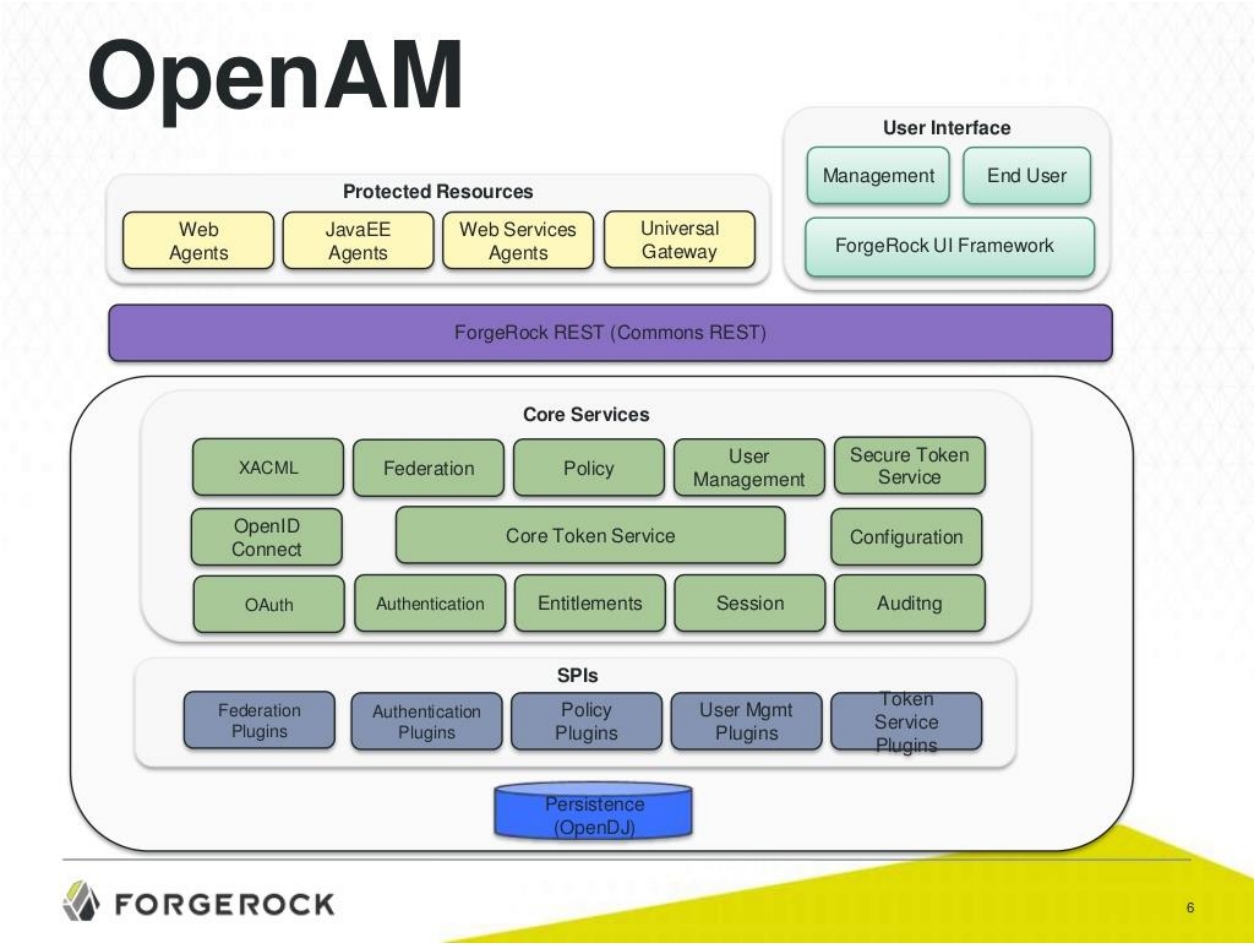


Figure 41: OpenAM Architecture[10]

11.2 General Instantiation

OpenAM will play a two-fold role in the CREDENTIAL Wallet system. On one hand, it will act as an Identity and Attribute Provider towards service providers, providing re-encrypted version of the attributes through prevalent protocol extensions or modifications. On the other hand, it will act as an access management solution that will govern accesses to personal data stored in the data service, effectively applying sharing policies according to user preferences. OpenAM will have to be extended with several components to fulfil the platform and scenarios' requirements, integrate research & innovation results and to maintain the alignment with OpenAM's architecture and principles.

[10] Source: ForgeRock https://www.slideshare.net/ForgeRock/ois-architecture-review

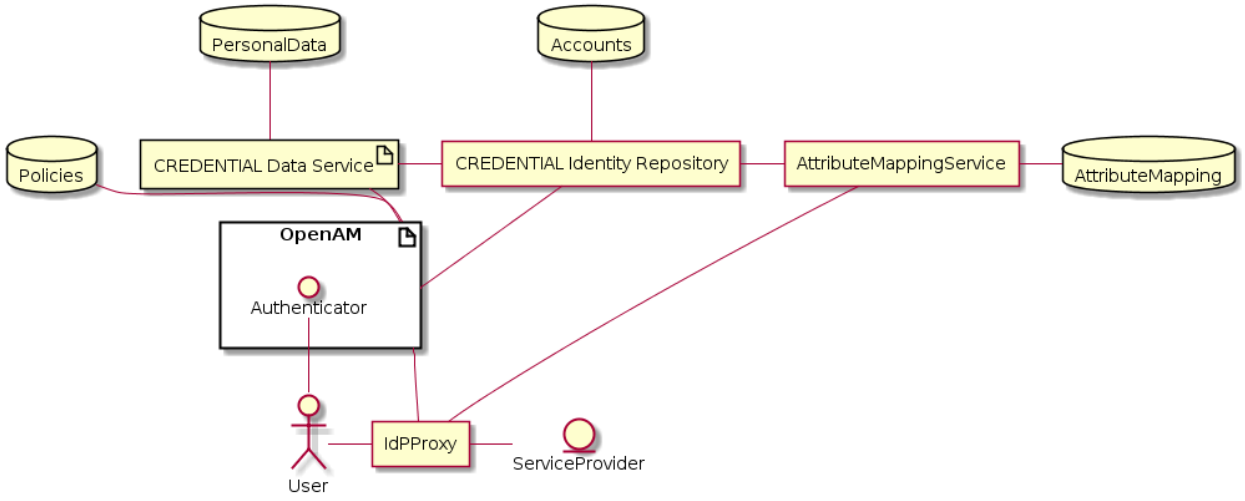


Figure 42: OpenAM Instantiation

The most promising approach is to create a CREDENTIAL Identity Repository, implementing OpenAM's IDRepo interface that will interface with the CREDENTIAL Data Service to fulfill identity or attribute requests. For such thing, the user will have to establish a service-provider specific mapping between requested attributes and data stored in the data service. An IDProxy is planned in the case that specific processing is required in-between OpenAM and the service provider (e.g. interact with the user to establish the attribute mapping). Figure 43 and Figure 44 show how the attribute mapping service could work.

