COMPETITIVENESS AND INNOVATION FRAMEWORK PROGRAMME
ICT PSP Fifth Call for proposals 2011 - Pilot Type A

Towards a single European electronic identification and authentication area

ICT PSP call identifier: CIP-ICT-PSP-2011-5
ICT PSP Theme/objective identifier: 4.2

Project acronym: STORK 2.0
Project full title: Secure idenTity acrOss boRders linKed 2.0
Grant agreement no.: 297263

D4.10 Final version of Technical Design

Deliverable Id :	**D4.10**
Deliverable Name :	**Final version of Technical Design**
Status :	**Final**
Dissemination Level :	**PU**
Due date of deliverable :	**July 31st, 2015**
Actual submission date :	**October 2nd, 2015**
Work Package :	**WP4**
Organization name of lead contractor for this deliverable :	**MINHAP**
Author(s):	**WP4 core team**
Partner(s) contributing :	**ALL**

Abstract: This document describes the final architecture of the systems that compose the common functionalities of the STORK 2.0 platform. This description is made from various points of view, conforming to the RUP methodology. The relevant points of view are applied to each of the two systems: PEPS and Virtual IDP. The deliverable also describes "commodities", which are software components discovered to be useful in several places.

Finally detailed software design is provided, describing class diagrams for each module and interface specification for each package.

Project co-funded by the European Community under the ICT Policy Support Programme

History

Version	*Date*	*Modification reason*	*Modified by*
0.0	23/09/2013	D4.3 is used as a template	
0.1	14/08/2015	Update with comments from the first review and from Enes Özbay. Included the software design.	John Heppe
0.2	7/09/2015	Inclusion of the contributions of Multicert, Tubitak, Advania, ARGE, UAegean	John Heppe
0.3	24/09/2015	Quality review	ATOS
0.4	29/09/2015	Integration of comments from quality check	John Heppe
0.5	02/10/2015	Final quality check	ATOS
Final	02/10/2015	Final deliverable	

Table of contents

List of figures

List of tables

List of abbreviations

AOI	Attribute Object Identifier
AP	Attribute Provider
A-PEPS	Attribute PEPS (PEPS role for attribute collection in foreign countries)
AQAA	Attribute QAA, quality of attribute(s)
AUB	Authentication on behalf of
BA	Domain-specific attribute
CMS	Cryptographic Message Syntax
C-PEPS	Citizen PEPS
CRL	Certificate Revocation List
CSR	Create Signature Request
DTL	Document Transfer Layer
e-CODEX	e-Justice Communication via Online Data Exchange
eID	Electronic Identity
epSOS	Smart Open Services for European Patients
EU	European Union
HSM	Hardware Security Module
IdP	Identity Provider
LSP	Large Scale Pilot
MS	STORK 2.0 Member State
MW	MiddleWare
NFR	Non Functional Requirement
OASIS DSS	OASIS Digital Signature Services
OCSP	Online Certificate Status Protocol
PEPPOL	Pan-European Public Procurement Online
PEPS	Pan European Proxy Server
PO	Powers (for digital signature)
PV	Powers Validation
QAA	Quality Authentication Assurance
SAML	Security Assertion Markup Language
SP	Service Provider
S-PEPS	The PEPS role to attend SP requests
SPI	Serial Peripheral Interface
SPOCS	Simple Procedures Online for Cross- Border Services

STORK 2.0	Secure idenTity acrOss boRders linKed 2.0
SVN	Subversion
VC	Version Control
VCF	Version Control File
VCP	Version Control PEPS
V-IDP	Virtual Identity Provider, the system of the decentralized deployment architecture (formerly referred to as “MW architecture”)

Executive summary

This document describes the final architecture of the systems that compose the common functionalities of the STORK 2.0 platform. This description is made from various points of view, each described in a separate subchapter. The relevant points of view are applied to each of the two models: The centralized deployment model “Pan European Proxy Service (PEPS)” and the decentralized model (formerly “Virtual Identity Provider, V-IDP”), being the system represinting the MW architecture.

Each of these systems is described in a separate chapter (2 and 3) subdivided in subchapters, according to these views. As both systems obey to the same objectives, the majority of functions and processes are the same. Therefore, the first described system, PEPS, is very detailed, whereas the description of the second system is much shorter, due to the fact that most text would be the same.

For each of these systems, all included modules are described: the AUB, PO, BA and PV business processes, the Signatures, Version Control and Anonymity; each within the roles of the PEPS (S-PEPS, C-PEPS and A-PEPS). However, for the V-IDP these roles are distinguished internally, but not expressed in the software architecture design, as, instead of redirections to different systems, the V-IDP performs internal routing between modules.

A fourth chapter describes “commodities” which are software modules found to be equal in several different parts of the project, which should be developed by a common team. These commodities are:

- eIdentifier encryption (National Identifier Privacy), in order to support avoiding that different files can be matched and user profiles created.
- Version Control (SPs), to support that national PEPSes obtain information from their SPs, and systems administratoirs are warned when changes have been taken place.
- Personal Data comparison (for re-authentication), which allows Service Providers to peforma an intelligent verification of the similarity of names (given name and surname), in order to determine if domain specific attributes may belong to the same person.
- Browser Temporary Storage Management, which allows a better user-experience from one session to the other one.
- SAML Unpackager, a javascript, which allows the C-PEPSes to show business attributes, without doing the interpretation. The unpackager can do the translation of the attribute names, but won’t do any translation of the contents, which can fogure in any language.

The final chapter describes the detailed software design with class diagrams for modules and interface specifications for each package. The modules correspond to the software modules in SVN, the common software repository of the project.

The first four chapters are based on the deliverable D4.3 First version of the technical design with an update due to comments by the reviewers, and to what is really implemented. Another important input for this document is D4.9 Final Version of the Functional Design, adapting many specifications.

1 Introduction

D4.10 describes the final architecture of the systems that compose the common functionalities of the STORK 2.0 platform from various points of view. The relevant points of view are applied to each of the two models: centralized and decentralized (formerly PEPS and MW), this last one including the Virtual ID Provider (V-IDP). The document also describes "commodities" which are software modules found to be equal in several different parts of the project, which should be developed by a common team. This document forms the basis for development of these solutions.

This deliverable is based on D4.3 First version of the Technical Design, which was used as a template for this deliverable, and D4.9 Final version of the Functional Design, which includes several changes in the functionalities to be built.

D4.10 uses the first version of the Technical Design (D4.3) as a template or draft version, with minor changes due to comments by the reviewers, the WP4 core group, as well as due to the real implementation. The major change in this document is the inclusion of the detailed software design, which describes the different modules, packages and classes in successive chapters. Another important input for this document is D4.9 Final Version of the Functional Design, adapting many specifications.

It is important to highlight that D4.10 only describes the common functionalities of the STORK 2.0 Platform. Specific functionalities, organisational and infrastructure aspects are to be determined, built and implemented by each Member State.

The deliverable is structured as follows: This document describes, after this first Introduction chapter, the two systems. As both are so similar, first the PEPS is described (Chapter 2) and in the following chapter (Chapter 3) the differences for the V-IDP are described. This approach is oriented to share as much as possible, the developments for both systems. The first subchapter for the PEPS starts with a general context description, zooming in, in the next subchapter with the use cases and other requirements. After these general overviews, the description gets to more detail in the next subchapters describing the views which are considered relevant by the architecture designers. The V-IDP chapter follows a similar structure, although empty paragraphs are to be understood as the same or very similar to the PEPS paragraphs, substituting the term PEPS by V-IDP. The last but one chapter (Chapter 4) is commodities; procedures or processes which have been mentioned along the documentation, especially the Functional Design, but also pilot documentation, which should be part of the common developments, but do not fit in the structure of chapter 2 and 3. The last chapter (Chapter 5) is new, compared to D4.3. It details the technical implementation with class diagrams for modules and the detailed interface specification for each Java package.

1.1 Methodology

The deliverable follows a methodology based on the *RUP (Rational Unified Process) 4+1* view model. This model considers five views as normally sufficient to describe a system, the first one being the use case view. Nevertheless, in different publications, different views are described for the other four views, although all agree on the most important one: the logical view.

In this logical view the system is divided in subsystems, and for each subsystem the different business processes are described.