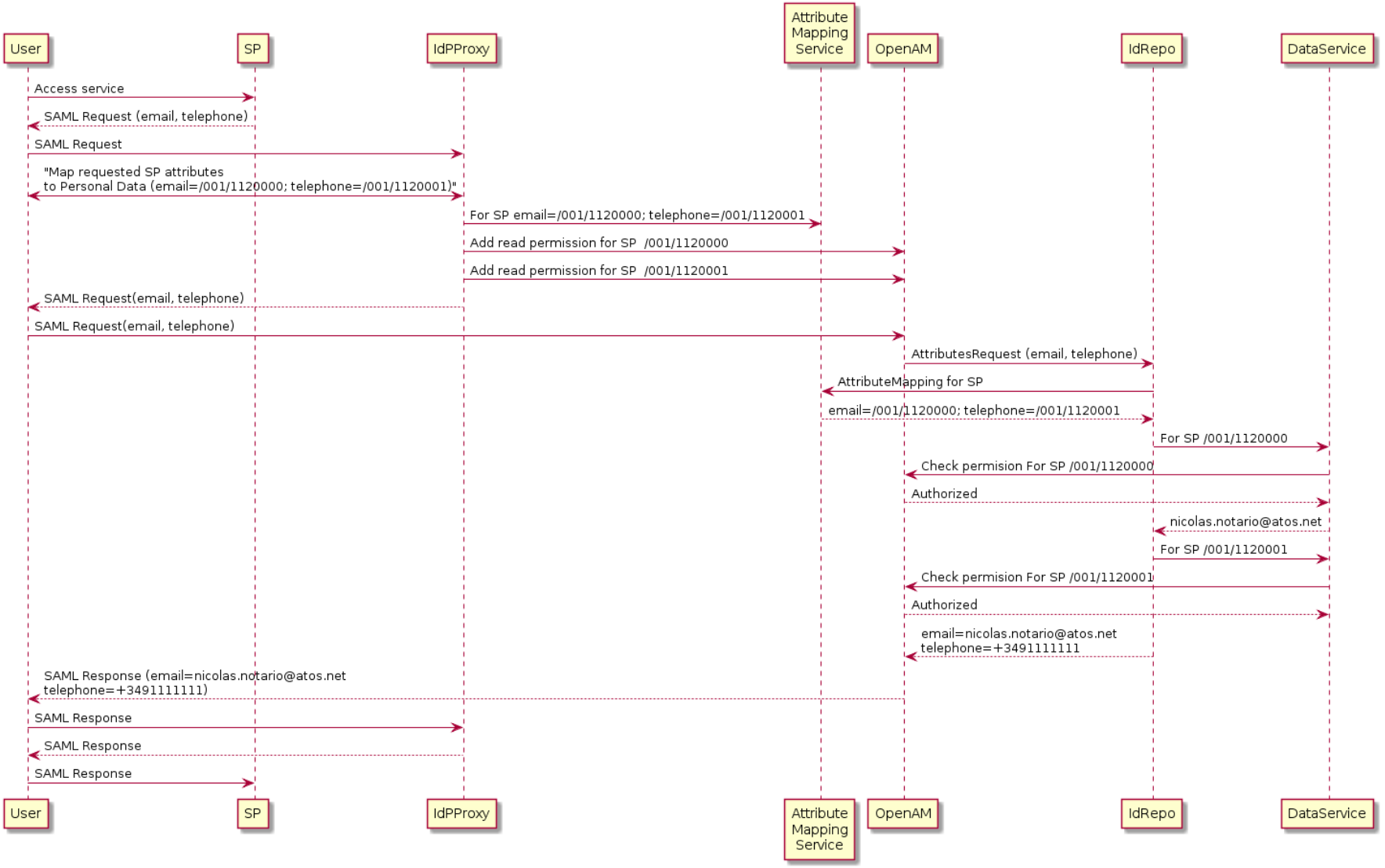


Figure 43: How the Mapping Service could Work (1/2)

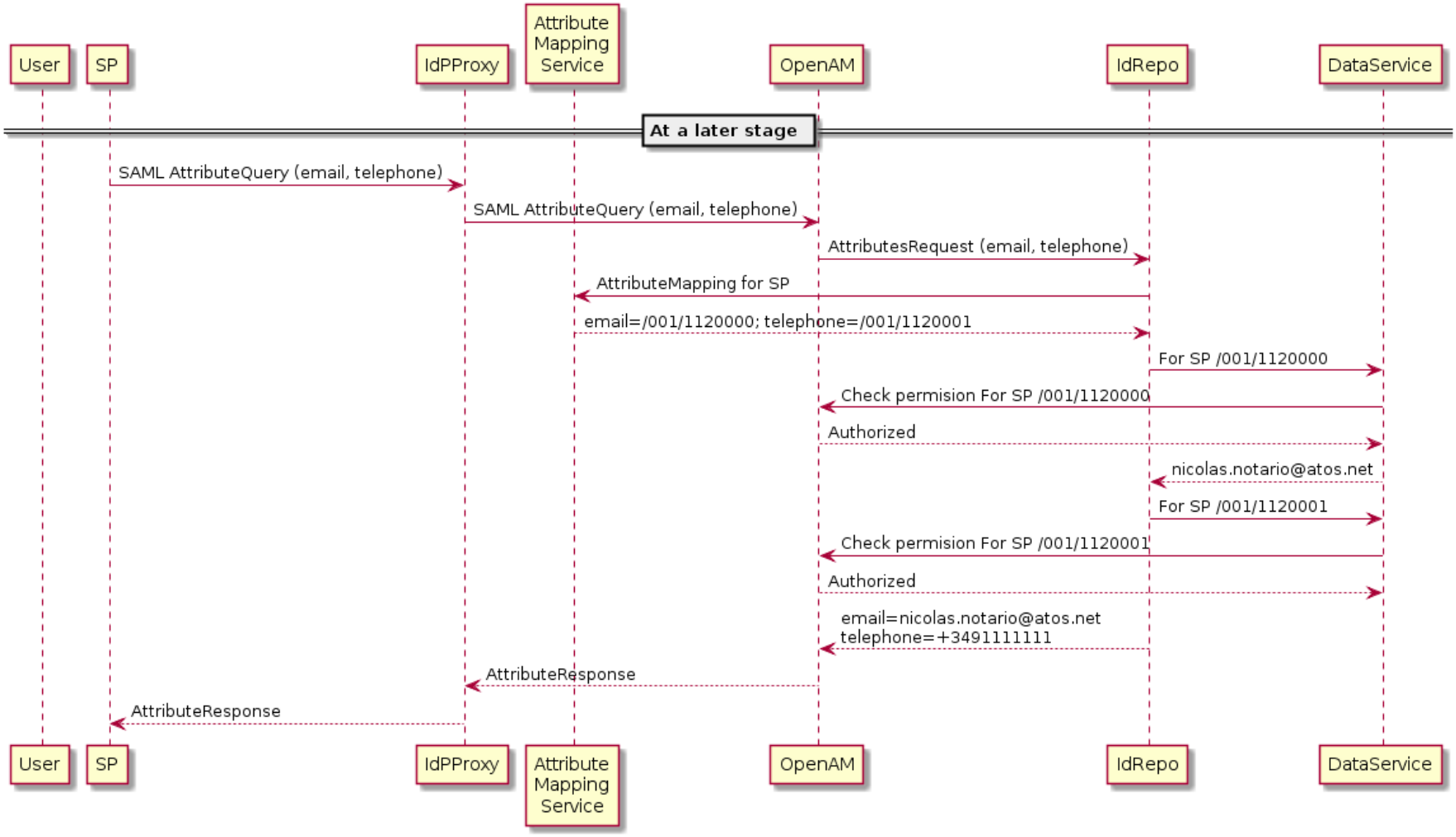


Figure 44: How the Mapping Service could Work (2/2)

Following, for each of the components described in the Development Architecture view (Section 6) are re-defined (if necessary) in the scope of OpenAM and its architecture and interfaces. In other cases, it is described how the originally planned component will interact with OpenAM's architecture.

11.3 Component by Component

11.3.1 Authentication Service

The Authentication Service development component described in 6.2.2 addressed three aspects of the authentication:

- Authenticating users towards OpenAM matching credentials provided by the users with ones stored within OpenAM
- Authenticating users towards other service providers, acting as an IDP and answering to identity requests through protocols such as SAML.
- Authenticating users towards OpenAM or towards other service providers relying on third party identity providers (e.g. eIDAS) for the user authentication

These three concepts are separated in OpenAM- The Authenticating users towards OpenAM functionality can be mostly matched to the Authentication module identified in the OpenAM's architecture coupled with the authentication plugins

11.3.1.1 OpenAM Authentication Plugins

OpenAM already provides, out of the box, 24 authentication modules[11] ranging from the classic user/password stored in LDAP or a User Data Store to more complex OAuth 2.0 plugins that can be used to satisfy the authentication from third party providers use case. According to the requirements and the needs of the Wallet, the list of authentication modules will be extended with two custom modules:

- A **generic Challenge Response Authentication Mechanism (CRAM) authentication plugin**. This plugin will use cryptographic material created through the cryptographic services and linked to accounts to ensure that the user is in control of the private keys. This will be ensured by sending a one-time challenge that will guarantee that no reply attacks can be exercised.
- In order to cope with the requirements of strong authentication, a **FIDO UAF plugin** will be developed, based on open source U2F[12] and UAF[13][14] implementations

It is mostly sure, looking at the actual implementation of the FIDO U2F OpenAM authentication plugin, that some **modifications will be required in the attribute storage scheme** in order to accommodate the information that is necessary to link accounts to cryptographic material.

[11] https://wikis.forgerock.org/confluence/display/openam/OpenAM+out+of+the+box+authentication+modules.
[12] https://github.com/arietimmerman/OpenAM-U2F
[13] https://github.com/eBay/UAF
[14] https://github.com/zsavvas/ReCRED_FIDO_UAF_OIDC

In order to fulfil the requirements regarding the usage of external IDPs to authenticate towards the Wallet, there are already several authentication modules that will support this:

- OAuth 2.0
- OpenID Connect
- Federation (in conjunction with OpenAM Federation component)

The plugins will be configured according to the piloting scenarios. For the specific purpose of fulfilling the requirements to integrate with STORK 2/eIDAS infrastructure (or other external SAML IDPs), the Federation module, along with the Federation authentication and possibly an IDP Proxy configuration will be configured and used. No significant development is expected for this purpose.

During the development and instantiation phase, the utility and convenience of enabling and configuring such other modules such will be considered, especially:

- Adaptive Risk Based Authentication module, this module measures the risk associated during an authentication event and based on a score it can trigger the use of an additional authentication module to request for additional credentials.
- Push Authentication module15: which provides a convenient mobile-based 2FA by using third party notification services and a mobile specific application and without relying in vulnerable SMS tokens.

11.3.1.2 OpenAM as Identity Provider

OpenAM supports, out of the box, the Identity Provider role for third party services using both protocols SAML and OpenID. It is just a matter of configuration. However, **improvements on the underlying SAML and OpenID protocols, are devised in CREDENTIAL research tasks, will be implemented in order to support the encryption schemes related to proxy re-encryption.**

Normal usage of Identity Provisioning protocols in OpenAM mainly involves the transference of attributes stored in OpenAM's IdRepo after a mapping between requested attributes and the internal attribute scheme (e.g. the service provider requests a "telephone" attribute but internally OpenAM is configured to distinguish between "WorkTelephone" and Mobile. At a configuration level system's administrator must do this mapping that will be applied for the attribute transference between OpenAM and the service provider.

However, and in the case of the CREDENTIAL Wallet, where the protected personal data structure is defined by the user (not by the Wallet) there is a dynamic layer that must be applied to the concept. E.g. the service provider requires a telephone and the user decides that, for a specific service provider, he wants to share his personal mobile phone which is stored under a UUID in the data management service. The complexity of this dynamic layer will be addressed with a new **Attribute Mapping Service**. Requests from the service provider to the Wallet will have to be intercepted (through the IDPProxy) in order to

[15] https://wikis.forgerock.org/confluence/display/openam/OpenAM+out+of+the+box+authentication+modules.

allow the user to select the attribute mapping it desires to use for the ongoing interaction. The user may choose to re-use a previous mapping or to specify a new one, linking each of the requested attributes to one of the users' document values.

11.3.2 Account Management Service

The account management service component is responsible for managing the different aspects of the account lifecycle: creation, management, and deletion. For the specific case of CREDENTIAL, it is also necessary to consider key updating processes which will be triggered once the original account-specific key-pairs are compromised or lost.

11.3.2.1 Account Registration Process

Due to the anonymous-by-default nature of the CREDENTIAL Wallet, it will not be necessary to provide any user-specific attribute for creating an account, and accounts will be created by the users themselves, meaning that self-registration will be the norm.

In order to link OpenAM's self-registration capabilities with the CREDENTIAL Wallet approach, the self-registration process has to be univocally linked to the account's key pair. This can be done by implementing a Challenge Response Authentication Mechanism and integrating it in OpenAM's self-service mechanisms[16]. Similar to the classical self-registration approaches where users need to click a link sent by email or they have to fill a captcha, new accounts have to be created for a specific public key. For this, a **custom plugin for the self-registration process will be required** (similar to UserRegistrationConfigProvider[17]). Such plugin will issue a unique challenge for each self-registration request. The user application will have to follow the next step by providing the public key of the account and the challenge signed with the private key. This public key or a hash of such key will be linked to created account and the corresponding private key will be the basic mean to unlock the account until additional authentication mechanisms are enabled (e.g. FIDO or link with external IDPs). This additional account attribute will be added using the mechanism described in Developer Guide[18], Section 4.1.

11.3.2.2 Account Management

OpenAM already includes a REST API covering all the functionality related to managing the account attributes. Public attributes, meaning all attributes that will be related to an account and stored in plain (not encrypted) will be decided for each of the scenarios. These attributes, which in no case will be mandatory, can be used by service provider participants to facilitate their localization by other participants. These attributes will be handled in the same way as the public key attributes related to the account registration and authentication process. The attributes may be updated using OpenAM's self-service REST API [19] (Section 2.1.5.1.3).

---

[16] https://github.com/OpenRock/OpenAM/tree/5d4589530d1353fdd627ab216a1cdcbcaf6b705e/openam-selfservice
[17] https://framagit.org/teddyber/openam/blob/f36d7fc719bcbc9aba272b40a4723a24a81d6e0c/openam-selfservice/src/main/java/org/forgerock/openam/selfservice/config/flows/UserRegistrationConfigProvider.java
[18] https://backstage.forgerock.com/cp/portal/cloudstorage/pdf-1771818580?redirect=true
[19] https://backstage.forgerock.com/cp/portal/cloudstorage/pdf-1771818580?redirect=true

For all other attributes that have to be protected the idea is to store them encrypted through the data management service and **to create a CREDENTIAL-specific OpenAM IdRepo compliant-class (CREDENTIAL Identity Repository)** that can be based in DatabaseRepo class[20] overwriting the get and setAttributes methods so attributes are no longer stored in the "local" database but in the Data Management Service. The exact personal data to be retrieved from the Data Management Service will be determined by the Attribute Mapping Service.

11.3.2.3 Account Searching

The searching functionality will be provided by OpenAM's corresponding REST interface[21]

11.3.2.4 Account Deletion

The deleting functionality will be provided by OpenAM's corresponding REST interface[22]. However, in order to trigger the physical or logical deletion of all personal data linked to the account to be deleted, the **CREDENTIAL-specific OpenAM IdRepo compliant-class** introduced in Section 11.2 will also **overwrite the delete method** and manage the communication with the data management service.

11.3.2.5 Key Recovery

OpenAM's closest functionality to the mechanism described as the key recovery one is the change password (Developer Guide[23], Section 2.1.5.1.7) or replacing forgotten passwords (Section 2.1.4.3). However, and as it does not fully match the needs of CREDENTIAL a specific plugin will have to be developed.