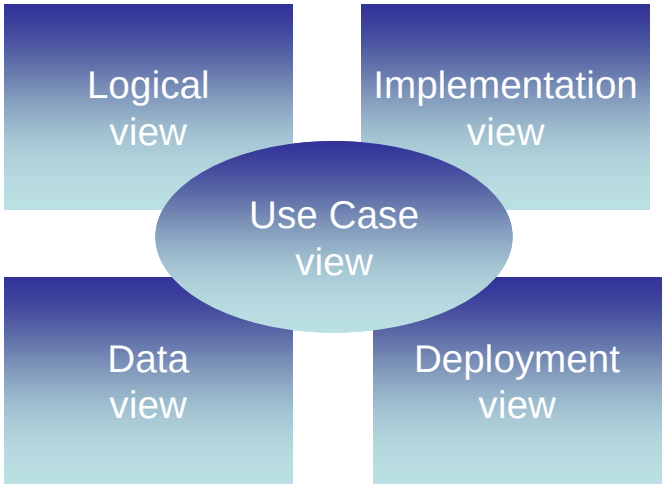


Figure 1: RUP 4+1 view model

2 PEPS Architecture Design

2.1 System Context

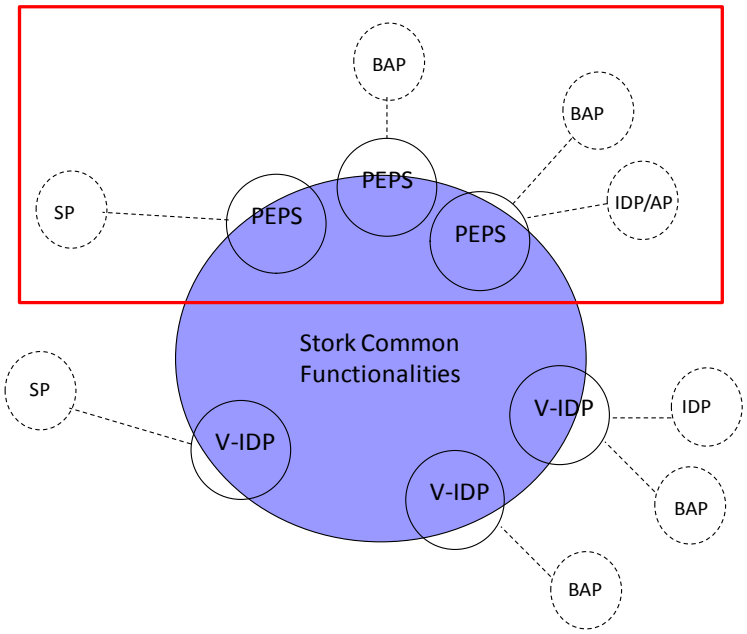


Figure 2: System Context Diagram

In each instance of a PEPS there are 3 roles: the one that attends to SP requests, the one in the country which issued the ID that the citizen wants to use, and the third role is in another country where the user may have domain-specific attributes. Compared with STORK1[1], this last role is new.

For each received request, the first one (S-PEPS) forwards this request to his colleague PEPS or V-IDP, and the second (role of the) PEPS (C-PEPS) resolves the requests received from his colleague PEPS or V-IDP. If in this request any domain-specific attributes (BA) are included, the user is prompted to indicate the Attribute Provider (AP), or alternatively the country in which they can be found. In such latter case, the user is redirected to the A-PEPS of that country, which will allow him to indicate the APs where to retrieve those attributes. The user may indicate several countries to his C-PEPS, however the additional country selection is not supported by the A-PEPS, in order to avoid excessive nesting and making the navigation clearer to the user.

Each PEPS includes the functionalities which are specific to its Member State, which are typically the interfaces with the local ID providers, domain-specific attribute providers and mandate providers. On the other hand, the interface with Service providers (SPs) may also be different from one country to the other one.

The communication between PEPSes and V-IDP and the common functionalities are standard. This blue part of the above diagram, within the red rectangle is object of description in the PEPS chapter of this document. The communication between a PEPS and a V-IDP is the same as between 2 PEPSes. As far as possible V-IDP - V-IDP *communication* will be avoided, i.e. if only countries which implement the decentralised model are involved, just one V-IDP will do the complete job. In such case, instead of the redirection from one PEPS to the other one it uses routing from one module to another one within the V-IDP. However, mixed scenarios between centralised and decentralised (PEPS and MW) countries may require, for reasons of

[1] STORK1: CIP ICT PSP Pilot A project https://www.eid-stork.eu/

trust, a V-IDP - V-IDP communication with redirection which will follow the same protocols and rules as the PEPS-PEPS communication.

2.2 Objectives of PEPSes and V-IDPs

V-IDPs fulfil basically the same objectives as the PEPSes:

- they form anchors of trust which allow to elevate the national circles of trust to European level and
- they hide country specific things like organisation, available ID providers and national and domain-specific attribute providers to the outside world and just offer standardised data through a standardised interface

In this sense, V-IDPs are also described in this chapter, most functions and structures are common to both approaches. The main difference is that V-IDP follows a distributed deployment, fully irrelevant for this document.

However, some important differences exist, which are described in chapter 3.

2.3 Use case view and other requirements

2.3.1 Use case view

2.3.1.1 STORK 2.0 core

The following diagram shows the system level use-cases offered by the core of the STORK 2.0 system.

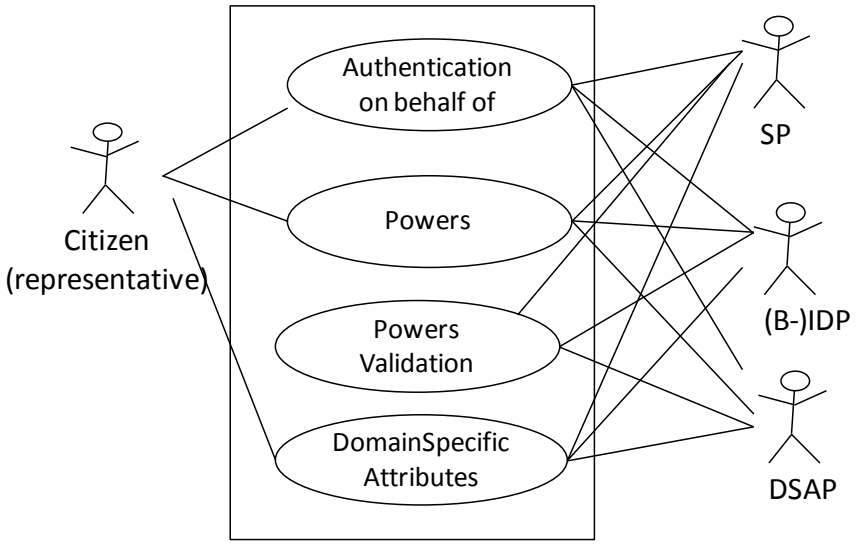


Figure 3: Use case view of the STORK 2.0 core

The functionality of STORK 2.0 is defined by the following processes.

1. **Authentication on behalf of** and **Powers (for digital signatures)** is the process[2] of verifying the identity of a particular representative (user), representing another person. This is achieved by asking for information that proves his identity, as well as the data about the represented persons and mandate for representation. As a result of this process, the user is allowed to access privileged data. Usually this process ends with a fully identified representative (user), represented person and mandate for

[2] Please note that, even though these are different business processes, for the PEPS they are exactly equal.

representation. This means that his eIdentifier, and the identifier of the represented person (eIdentifier or eLPidentifier) is transferred to the SP, and this SP recognises this represented person as a known customer, partner, patient or whatever relationship this person may have with the SP, and recognises the powers of representation of the user.

Note that *Authentication on behalf of* and *Powers* may include collecting the user's domain-specific attributes. Also note that represented person may also be a natural person.

2. **Domain-specific attributes** is the process of verifying the identity of a particular user, and (possibly) collecting additional domain-specific attributes. Functionally the first step is equal to the STORK1 process of Authentication; the second step is new, to be detailed in next paragraph. The standard authentication is achieved by asking for information that proves his identity.
   The second step in the Domain-specific attributes process allows the Service Provider to obtain, through the STORK 2.0 infrastructure and with the collaboration and consent of the citizen, domain-specific attributes stored at national domain-specific attribute provider's sites, and at foreign sites. This functionality is quite the same as the domain-specific attribute retrieval described at previous process.
   As a result of this process, the user is allowed to access privileged data. Usually this process ends with a fully identified user, which means that his eIdentifier and any collection of other personal attributes is transferred to the service provider (SP), and this SP recognises this user as a known customer, student, partner, or whatever relationship this person may have with the SP.
3. **Powers Validation** is the process of verifying that the mandate of a particular user to represent another person is still valid. This process is designed for SPs which maintain a database with the representation-powers, and its users often represent several persons. The process is designed not to include any user-interaction, not even for consent, as all personal data have already been sent to the SP; however whether or not consent is to be requested is configurable.
   This process is only allowed for SPs in which the user is already authenticated using the STORK 2.0 authentication process, and within a certain time-frame. This process is very similar to the AUB process, taking into account that the Single Sign On feature is used.
   As a result of this process, the user is allowed to access privileged data. Usually this process ends with a fully identified user, which means that his eIdentifier and any collection of other personal attributes is transferred to the service provider (SP), and this SP recognises this user as a known customer, student, partner, or whatever relationship this person may have with the SP.

Just like in STORK1, all these use cases may be used for simple authentication of known users as well as registration of new users; both for users on behalf of themselves with domain-specific attributes, as for users on behalf of other users. The difference between authentication and registration lays in the set of attributes requested by the SP.

These four use cases form the core of STORK 2.0. However, other use cases exist, according to the following sections and two diagrams.

2.3.1.2 Version Control

STORK1 is not just one system; it is a platform of connected systems, which are interdependent. The functioning of one system depends directly on the versions of software

and configurations of all other surrounding systems: a change in one of the surrounding systems may provoke any STORK1 node to behave incorrectly.

Furthermore, inclusion of new systems which should not be totally transparent, like new MS nodes, requires intervention in production systems in all STORK1-connected SP-nodes.

STORK 2.0 in this sense is exactly the same, although a bit more complex: more processes, more data. The version control function solves these issues, with the interchange of version control information in an XML formatted file. As described in "D4.9 First version of the Functional Design" [13], in STORK 2.0 the version control has two parts: one part located at the PEPSes and V-IDPs, the other part located at the SPs.