First, an additional attribute must be associated to each account, the recovery public key. This key represents, for all means, a new account. Once activated the key recovery mechanism, all information associated to the original account will be transferred to the latter. As a result of the process, the original account will be deleted from the system (logically or physically) and the new account will become the main one. The recovery account attribute will be added to OpenAM using the same mechanism as other attributes and described in its documentation (Developer Guide[24], Section 4.1). The recovery mechanism will be triggered using the same mechanism as the one used for recovering the username or the password. Similar to the ForgottenPasswordConfigProvider[25] a KeyRecovery plugin will have to be developed and will include all the functionality related to the key recovery identified in the Account Management Component:

- **Challenge recoverAccount(byte[] recoverAccountPublicKey)**

[20] https://github.com/OpenRock/OpenAM/blob/master/openam-core/src/main/java/com/sun/identity/idm/plugins/database/DatabaseRepo.java
[21] https://backstage.forgerock.com/docs/openam/13/dev-guide#rest-api-query-identity
[22] https://backstage.forgerock.com/docs/openam/13/dev-guide#rest-api-query-identity
[23] https://backstage.forgerock.com/cp/portal/cloudstorage/pdf-1771818580?redirect=true
[24] https://backstage.forgerock.com/cp/portal/cloudstorage/pdf-1771818580?redirect=true
[25] https://framagit.org/teddyber/openam/blob/f36d7fc719bcbc9aba272b40a4723a24a81d6e0c/openam-selfservice/src/main/java/org/forgerock/openam/selfservice/config/flows/ForgottenPasswordConfigProvider.java

- **Challenge confirmRecoverAccount(ChallengeResponse challengeResponse)**

The recovery of a key will immediate trigger the re-generation of the re-encryption keys associated to the recovered account.

11.3.3 Access Management Service

The access management concepts in the CREDENTIAL Wallet in the context of OpenAM are handled in two different ways:

- System-wide authorization policies: are related to what users are generally entitled to do in OpenAM
  - Administrators: may modify the configuration of OpenAM or the CREDENTIAL Wallet
  - Self service operations: the system may be configured so that users may handle their own accounts (creation, modification, deletion, key recovery) and their associated personal data, establishing their own policies (dynamic policies)
- User-managed authorization policies: related to the personal data stored in the Data Management Service. Users may choose to authorize other participants to access or even modify their data

System-wide authorization policies are managed at a configuration and deployment level. The system will be configured so the requirements are met (e.g. only owners of an account may delete it or recover it, only owners of personal data may establish access policies related to them).

User-managed authorization policies, in the case of the CREDENTIAL Wallet using OpenAM will rely on its implementation of the UMA standard. This standard specifics two different sets of APIs that, applied to the CREDENTIAL Wallet context, results in:

- **Managing resource sets**: Users willing to share some of their personal data will have to register such data as resource sets. While this operation should be transparent for the user, the personal data to be shared with other participants will be registered in OpenAM using its REST APIs (see OpenAM Administration Guide[26], Procedures 15.8, 15.9, 15.10, 15.11 and 15.12). Notice that the Data Management Service will need to keep the track between its internal identifiers and resource identifiers created by OpenAM
- **Managing access policies**: OpenAM, through its REST APIs allow resource owners to manage the policies associated to its registered resource sets, procedures 15.17 through 15.21 of OpenAM Administration Guide[27]

Besides these aspects, OpenAM already also provides, out of the box, the remaining required functionality to operate UMA's protocol, e.g. token registration and management (permission, relying parties and access). The only aspect to be developed is the association of re-encryption keys and the policies. Re-encryption keys will have to be associated to policies and to the origin and destination accounts.

[26] https://backstage.forgerock.com/cp/portal/cloudstorage/pdf-186653850?redirect=true
[27] https://backstage.forgerock.com/cp/portal/cloudstorage/pdf-186653850?redirect=true

As a summary, there is **limited implementation effort related to the access management service** as all required functionality is mostly provided out-of-the box. All **related effort is related to the configuration of the system** and the implementation of the re-encryption key store, the request permission service and the clients that will make use of the referenced APIs.

11.3.4 Data Management Service

The data management service (DMS) is an encapsulated service and the only part where this service will have to be integrated with OpenAM is UMA. The DMS will have to act as UMA Resource Server by making use of OpenAM's "UMA Resource Set Endpoint for Resource Servers"[28]. It will also have to have a strong connection with the Access Control Filter, which will be the responsible for granting or denying access to the resources.

11.3.5 Auditing Service

The auditing service has two major functions in the CREDENTIAL Wallet. First, it is responsible to log certain events that need to be logged. Second, an interface for querying the log events occurred in the past for a given search query has to be offered.

The logging functionality is realized in OpenAM through the "Common Audit" feature. This will be done by plugging in audit event handlers in the components which in response handle events that need to be logged and write them into a log file or database. The logging service of OpenAM keeps track of which user is affected by a certain log event. By extending it, it is possible to write a logger that writes the log events separated for each user. In addition, the privacy and security critical parts can be encrypted using the CREDENTIAL Wallets cryptographic service.

The query functionality is independent from OpenAM. It will be implemented by reading the log files generated from the OpenAM framework.

11.3.6 Notification Service

On the one hand, we will have the notification service that interfaces with the user to establish the notification preferences. This preference setting service has no equivalent in OpenAM and will have to be implemented from scratch. On the other hand, the notification service needs to process events so it can notify in accordance to the preferences. We can a) use the events stored in a log database as the source of notifications or b) overwrite or extend an existing EventHandler to include this logic. If there is a need for handling other sources of events, it is out of the scope of OpenAM so it will have to provide a separate interface

11.3.7 Authentication Client

As stated in the Authentication Service section of this Appendix, there are many authentication methods directly supported by OpenAM. Each of these methods requires a different number of steps or inputs

[28] https://backstage.forgerock.com/docs/openam/13.5/admin-guide/chap-uma#managing-uma-resource-sets

therefore no generic client can be created. However, the Android Authentication Client will support the CREDENTIAL specific authentication methods: FIDO UAF and CRAM. The client will support, in one hand, the interaction between the Mobile application and a FIDO server and on the other, the interaction with the Authentication Plugins, and the necessary authentication steps through OpenAM's RESTful authentication services.[29] The client will be based on an open source implementation of an Android Authentication client for OpenAM[30].

11.3.8 Account Management Client

The account management client will be implemented as a java client for OpenAM's self-service REST API [31] (Section 2.1.5.1.3) and the identity management REST API[32]

11.3.9 Access Management Client

The access management client will be implemented as a java client for policy management REST services[33 and 34] provided by OpenAM.

[29] https://wikis.forgerock.org/confluence/display/openam/Use+OpenAM+RESTful+Services#UseOpenAMRESTfulServices-Authentication.
[30] https://github.com/codice/forgerock-commons/tree/master/samples/mobile/android/openam-apps/trunk/android_auth_app
[31] https://backstage.forgerock.com/cp/portal/cloudstorage/pdf-1771818580?redirect=true
[32] https://backstage.forgerock.com/docs/openam/13/dev-guide#rest-api-query-identity
[33] https://backstage.forgerock.com/docs/openam/13/dev-guide/chap-client-dev#rest-api-authz-applications
[34] https://backstage.forgerock.com/docs/openam/13/dev-guide/chap-client-dev#rest-api-authz-policies