The use case view from the functional design included 2 processes. However, within the scope of the PEPS, the version control for SPs is irrelevant. Nevertheless, SP version control is considered important and common software, so this is described in chapter 4.

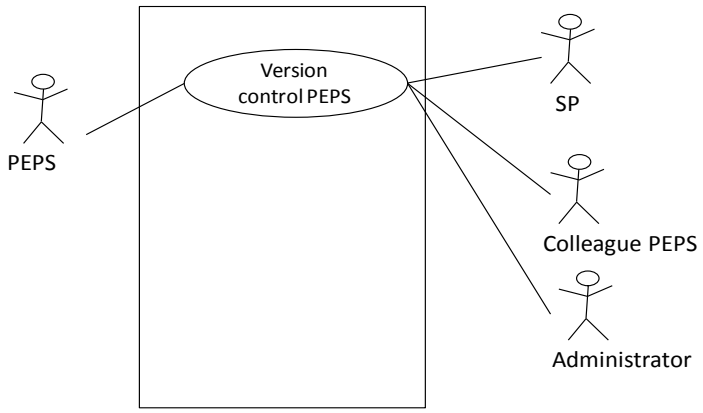


Figure 4: Use case view of the version control

Please note that Version Control for SPs is described in commodities, as it is not part of the common cross-border interface.

2.3.1.3 Anonymity

The main goal of the **Anonymity layer** is to build a system in which trust on delivering is serialized amongst several people/organizations. That is, all of them should plot together to break anonymity; just like on traditional paper based methods, where at least the survey data collector and the processor should collaborate to break anonymity, or on an election, where all the members of a polling station should plot to guess the content of someone's ballot. And even in those scenarios, the participant still can uncover the plot by several means. Thus, the system can be defined as the solution that provides the user participating on an eSurvey with the maximum anonymity with the least impact on usability and the ability to be easily deployed.

The main role of the infrastructure in the anonymity system is to provide the anonymity layer. This layer will dissociate the moment of the input of the packet into the network and the moment the packet finally reaches the Service Provider in which, the results will be considered, making impossible to correlate the participant identity with the output packet of the network.

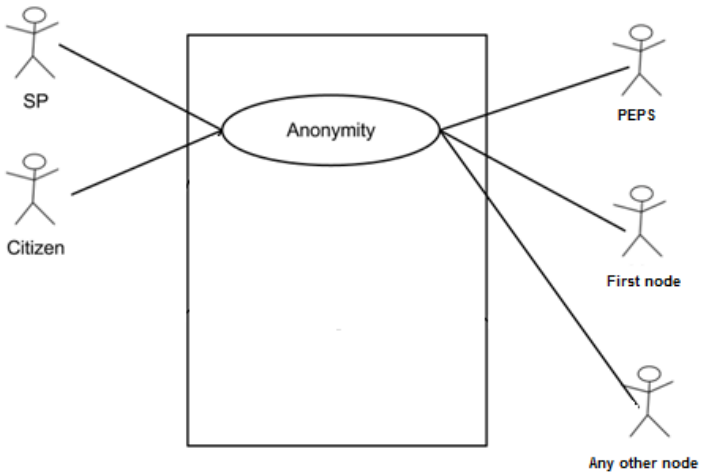


Figure 5: Anonymity system use cases

2.3.1.4 Signatures

STORK 2.0 enhances the Signature-Creation function that has been introduced in STORK1. It allows for more document formats and gives Service Providers more control over the needed signature formats and signature qualities, to better suit the business processes. The new signature functions maintain backward compatibility with STORK1, i.e. continue to use OASIS-DSS as a proven industry standard. The DSS request is further profiled to support the additional functions. This profile has taken PEPPOL (the eProcurement LSP) results into account to support cross-LSP alignment. The document viewer functions are better specified – resulting from STORK1 experience. A signature-verification function is introduced as a supporting service.

Given that several MS have signature infrastructure in place, whereas others do not, the specifications have been drafted so that no assumptions on this infrastructure are made. MS can continue to use their established components, such as signature-creation modules; other can use a STORK 2.0 reference implementation. Results of other LSPs, namely PEPPOL have been taken into account and adopted where suitable.

STORK1 already has the possibility to sign documents, but in limited way. It is only possible to sign documents combined with an authentication request. Once a session is established there is no way of issuing another signature request. While the STORK 1 approach has merit for some business cases (e.g. for signing proof of receipts in eDelivery), it is limiting the potential of this function, as e.g. signed transactions at the end of a process are not supported.

Three ways of issuing a signature request have been specified:

- Combined with an authentication request (as already in STORK 1)
- As a separate request referring an authenticated session (signing a transaction)
- As a request not linked to an authenticated session (signing arbitrary eDocuments)

The two latter requests in fact embrace the same technical request, the difference is just from a business process perspective, i.e. whether the SP requesting a signature needs to have the citizen authenticated.

The overall idea is to delegate the actual signing function to the citizen MS's infrastructure – as is done with handling STORK eID. This means, that the signature-request is issued by the SP and then (via an S-PEPS or V-IDP) routed through the STORK infrastructure to the C-PEPS or V-IDP.

Further this section extends the signature request to gain more control over the signature creation process. It allows the SP to request the signature quality and signature format needed.

2.3.2 Non Functional requirements (NFR)

2.3.2.1 NFR: User Requirements

Name	***Description***
Availability	The system should be designed for high availability, and implemented that way. Thus an availability of 99,7% is to be achieved, see also section 2.3.3.4.
Capacity	The system should be designed for thousands of transactions a day.
Reliability	Stored data cannot be corrupted by defective code, concurrent access, or unexpected process termination.
Performance	A response time of 5 seconds is acceptable for each of the interactions perceived by the user, measured between clicking and the first part of the reply

Table 1 – User requirements

2.3.2.2 NFR: Evolution Requirements

Name	***Description***
Scalability	The system must be designed to scale well with • More countries (we can expect up to some 30) • More IdP's • More Domain-specific attribute Providers • More Business Sectors Very many SP's
Flexibility	The system must allow for personalisation in each of the Member States. This applies of course to the user interface, but also to available data, etc.
Portability	The common code should work • with Tomcat, JBoss, WebLogic and Glassfish under Linux (Ubuntu, RedHat) and Windows 20xx server

Reusability	Components of the system must be created to function and integrate within more than one environment
Extensibility	The system must be prepared to be extended with other functionalities
Maintainability	Although the project has a limited duration, an extension should be foreseen, so the code should obey to normal maintainability criteria.

Table 2 – Evolution requirements

2.3.3 Availability

2.3.3.1 High Availability deployment

The common software is designed and tested to be deployed in a high-availability environment, with a load balancer determining, based on IP and session-id, to which node the request should be sent. Although not any country has deployed this software on more than two nodes, the software is designed to work perfectly with any number of nodes.

As indicated in the STORK performance analysis, also crypto-hardware (HSM) can increase the performance drastically, estimated in a factor of 250% in the performance analysis realised by the STORK project.

2.3.3.2 Cloud deployment

Deployment in "the cloud" would not work however. Several problems would be encountered, in the first place the session-maintenance is done in one server, and would break the authentication-process. Sharing the session-information between all nodes in the cloud would technically not be trivial, and solving the associated data protection problems would not be so either. Other foreseen problems are:

- The random distribution of transaction logs and application traces, which would make it nearly impossible to trace a transaction if an error would be found
- The undefined synchronisation of clocks would cause transactions to fail in an undefined way, as the systems need the clocks to be "synchronised", with a time difference of less than 4,5 minutes
- The Single Sign On (and associated Powers Validation) would have serious problems maintaining all other session-information.

2.3.3.3 Monitoring

In any datacenter the common practice is to use monitoring in order to anticipate any problems of performance; thus all datacenters where a PEPS is deployed have a monitoring of basic system functions in place. Normally these products monitor the use of system resources (CPU, memory, disk usage, net usage, etc). This way the system administrator knows beforehand when machines are getting overloaded. Sometimes also the availability of services on a port is monitored.

However, this monitoring is incomplete in the sense that availability of system resources does not imply automatically that the service is available. Therefore, as a complementary service the AT team has established a more advanced monitoring which sends http requests to all

PEPSes, and reviews the results. If any service is down, the corresponding administrator is warned.

A better monitoring has been thought of, which would send valid SAML requests to all PEPSes and VIDPs and check if at least such request is accepted. Such monitoring service would be a more reliable source for the fact that the PEPS is fully operational. This service would not be hard to build under Jmeter with some custom development. But the project has prioritised on getting the maximum functionality of the central infrastructure and maximum services of the piltos.

2.3.3.4 Resulting availability

Even though a wide variety of measures are taken to ensure high availability, a full, 100,00% availability can never be reached. Downtime can be caused by planned maintenance or incidents. Planned maintenance in a high-availability scenario normally does not cause major unavailability, as the common practice is to first isolate one node and apply and test the changes on this node, before reconnecting it to the pool. Immediately afterwards, the other nodes are disconnected and the changes are applied and tested on these machines. This way the unavailability is rather to be measured in seconds for each maintenance action, which will normally less than once each quarter.

Incidents causing downtime might be less frequent, but as they are not foreseeable, may cause major unavailability. If an incident happens just after working hours, it may cause a downtime of some 15 hours; if this happens once a year an availability of 99,8% would be the resulting availability. Reserving some time for other causes, an availability of 99,7% should be considered as reasonable. In fact in most known professional sites this is considered as the normal requirement.

2.4 Logical view

The main goal of the logical view is the decomposition of the system into subsystems. This can be done by component and/or class diagrams, showing the architecturally important components and their relationships.

The sequence diagrams show the sequence of messages passed between objects using a vertical timeline.

2.4.1 S-PEPS

2.4.1.1 Authentication on behalf of and Powers

The authentication on behalf of process is initiated when a Service Provider needs to know the user's and the represented person's identity as well as mandate data, and sends an *authentication on behalf of* request to the S-PEPS through the citizen's Web Browser. In the same way, the S-PEPS will redirect the authentication request (as a SAML AuthnRequest) to the C-PEPS through the citizen's Web Browser as well.

In D4.2 Functional Design [13] it was explained that both processes are the same, so they are described here at a time.

2.4.1.1.1 Country selector implementation

If the SP does not have the ability to generate SAML messages then it can ask for a SAML AuthRequest to the S-PEPS (Sequence diagram Prerequisite for SP without SAML capacities). During the STORK1 project it has been decided to recommend the implementation of this country selector at Service Providers. There are several benefits doing things this way:

- the PEPS can verify that the Service Provider is authorised;,

- its ProviderName is the correct one, thus avoiding C-PEPSes to ask consent to send data to provider X, when in the end they are sent to provider Y;
- it reduced slightly the load of the PEPS machine;
- it avoids Denial of Service attacks to the PEPS, as requests need to be signed3;

As a summary, this interface is deprecated, i.e. is supported but will disappear in future versions of STORK.

2.4.1.1.2 Sequence diagram Prerequisite for SP without SAML capacities

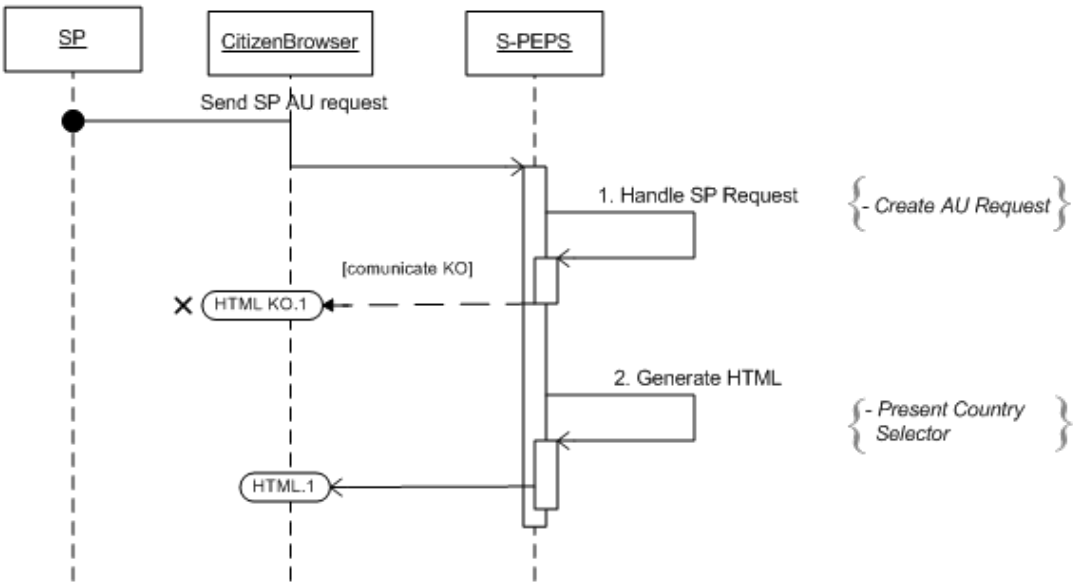
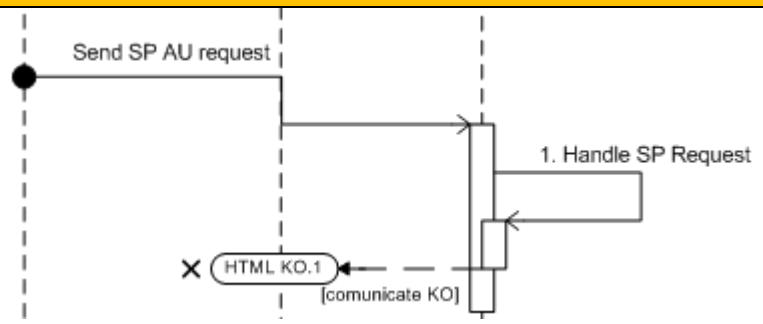
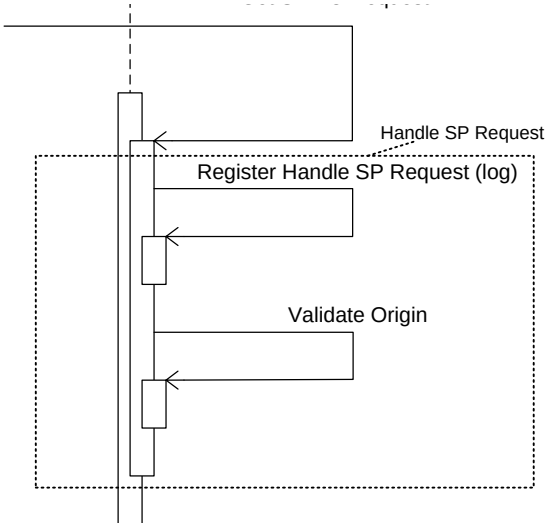
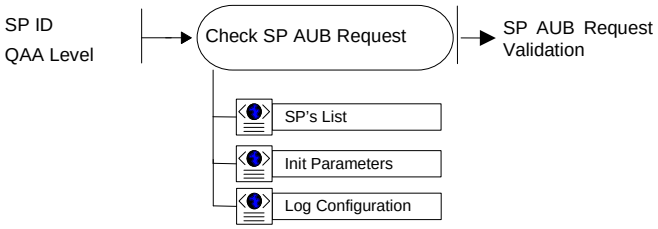


Figure 6: Sequence diagram Prerequisite for SP without SAML capacities

2.4.1.1.3 Description

Message sequences (interactions)		*Description*
1	Handle SP Request	**Description** The objective of this activity is to check the origin of the request and decide if the request, in this first step, is accepted or not.
		Sequence Diagram

[3] If DoS attacks use exactly the same requests, this can be done to the machines, but normally IPSes will filter equal requests.

Message sequences (interactions)	*Description*
	
	Detailed Sequence Diagram 
	External Actors • Citizen-Browser: The S-PEPS receive the Request from a SP through the Citizen-Browser.
	Data  INPUT • SP ID • SP URL • Country of origin (in case a PEPS is shared) • Sector ID • Application name

Message sequences (interactions)		*Description*
		• QAA Level requested by the SP OUTPUT • SP AUB Request Validation CONFIG FILES NEEDED • SP's list: List of SP's allowed to communicate with this PEPS • Init Parameters
		Error Communications • Send HTML KO.1 to the Citizen-Browser. If a Handle Error succeed.
		Actions • Get SP AUB Request ( ) • Register Handle SP AUB Request (SP ID, S-PEPS, "SP AUB Request") • Validate Origin (SP ID, QAA Level). (If SP ID and QAA Level are missing → Handle Error SPAUB0101) o If access control is based on SP's list: ✓ Check SP (SP ID, SP List): SP ID in SP's List. (If SP ID is not in the list → Handle Error SPAUB0102) ✓ Else, check SP Domain (SP Request Domain): validate if the request domain matches the registered SP Domain. (If the origin is not correct → Handle Error SPAUB0103) o Check number of requests (SP ID): number of requests in the last period of 60 seconds. (If this number is greater than or equal to the maximum number - to avoid DoS → Handle Error SPAUB0104)
2	Generate HTML	**Description** Generates the HTML that shows the Country Selector form and gets citizen's nationality selected.