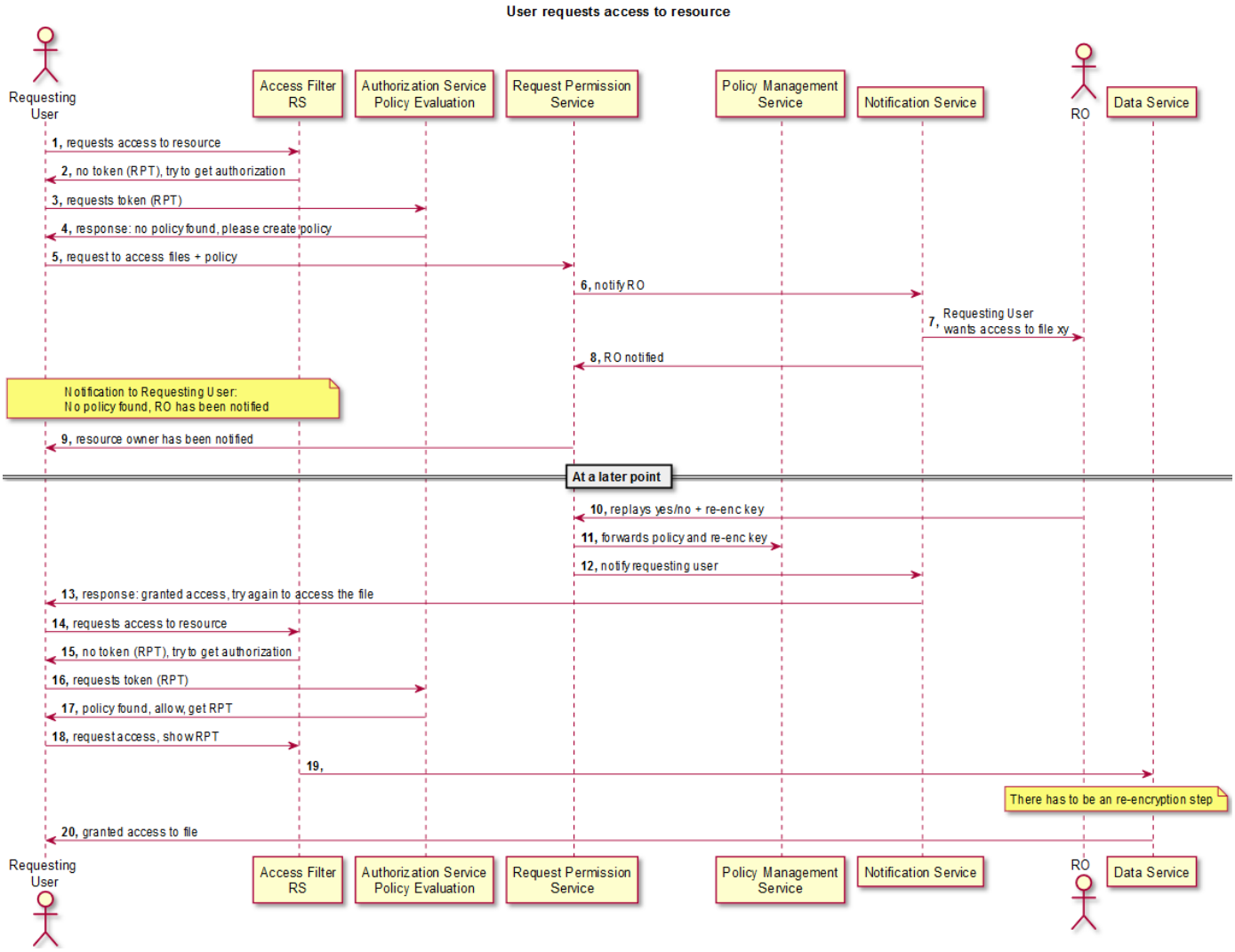


Figure 45: User Requests Access to a Resource Sequence Diagram

Figure 45 shows a sequence diagram where a Requesting User tries to access a resource. While realizing this use case, the second use case where a resource owner shares a resource is covered as well. In this scenario, we have to consider that we will have to extend the UMA flow. In step 3 the requesting user tries to get the RPT to be able to access a resource. The evaluation service will request a policy from the requesting user, which will be generated on his/her phone based on the resources he/she wants to access. This policy has to be confirmed by the user by using a push notification. After the resource owner allows or denies the request the requesting user is being notified. If the Resource Owner (RO) allows the access to the requested files not only the policy will be saved, but also the required re-encryption keys are generated and stored. Next, the requesting user will have to get authorization again. Now he/she will receive the RPT, which allows the user to access the resources on the data service.

11.3.10 Data Management Client

The data management client is fully independent from OpenAM services and it will be implemented as a java library proxying the functionalities provided by the Data Management Service.

11.3.11 Notification Client

The data management client is fully independent from OpenAM services and it will be implemented as a java library proxying the functionalities provided by the Notification Service.

11.3.12 Cryptographic Service

The data management client is fully independent from OpenAM services and it will be implemented as a standalone java library suitable for integrating in the mobile application and in the CREDENTIAL Wallet.

11.3.13 Access Control Filter

The access control filter (ACF) protects the underlying services and data from the incoming requests. There are different request scenarios related to the part where the request arrived. This section details the different types of ACF and how they are being realized using OpenAM. The more generic description can be found in Section 6.4.1 Access Control Filter.

Basically, we have to distinguish three types of access control filters:

11.3.13.1 Access Control Filter for the server-side

For the OpenAM functionality (IDP, Manage Policies, Audit Service …), we should be able to re-use the built-in access control by configuring it. OpenAM supports a variety of different configurations, which can be used to solve the access control filter problem on the server side.

The built-in OpenAM configurator is used to setup the OpenAM server. The configuration guide[35] shows how to configure the server as well as how users, web agents and policies are being created.

11.3.13.2 Access Control Filter for the Data Service

The Data Service has its own Access Control Filter, which should protect the resources on this service. It is protected using the UMA approach. The basic UMA flow using OpenAM is depicted in Figure 46. The data service will be the resource server in this scenario. A requesting party will only get access from the ACF to a resource if the request includes a valid Requesting Party Token (RPT). Otherwise, the requesting party has to try to get the RPT.

We will use the already existing UMA mechanisms to protect our data service, which will provide fine-grained granularity to fulfil our needs.

[35] https://wikis.forgerock.org/confluence/display/openam/3+OpenAM+Server+Configuration

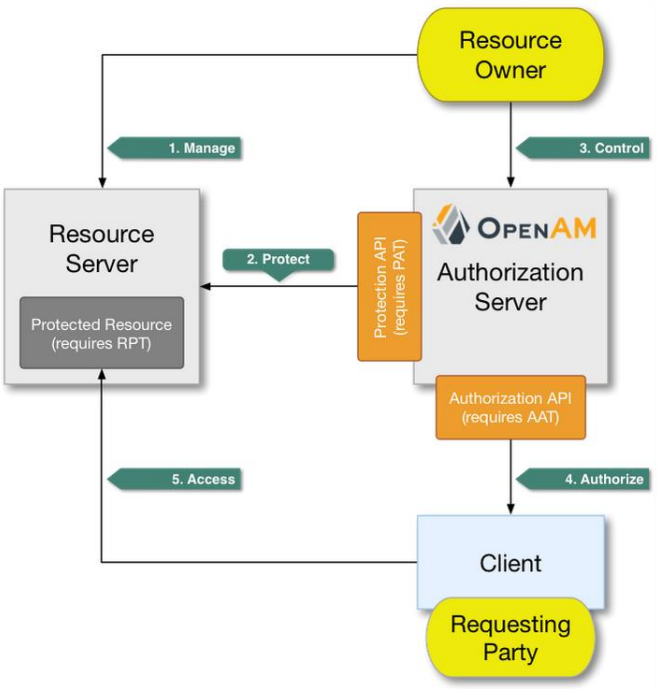


Figure 46: OpenAM UMA Process Flow[36]

11.3.13.3 Access Control Filter for custom services (outside OpenAM)

OpenAM offers – to protect external services – the so-called OpenAM Policy Agents[37]. These agents inspect the incoming requests and decide what the next steps are. The basic interaction between user, agent and OpenAM is depicted in Figure 47: OpenAM Policy Agents Basic Interaction Flow and consists of five steps, which are detailed as follows.

1. The web client requests access to a protected resource.
2. The web server runs the request through the policy agent that protects the resource according to OpenAM policy. The policy agent acts to enforce policy, whereas OpenAM handles the policy configuration and decisions.
3. The policy agent communicates with OpenAM to enforce a policy decision.
4. When OpenAM approves access to a resource, the policy agent allows access.
5. The web server returns the requested access to the web client.