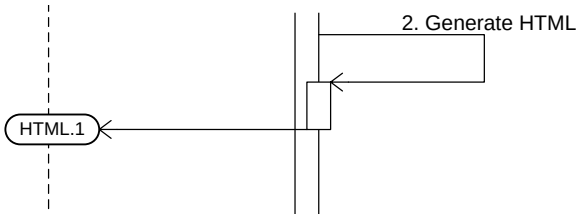
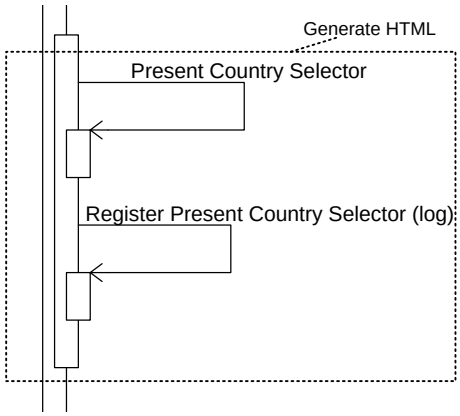
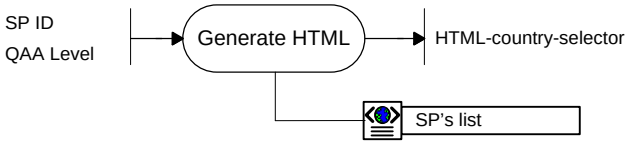
Message sequences (interactions)	*Description*
	Sequence Diagram 
	Detailed Sequence Diagram 
	External Actors •
	Data  INPUT • SP ID • Attributes requested (mandatory/optional), including the requested mandate data • QAA Level OUTPUT • HTML-country-selector CONFIG FILES NEEDED • SP's list (optional)
	Error Communications • None
	Actions • Present Country Selector (SP ID, QAA Level)

Table 3 – Description sequence for SP without SAML capacities

2.4.1.1.4 Sequence diagram AUB

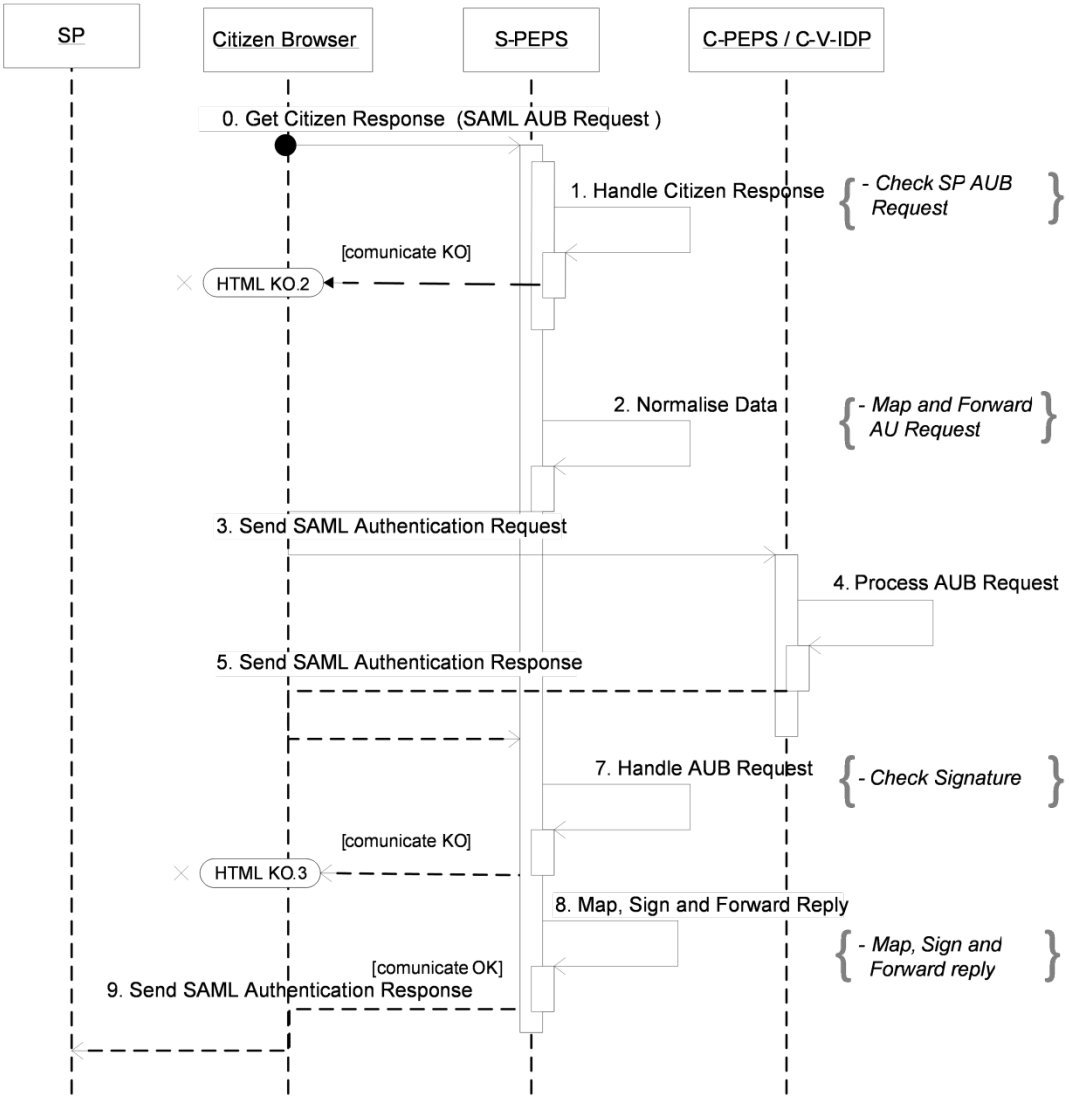
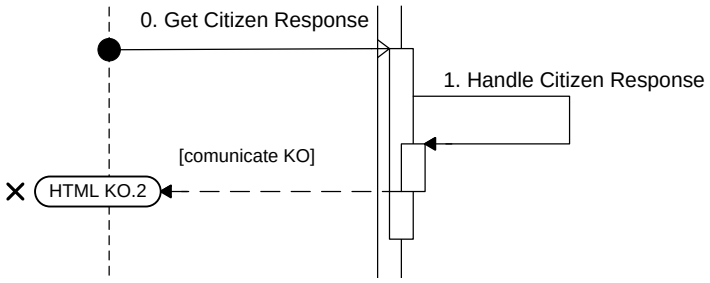
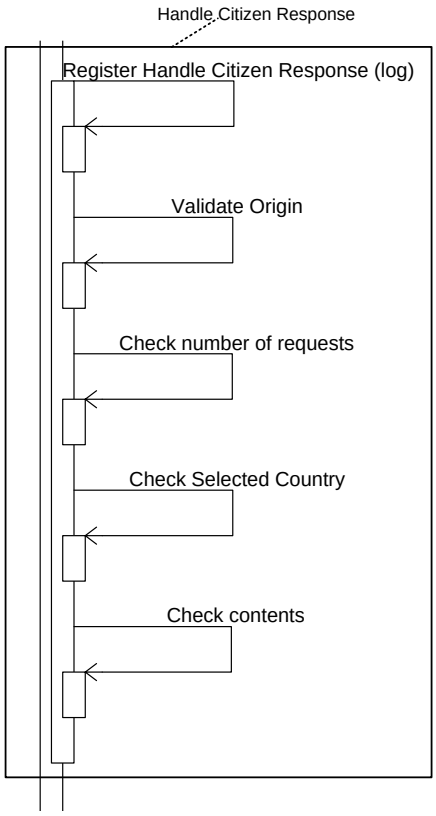
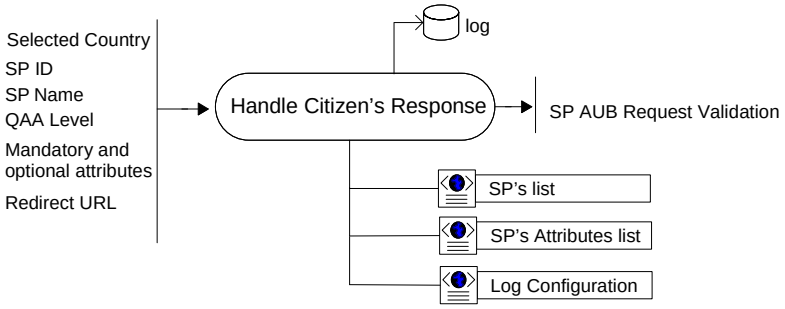


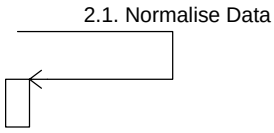
Figure 7: Sequence diagram Authentication on behalf of in S-PEPS

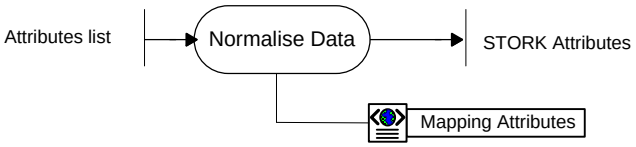
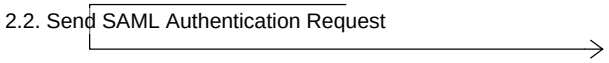
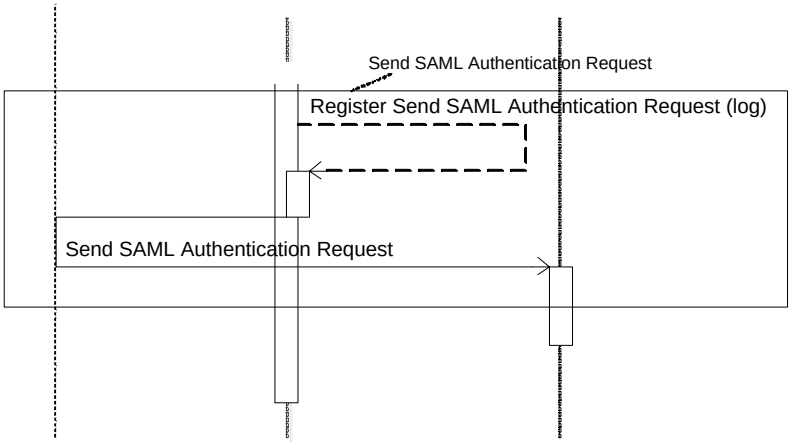
2.4.1.1.5 Description

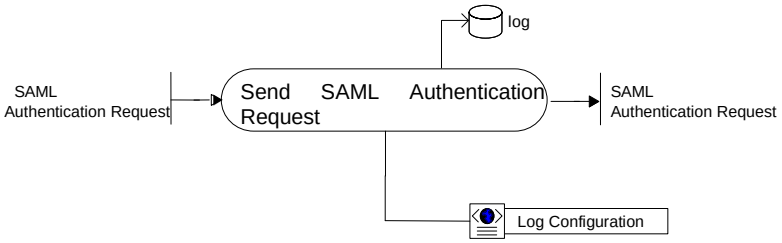
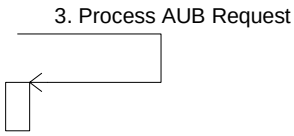
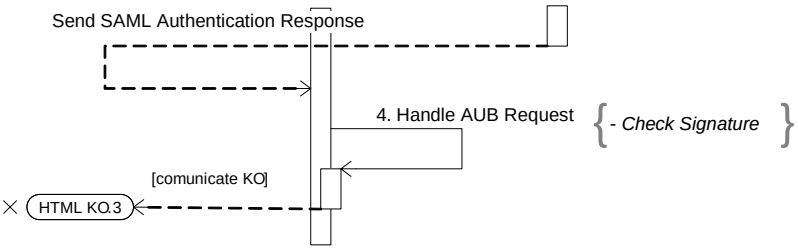
Message sequences (interactions)		*Description*
0-1	Handle Citizen's Response	**Description** Receives Citizen's reply, add it to the AUB Request and check AUB request validation (this task includes log activity).

Message sequences (interactions)	*Description*
	Sequence Diagram 
	Detailed Sequence Diagram 
	External Actors • Citizen-Browser
	Data

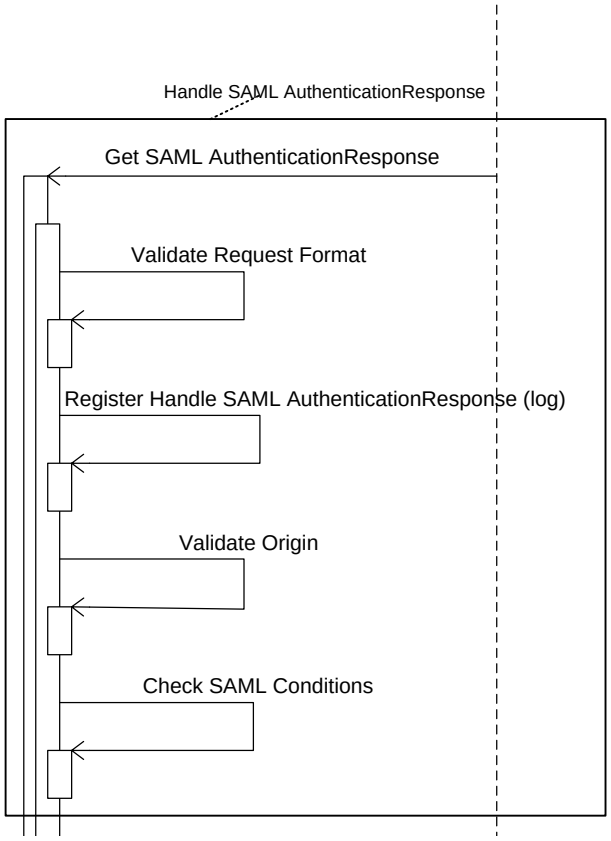
Message sequences (interactions)	*Description*
	 INPUT • Citizen's selected country • SP ID • SP Name • QAA Level • Mandatory and optional Attributes list, including requested mandate data • Redirect URL OUTPUT • SP AUB Request Validation COMMON CONFIG FILES NEEDED • SP's list • SP's Attributes List • Log Configuration
	Error Communications • Send HTML KO.2 to the Citizen-Browser. If a Handle Error succeed.
	Actions • Get Citizen's Country Selected (Citizen's Reply) • Validate Origin (SP ID, QAA Level). (If SP ID and QAA Level are missing → Handle Error SPAUB0401) o If Authentication Type is based on SP's list: ✓ Check SP (SP ID, SP List): SP in SP's List. (If SP is not in the list → Handle Error SPAUB0402) ✓ Else, check SP Domain (SP Request Domain, Redirect

Message sequences (interactions)		*Description*
		URL): validate if the request domain and Redirect URL matches the registered SP Domain. (If the request domain or Redirection URL is not correct → Handle Error SPAUB0403) o Check number of requests (SP ID, 60): number of requests in the last period of 60 seconds. (If this number is greater than or equal to the maximum number - to avoid DoS → Handle Error SPAUB0404) o Check Selected Country (SelectedCountry) (If selected country is not a valid country → Handle Error SPAUB0405) o Check contents (Mandatory and optional Attributes list). (If any Mandatory or optional Attribute isn't in SP's Attributes List → Handle Error SPAUB0406) o Register Handle Citizen Response (Citizen, S-PEPS, "Select Country")
2	Map and Forward AUB Request	**Normalise data and send AUB request to colleague PEPS** This task includes some internal activities to normalise all the received data, log activity and send to Colleague PEPS/V-IDP. 2.1 Normalise data 2.2 Get SAML Authentication Request
2.1	Normalise data	**Description** Normalise all received data, mapping the MS values to STORK nomenclature.
		Sequence Diagram 

Message sequences (interactions)		*Description*
		Data  INPUT • Attribute list with values OUTPUT • Attribute names and data values in STORK nomenclature. CONFIG FILES NEEDED • Mapping Attributes: Map between STORK names/values and MS names/values for the attributes (text values are not included) • Mapping values: as far as values are included in the requested attributes, some may need a semantic translation. This would apply on the legal form of the company in the question "Is this person a legal representative of {"Daimler Benz", "Stuttgart", **AG**, ….}
2.2	Send SAML Authentication Request	**Description** Send AUB Request to Colleague PEPS/V-IDP.
		Sequence Diagram 
		Detailed Sequence Diagram 

Message sequences (interactions)		*Description*
		Data  INPUT • SAML Authentication Request OUTPUT • SAML Authentication Request CONFIG FILES NEEDED • Log Configuration
		Actions • Register Send SAML Authentication Request (S-PEPS, C-PEPS, "Map and Forward AUB Request")
3	Process AUB Request	**Description** The request is processed in the Colleague PEPS/V-IDP. The authentication on behalf of is performed with the user and the security token created.
		Sequence Diagram 
4	Handle AUB Request	**Description** S-PEPS receives SAML AuthenticationResponse through the Citizen-Browser issued by Colleague PEPS/V-IDP. S-PEPS validates SAML AuthenticationResponse signature.
		Sequence Diagram 

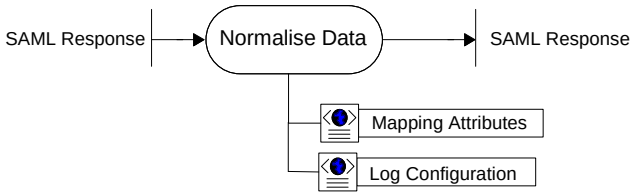
Message sequences (interactions)	*Description*
	Detailed Sequence Diagram



External Actors

- Citizen-Browser: The S-PEPS receives the SAML AUB Response from a Colleague C-PEPS/V-IDP through the Citizen-Browser.

Data

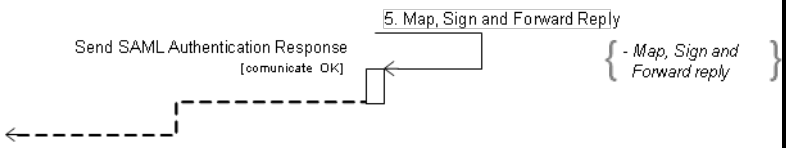


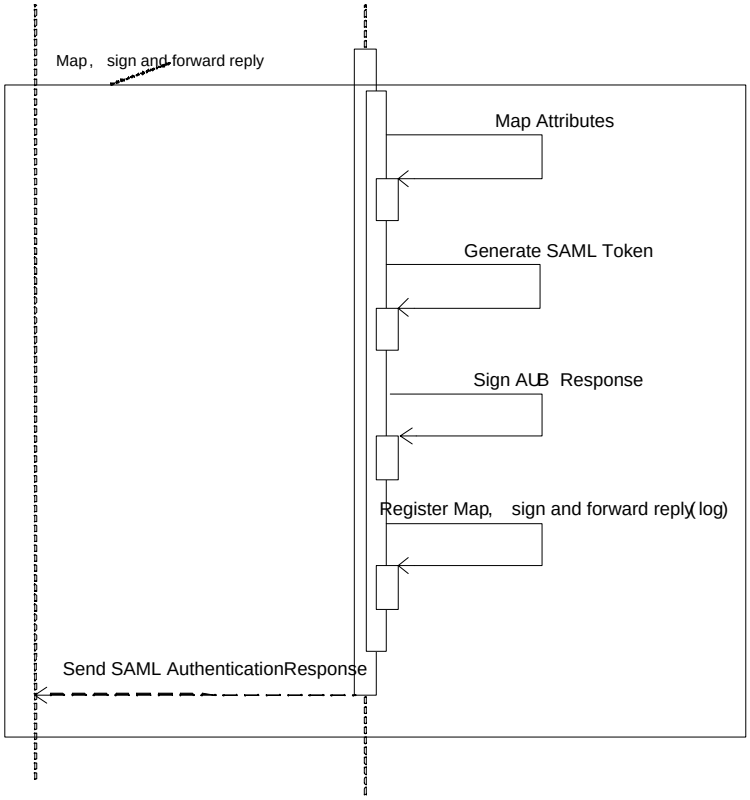
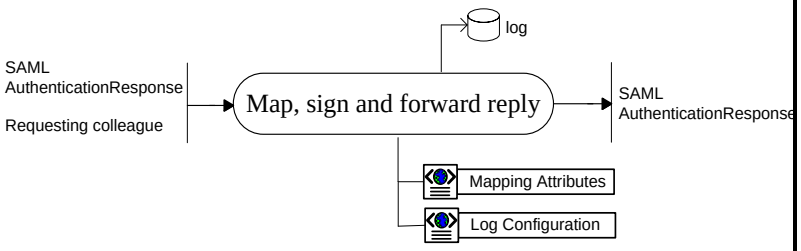
Normalisation of data is the changing of attribute names and values according to the STORK standard to any national standard. This includes translation of common values to national values in the case of attributes defined on an enumerated domain, like the legal form of a legal person, the study one has finalized, etc.