[36] https://backstage.forgerock.com/docs/openam/13/admin-guide/chap-uma
[37] https://wikis.forgerock.org/confluence/display/openam/OpenAM+Policy+Agents

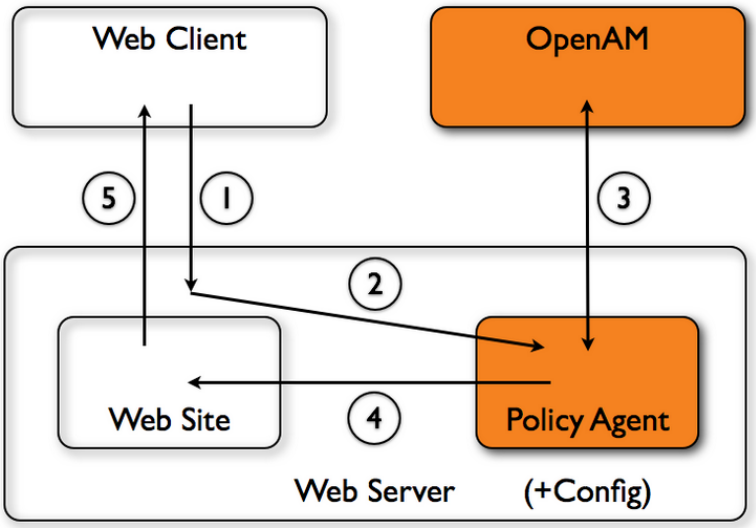


Figure 47: OpenAM Policy Agents Basic Interaction Flow[38]

There are different kinds of policy agents available such as Web Agents, or the Tomcat J2EE Agent. These agents are placed in front of an external service to protect its resources. The installation and configuration procedure is described in the related guide[39].

11.3.14 Data Sanitization Filter

It is not directly affected by the OpenAM decision; however, it should use the OpenAM logging service to store the events if logging is configured.

Similar to the Forgerock-provided CORS-Filter the data sanitization filter will be added through a "*" filter-mapping within WEB-INF/web.xml. This will redirect all incoming traffic to the supplied filter.

OpenAM offers a RESTful interface for its logging service[40]. If logging of potential malicious data is enabled, this logging service will be used to persist the gathered data. To achieve this, a REST-based communication between the Data Sanitization Filter and the OpenAM Logging Service will be performed.

[38] https://backstage.forgerock.com/docs/openam-policy-agents/3.1.0/agent-install-guide#chap-about-web-agents
[39] https://backstage.forgerock.com/docs/openam-policy-agents/3.1.0/agent-install-guide#chap-apache-20
[40] https://wikis.forgerock.org/confluence/display/openam/Use+OpenAM+RESTful+Services#UseOpenAMRESTfulServices-Logging

11.3.15 Auditing Filter

It is not affected by the OpenAM decision; however, it should use the OpenAM logging service to store the events. OpenAM offers a RESTful interface for its logging service[41]. The audit filter therefore has to just create the loggable event and send a request to the REST service of the OpenAM logging service.

41 https://wikis.forgerock.org/confluence/display/openam/Use+OpenAM+RESTful+Services#UseOpenAMRESTfulServices-Logging

12 Appendix B – CREDENTIAL Generic Mobile Application

12.1 Introduction

One of the objectives CREDENTIAL aims to is "*02.Protect access to identity data with strong authentication mechanisms*". The multi-factor authentication schemes supported by hardware, such as mobile devices, is one of the methods, among others, developed by CREDENTIAL to achieve this goal.

Therefore, the CREDENTIAL consortium agrees focusing in the Mobile-First approach, where effort is concentrated on the creation of a Generic Mobile App which will allows the user to interact in a secure way with the CREDENTIAL Wallet deployed in the cloud, providing all the generic functionality, which includes authentication, granting access rights, notifications and managing the user's data. This CREDENTIAL mobile application facilitates the CREDENTIAL account and data management by the user through the smart phone, demonstrating the use of the CREDENTIAL Wallet from mobile devices on different scenarios.

Thus, the CREDENTIAL Mobile App becomes a key and complex component of the CREDENTIAL system.

12.2 CREDENTIAL Mobile App Architecture

The CREDENTIAL account owner will use the mobile application for the following purposes:

- Register on the CREDENTIAL Wallet;
- Authentication against CREDENTIAL Wallet or against external IDPs;
- Use of cryptographic services offered by CREDENTIAL;
- Upload/download data to/from CREDENTIAL Wallet;
- Control the access to his data. Give consent for data release to trusted participants (SP);
- Monitor the access to his data, who, when and how frequently his data is accessed.

The CREDENTIAL Generic Mobile application based on Android 6.0 Marshmallow version is covering all of these functionalities. The mobile application comprises three main components:

- **User Interface**: Generic UI providing most of CREDENTIAL functionalities, that can be used by all the pilots;
- **Participant Toolkit**: Provided by CREDENTIAL is built by components containing services/functionalities which will facilitate the integration into CREDENTIAL Wallet.
- **Orchestrator module**: Module in charge of receiving the user requests and managing the processes involved, making the appropriate calls to the Participant Toolkit component to connect to the CREDENTIAL Wallet right service and secure elements.

These components are depicted in Figure 48 and a more detailed description is providing in the next subsections.

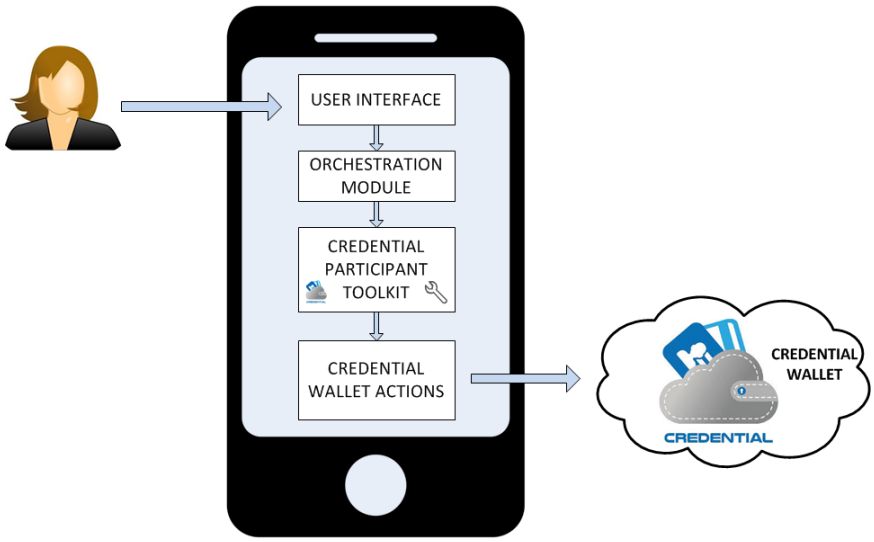


Figure 48: Mobile Generic Application High Level View

12.3 User Interface

The UI has been developed taking into account a user-centered design approach (work developed in parallel with this document in D3.1 UI Prototypes V1), and evaluated against users. Figure 49 shows three different screenshots developed under these principals.

Figure 49: UI Screenshoots for Sign-in, Authentication and Data Management

Figure 50 shows the needed data and operations for the user interface.

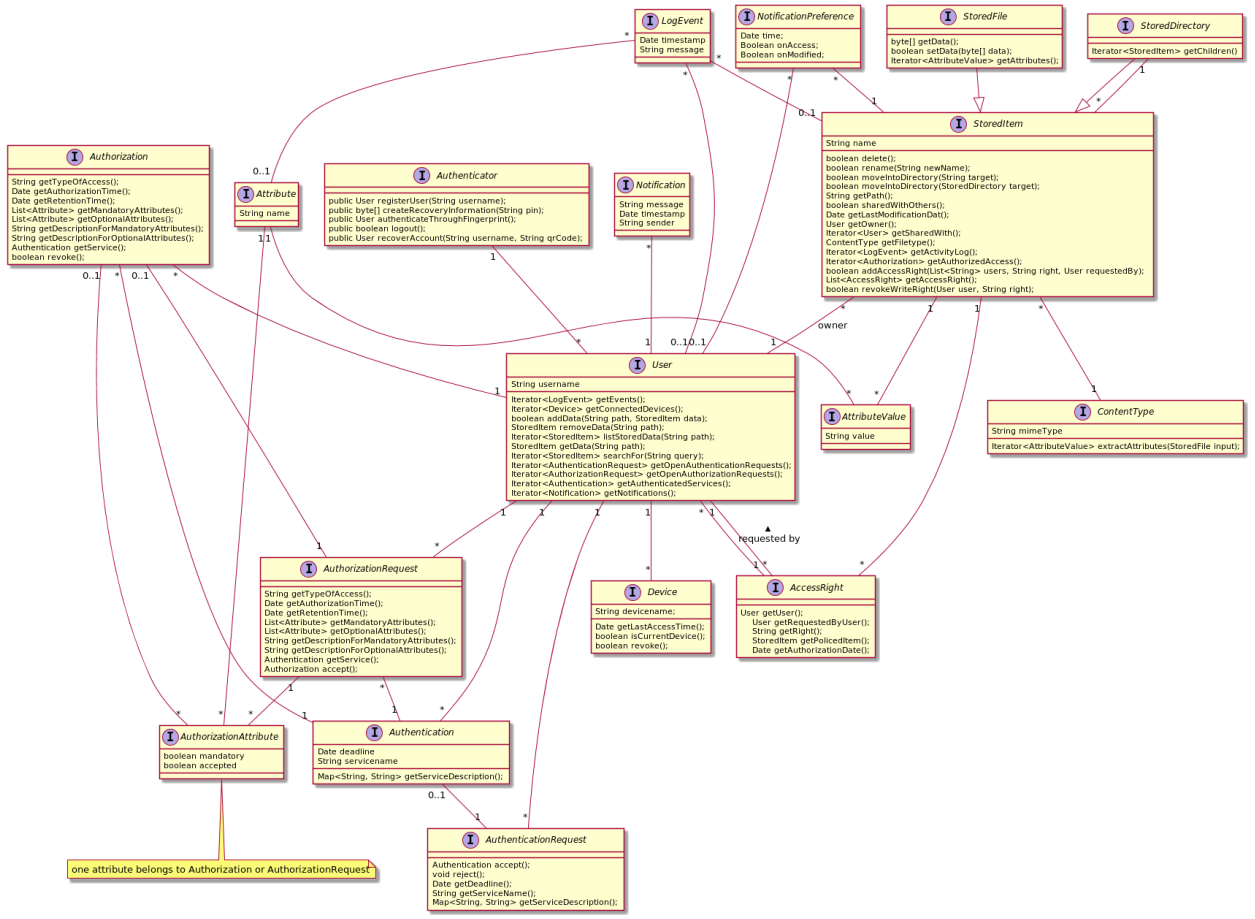


Figure 50: Mobile Generic Application Data Model

12.4 Participant Toolkit

The Participant Toolkit is made of client components providing the means to integrate the CREDENTIAL Mobile App with the services provided by the CREDENTIAL Wallet. The client components consume the CREDENTIAL Wallet's services but the cryptographic subcomponents. In this case the functionalities are included in the subcomponent itself. The components of the Participant Toolkit are depicted in Figure 51, where the links with other mobile app components e.g. UI, and with elements belonging to the mobile device, such as secure HW modules, are also included.

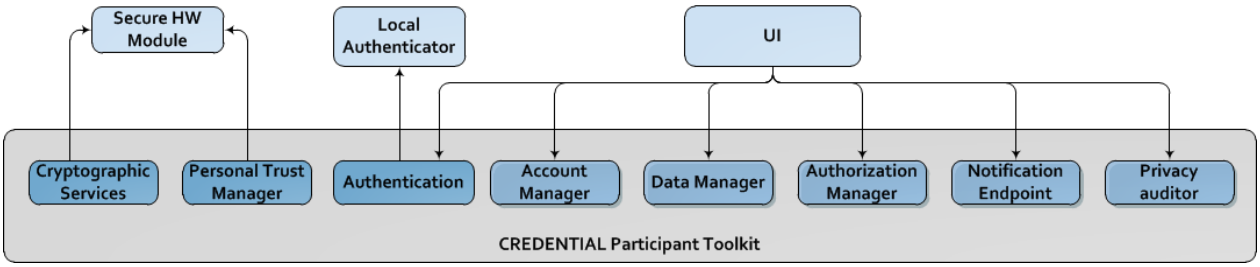


Figure 51: Participant Toolkit Components

A short description on the Participant Toolkit components is provided next (a more detailed description of these clients can be found in section 6.3, and OpenAM details are provided in section 11) .

- **Authentication Client**: Consumes the authentication services provided by OpenAM using an API REST interface. Also provide access to native libraries, e.g. the authentication client interface with FIDO's native UAF client;
- **Access Management Client**: Uses OpenAM through its REST APIs;
- **Account Management Client**: Linked to OpenAM's self-registration capabilities, and to cryptographic services;
- **Data Management Client**: Client-side library consuming Data Management CREDENTIAL Wallet's service;
- **Notification Client**: Uses the REST-interface offered by the server-side for managing user preferences. Receives notifications from CREDENTIAL Wallet services.
- **Cryptographic Service**: Cryptographic library system specific to the client, including re-encryption, key generation, personal trust store and encryption/decryption service functionalities.
- **Privacy auditor service**: Components which provides horizontal functionalities such as access control filter, auditing filter and data sanitation filter. These subcomponents ensure and additional level of security and auditing.

12.5 Orchestrator Module

This module plays a management role for the CREDENTIAL Mobile App, managing connections and interactions between the UI and the Participant Toolkit components, as follows:

- Receives the requests coming from the UI;
- Associates the requested call with the appropriate client component;
- Once the client performs the action, receives the process result and performs additional actions if needed;
- Provides to the UI the result of the process, if needed.

Figure 52 displays the interactions between the orchestration module and the rest of mobile application components.

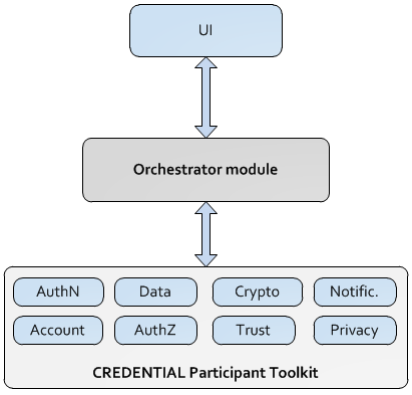


Figure 52: Orchestrator Module's Interactions

12.6 CREDENTIAL Mobile App and Pilots

The CREDENTIAL pilot partners will provide different scenarios where the CREDENTIAL Wallet services and functionalities are used.

The pilots will not directly interact with the generic mobile app, but the owner of the CREDENTIAL account will control the access to her data requested by the pilot SP. The account owner will receive notifications asking for permission to use her data, the user through the CREDENTIAL mobile app will give consent for data release to the requested SP.

Figure 53 gathers the different scenarios envisaged in CREDENTIAL and shows the use of the CREDENTIAL Mobile App by the owner account for accessing the CREDENTIAL Wallet services.

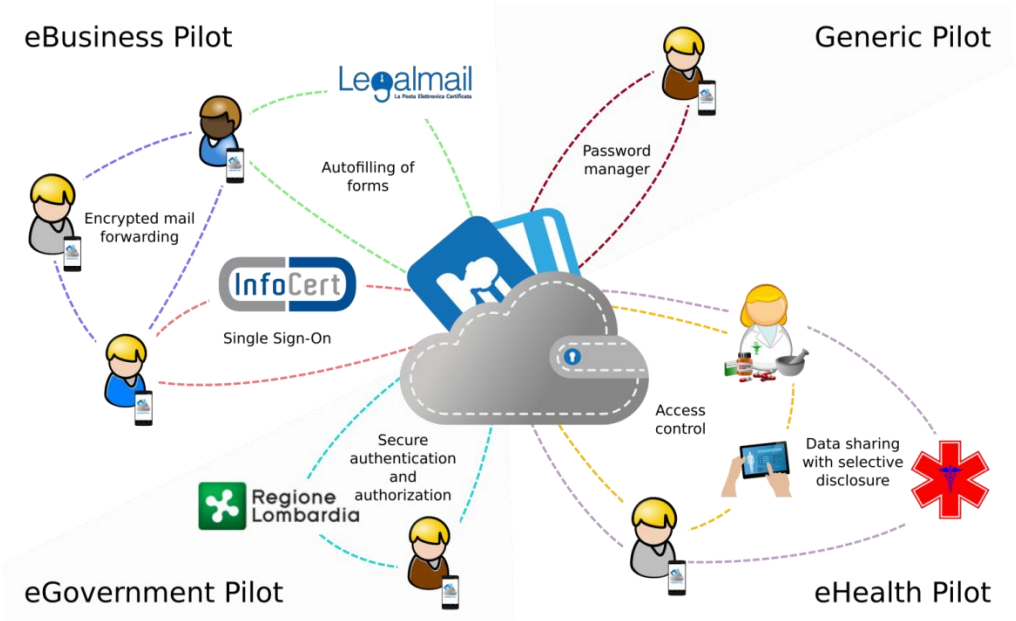


Figure 53: Mobile Generic Application Used by Pilots

13 Appendix C – Mapping between Use Cases and Development Components

In order to ensure that the CREDENTIAL Wallet supports all the generic logical use cases identified in D2.1 and that all the Wallet's development components are justified by some required functionality, a traceability matrix has been built. In it, each of the components is mapped to their corresponding logical use cases.