INPUT

- SAML Response

OUTPUT

Message sequences (interactions)		*Description*
		• SAML Response CONFIG FILES NEEDED • Mapping attributes • Log Configuration • Mapping of values
		Error Communications • Send HTML KO.3 to the Citizen-Browser. If a Handle Error succeed.
		Actions • Receive SAML AuthenticationResponse ( ) • Register Handle SAML AuthenticationResponse (C-PEPS, S-PEPS, "Check Signature") • Validate Response Format (SAML) (If the format is not correct → Handle Error SPAUB2201) • Validate Origin (Colleague PEPS/V-IDP): Validate Colleague PEPS/ V-IDP Signature. (If the origin is not correct → Handle Error SPAUB2202) • Check SAML Conditions (notBefore, notAfter, etc.) (If conditions are not fulfilled → Handle Error SPAUB2203)
5	Map, Sign and forward reply	**Description** Maps, generates the SAML token, signs and sends it to SP.
		Sequence Diagram 

Message sequences (interactions)	*Description*
	Detailed Sequence Diagram 
	External Actors • SP
	Data  INPUT • SAML AuthenticationResponse • Requesting colleague PEPS OUTPUT • SAML AuthenticationResponse. CONFIG FILES NEEDED • Mapping Attributes: Map between STORK names/values and MS values for the attributes (text values are not included)

Message sequences (interactions)		*Description*
		• Log Configuration
		Error Communications • None.
		Actions • Map STORK values to MS values (STORK Values, MS Values) • Generate SAML Response (Colleague SAML Request) • Sign SAML Authentication Response (SAML) • Send SAML Authentication Response ( ) • Register Map, sign and forward reply (S-PEPS, C-PEPS, "Map, sign an forward reply") Note that, when mapping attributes coming from A-PEPSes, the original value must be retained, in order to maintain the correctness of signatures, and thus allow SPs to store the message as an evidence chain.

Table 4 – Description sequence Authentication on behalf of in S-PEPS

2.4.1.2 Domain-specific attributes (BA)

The Domain-specific attributes process, similar to the *Authentication on Behalf of*, is initiated when a Service Provider needs to know the user's identity, and sends a *Domain-specific attributes* request to the S-PEPS through the citizen's Web Browser. In the same way, the S-PEPS will redirect the request (as a SAML AuthnRequest) to the C-PEPS through the citizen's Web Browser as well.

Regarding the country selector, for Powers the same applies as for Authentication on behalf of.

2.4.1.2.1 Sequence diagram BA

Same as 2.4.1.1.4.

2.4.1.2.2 Description

Same as 2.4.1.1.5.

2.4.1.3 Signature Creation on Authentication

In this case, the signature creation process takes place during the SAML authentication request. The OASIS-DSS signature request is embedded in a STORK2 <stork:RequestedAttribute> Element. This section describes the signature creation workflow, for authentication request specific details refer to the STORK 1 interface specification [1], the Authentication on Behalf (AUB) specification in this document, respectively.

2.4.1.3.1 Sequence Diagram

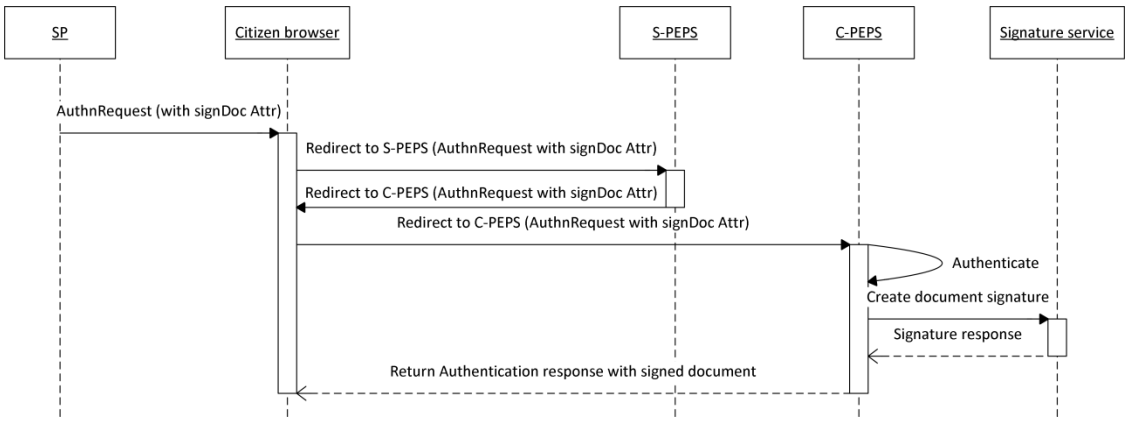
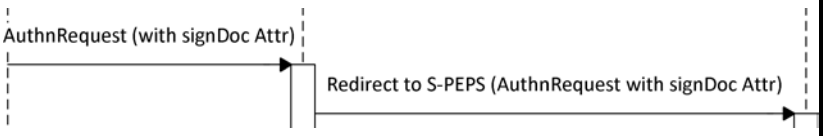
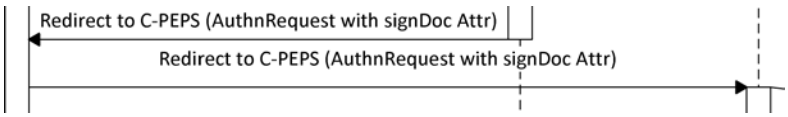
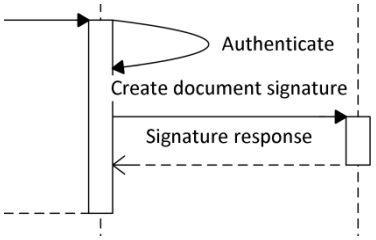


Figure 8: Signature creation on authentication

Figure 8 limits itself to the PEPS-PEPS scenario. For middleware-scenarios, the same sequence diagram applies. Depending on the case, either the SP-request is sent to a V-IDP (instead of an S-PEPS), or finally processed by the V-IDP (instead of C-PEPS).

2.4.1.3.2 Description

Message sequences (interactions)		***Description***
1	Redirect to S-PEPS	**Description** The SP issues an authentication request which contains an *http://www.stork.gov.eu/1.0/signedDoc* attribute and sends it to the S-PEPS (or V-IDP) through the citizen's browser. For details on the authentication request, refer to the the STORK 1 Interface Specifications [1].
		External Actors • Citizen-Browser: The Citizen Browser redirects the Authentication request from the SP to the S-PEPS (or V-IDP).
		Sequence Diagram 
		Data INPUT OUTPUT • SAML authentication request with signedDoc attribute attached
2	Redirect to C-PEPS	**Description** The S-PEPS decides if it can handle the request on its own (not shown in the sequence diagram) or redirects the request through the citizen's browser to the C-PEPS.

Message sequences (interactions)		*Description*
		External Actors • Citizen-Browser: The Citizen Browser redirects the Authentication request from the S-PEPS to the C-PEPS/V-IDP@PEPS.
		Sequence Diagram 
		Data INPUT • SAML authentication request OUTPUT • SAML authentication request (redirected to C-PEPS/V-IDP@PEPS)
3	Create signature	**Description** The C-PEPS (or V-IDP) initiates the document signature. The signature creation process may vary in different MS, depending on the used eId technology (signature-creation may take place at the C-PEPS/V-IDP, or get delegated to a signature service. The C-PEPS finally attaches a *http://www.stork.gov.eu/1.0/signedDoc* to the response which contains the signed data.
		External Actors • The C-PEPS may use external devices or services to issue the MS specific signature.
		Sequence Diagram 
		Data INPUT • SAML authentication request OUTPUT • Issued signature (from signature service)
4	Return results	**Description** The C-PEPS returns the results to the calling instance (citizen's

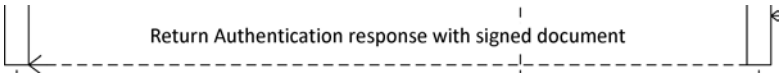
Message sequences (interactions)		*Description*
		browser)
		External Actors • The citizen's browser receives the results of the authentication/signature request.
		Sequence Diagram 
		Data INPUT • Issued signature OUTPUT • SAML authentication response with signedDoc-Response Attribute attached

Table 5 – Description sequence Create signature in S-PEPS

2.4.1.4 Signature Creation with optional Authentication

The former described signature creation method (section 2.4.1.3) requires the SP to issue a SAML authentication request to invoke the signature creation workflows. For business cases where digital signatures are not required during the authentication phase or no authenticated session is required at all, this method is not practicable. Hence, a further signature creation workflow is specified (tightly aligned with the former workflow), which directly uses an HTTP POST enabled OASIS-DSS interface without embedding the request in the SAML request. For a detailed specification on the OASIS-DSS profile refer to [12] and its STORK profile in D4.4 [15].