	Notification Service	Data Management Service	Authentication Service	Access Management Service	Account Management Service	Auditing Service	Mobile App	Web Browser	Cryptographic Service	Service Provider	Id. Provider	Attr. Provider
G-LUC-REENCDATA		x							x			
G-LUC-VERIFYRECOVERYREQUEST					x							
G-LUC-READRECOVERYPRIVKEY							x					
G-LUC-FILLREGFORM										x		
G-LUC-RENDERREGFORM										x		
G-LUC-ADDATTRIBUTSREGFORM								x		x		
G-LUC-SUBMITREGFORM								x		x		
G-LUC-AUDITING						x						
G-LUC-RECDATA		x										
G-LUC-DECDATA		x							x			
G-LUC-ENCDATACREDENTIAL		x							x			
G-LUC-SENDENCDATA		x										
G-LUC-REGDATA		x										
G-LUC-DEFREQPARAMS		x					x			x		
G-LUC-REQDATA		x					x			x		
G-LUC-SEARCHDATA		x										

	Notification Service	Data Management Service	Authentication Service	Access Management Service	Account Management Service	Auditing Service	Mobile App	Web Browser	Cryptographic Service	Service Provider	Id. Provider	Attr. Provider
G-LUC-REQSIGN												
G-LUC-CREATESIGNREQ												
G-LUC-PROVSIGNREQ												
G-LUC-PROVDATASIGNREQ												
G-LUC-RECSIGNREQ												
G-LUC-SIGNDOC									x			
G-LUC-RECSIGNDOC												
G-LUC-RECEXTTRIGEVENT	x											
G-LUC-EVALNOTIFYCONF	x											
G-LUC-CREATENOTIFYLIST	x											
G-LUC-SENDNOTIFY	x											
G-LUC-PROCNOTIFY	x											
G-LUC-CREATEDELDATAREQ		x										
G-LUC-SUBMITDELDATAREQ		x										
G-LUC-DELDATA		x										
G-LUC-ENCDATA		x							x			
G-LUC-REQREK									x			
G-LUC-PROCEXC												
G-LUC-AUTHSPUSINGWALLETENABLEDIDP			x							x		
G-LUC-AUTHSPUSINGWALLETIDP			x							x		
G-LUC-AUTHWALLETUSINGENABLEDIDP			x								x	
G-LUC-AUTHWALLETUSINGEXTERNALIDP			x								x	
G-LUC-SELECTCREDIDP			x								x	

	Notification Service	Data Management Service	Authentication Service	Access Management Service	Account Management Service	Auditing Service	Mobile App	Web Browser	Cryptographic Service	Service Provider	Id. Provider	Attr. Provider
G-LUC-AUTHSPUSINGWALLETANDIDENTITYFEDERATION			x								x	
G-LUC-ATTCOLLECTION											x	
G-LUC-REQIDENTIYASSERTION			x									
G-LUC-ENCATTRIBUTES									x			x
G-LUC-ISSIDENTITYASSERTION			x									
G-LUC-RECIDENTITYASSERTION			x									
G-LUC-DECIDENTITYASSERTION									x			
G-LUC-CREATELOGOUTREQ			x									
G-LUC-LOGOUTCREDWALLET			x									
G-LUC-INVSESSION			x									
G-LUC-RESPSUCCLOGOUT			x									
G-LUC-XYZ			x									
G-LUC-ASKIDPS			x									
G-LUC-ASKIDPFROMLIST			x									
G-LUC-SELECTIDP			x									
G-LUC-REDIRECTUSER			x									
G-LUC-SELECTATTRDISCLOSE			x						x		x	
G-LUC-REDACTIDASSERTION			x						x			
G-LUC-PROVIDASSERTION			x						x		x	
G-LUC-VERIDASSERTION			x						x	x		
G-LUC-REQACCESSRIGHTS				x								

	Notification Service	Data Management Service	Authentication Service	Access Management Service	Account Management Service	Auditing Service	Mobile App	Web Browser	Cryptographic Service	Service Provider	Id. Provider	Attr. Provider
G-LUC-REGACCESSRIGHTSREQ				x	x							
G-LUC-PROVACCESSRIGHTSREQ				x								
G-LUC-DEFINEACCESSRIGHTS				x								
G-LUC-GRANTACCESSRIGHTS				x								
G-LUC-PROVIDEACCESSRIGHTS				x								
G-LUC-REGACCESSRIGHTS				x								
G-LUC-VERREENCREQ									x			
G-LUC-REQACCRIGHTSLIST				x								
G-LUC-RECACCESSRIGHTSLIST				x								
G-LUC-ACCESSDENIED				x								
G-LUC-REGLINKACCREQ					x					x		
G-LUC-GENKEYPAIR									x			
G-LUC-LINKACCREQ					x					x		
G-LUC-REDIRECTUSERCREDAUTH			x									
G-LUC-PROVLINKACCREQ					x					x		
G-LUC-AUTHORIZATION				x								
G-LUC-SEARCHUSER					x							
G-LUC-CREATEDEREGACCREQ					x							
G-LUC-SUBMITDEREGACCREQ					x							
G-LUC-DEREGACC					x							
G-LUC-CREATECREDACCREQ					x				x			
G-LUC-SUBMITCREDACCREQ					x							
G-LUC-CREATECREDACC					x							

	Notification Service	Data Management Service	Authentication Service	Access Management Service	Account Management Service	Auditing Service	Mobile App	Web Browser	Cryptographic Service	Service Provider	Id. Provider	Attr. Provider
G-LUC-CREATELISTPREVLOGREQ						x						
G-LUC-SUBMITLISTPREVLOGREQUEST						x						
G-LUC-QRYLISTPREVLOGINS						x						
G-LUC-RETLISTPREVLOGINS						x						
G-LUC-CREATEBANUSERREQ					x							
G-LUC-SUBMITBANUSERREQ					x							
G-LUC-BANUSER					x							
G-LUC-SUBMITUNBANUSERREQ					x							
G-LUC-UNBANUSER					x							
G-LUC-CREATEEXPORTDATAREQ		x										
G-LUC-SUBEXPDATAREQ		x										
G-LUC-RETEXPDATA		x										
G-LUC-CREATEVIEWACCESSESREQ						x						
G-LUC-SUBMITVIEWACCESSESREQ						x						
G-LUC-SEARCHACCESSES						x						
G-LUC-RETLISTALLACCESSES						x						
G-LUC-ASSOCKEYPAIR					x							
G-LUC-GENUPDREENCKEY							x		x			
G-LUC-GENREENCREQ							x		x			
G-LUC-DATAREGCONF		x										
G-LUC-CREATERECREQ					x		x					
G-LUC-SUBMITRECOVREQ					x		x					
G-LUC-EXPPRIVKEY							x		x	x		

	Notification Service	Data Management Service	Authentication Service	Access Management Service	Account Management Service	Auditing Service	Mobile App	Web Browser	Cryptographic Service	Service Provider	Id. Provider	Attr. Provider
G-LUC-IMPPRIVKEY							x		x	x		