2.4.1.4.1 OASIS-DSS HTTP POST Transport Binding

To support both a signature request during an authenticated session and as part of non-authenticated sessions, the OASIS "HTTP POST Transport Binding" as specified in section 6.1 of the document [10] is used.

The "TLS Security Binding" with "TLS X.509 Server Authentication" MUST be used (section 6.3.1 of the same document [10])

This transport binding has been chosen, as OASIS-DSS HTTP POST matches with the SAML HTTP POST binding already used by STORK, but it also supports issuing the signature-creation request from non-authenticated sessions.

2.4.1.4.2 STORK 2.0 Integration

As illustrated in Figure 9 the S-PEPS and C-PEPS interfaces get extended to support OASIS-DSS requests. It is required to use the profile specified in section deliverable D4.4 [15] which basically adds optional parameters and is compatible with the OASIS-DSS-core protocol [10].

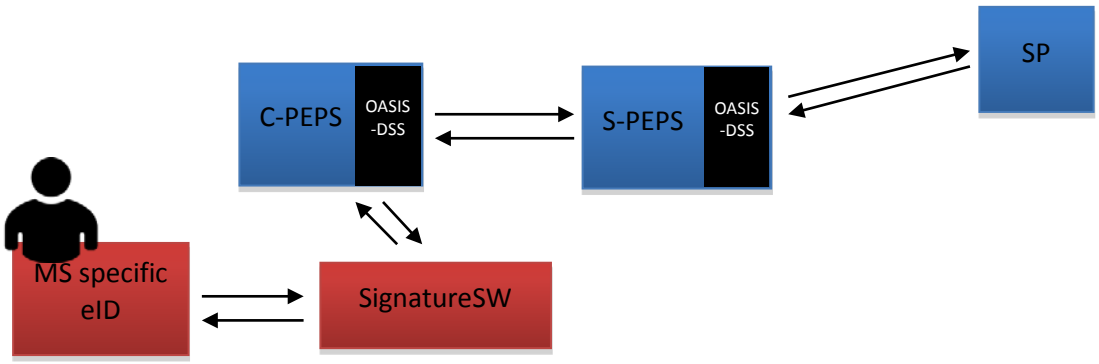


Figure 9: Scheme of signature creation without authentication

The SP sends an OASIS-DSS request to the corresponding S-PEPS interface (or V-IDP), which then decides to redirect the request to the C-PEPS which is aware of the MS specific signature solution. (again, decentralised scenarios can be constructed by, depending on the case, replacing the S-PEPS or the C-PEPS by a V-IDP):

- *Authenticated sessions*: If an authenticated session is available on the SP and signature issuer checking is enabled, the SP checks the issuer of the signature and provides the storkID.
- *Unauthenticated session*: The workflow for signatures in unauthenticated sessions is the same as for authenticated session, except that the SP cannot request binding the issuer of the signature to an authenticated session.

An option "strong identity binding" is introduced that allows SPs better control on whether the same user that authenticated actually signs the document later on. This is done to support high-value services that want to avoid session high-jacking or substitution attacks. The overall process is that the SP may include the "storkID" it got during authentication. The citizen MS infrastructure (C-PEPS or V-IDP) than confirms if the token used to sign corresponds to the storkID. This in many cases can easily be done, as the identifier used to create the storkID is anyhow kept in the certificate (e.g., in Spanish DNIe or Belgian BELPIC), or a reference is given (e.g., identity link in Austria).

Note: While establishing "strong identity binding" can easily be done in several cases, it may not be possible in others or may need re-authentication of the citizen which lowers usability. SPs shall decide based on the risk associated, if the SPs authenticated browser session is sufficient to assume that he same user signs, or if using "strong identity binding" through the STORK infrastructure is advisable. To allow a further granularity, an "*EnforceIdentityBinding*" attribute is used. If set, the STORK infrastructure shall not return signed data, if identity cannot be ensured. If not set by the SP, the signed document is returned, the SP may decide in backend processes depending on the actual risk upon the identity binding result (success, fail, or incomplete).

2.4.1.4.3 Sequence Diagram

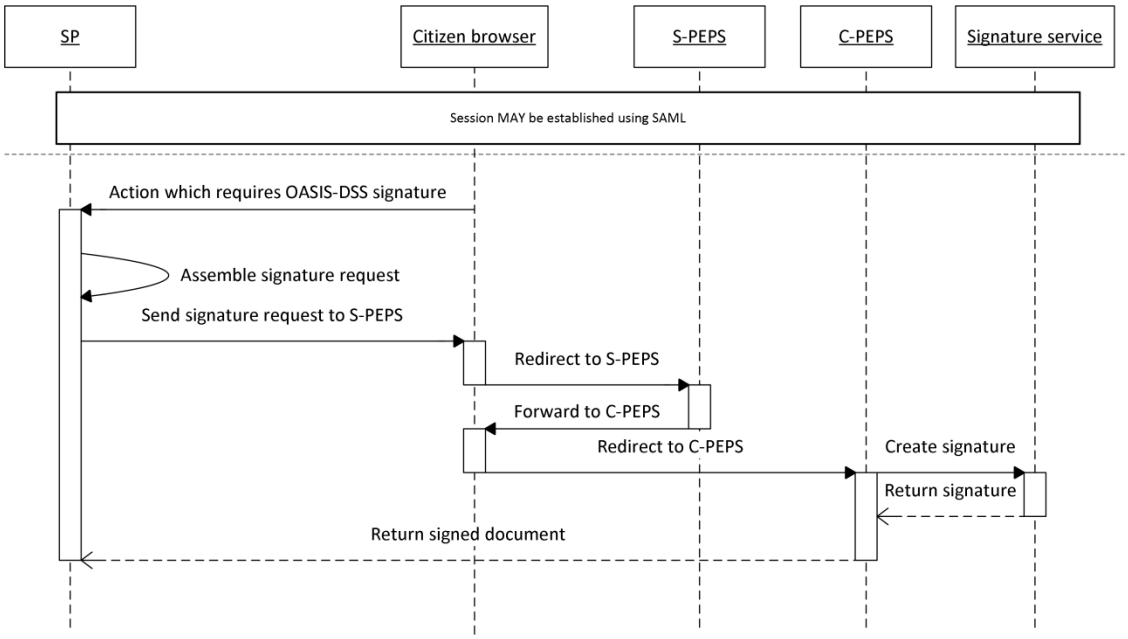
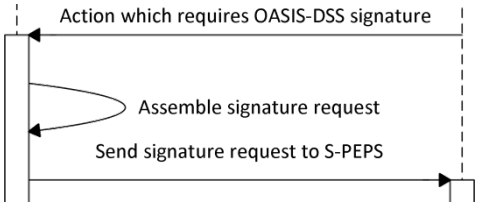
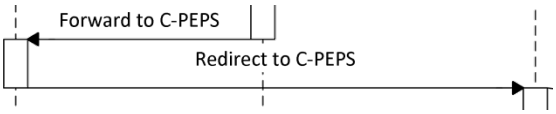
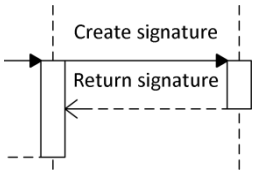


Figure 10: Signature creation with optional authentication

2.4.1.4.4 Description

Message sequences (interactions)		***Description***
1	Generate OASIS-DSS signature request	**Description** The SP issues an OASIS-DSS signature request. The request is sent to the S-PEPS through the citizen's browser.
		External Actors • Citizen-Browser: The Citizen Browser redirects the request from the SP to the S-PEPS.
		Sequence Diagram 
		Data INPUT OUTPUT • OASIS-DSS signature request
2	Redirect to C-PEPS	**Description** The S-PEPS decides if it can handle the request on its own (not shown in the sequence diagram) or redirects the request through the citizen's browser to the C-PEPS/V-IDP@PEPS.

Message sequences (interactions)		***Description***
		External Actors • Citizen-Browser: The Citizen Browser redirects the OASIS-DSS request from the S-PEPS to the C-PEPS/V-IDP@PEPS.
		Sequence Diagram 
		Data INPUT • OASIS-DSS signature request OUTPUT • OASIS-DSS signature request (redirected)
3	Create signature	**Description** The C-PEPS (or V-IDP) initiates the document signature. The signature creation process may vary in different MS, depending on the used eId technology (signature-creation may take place at the C-PEPS/V-IDP@PEPS, or get delegated to a signature service. The C-PEPS generates an OASIS-DSS signature response according to the specification.
		External Actors • The C-PEPS forwards the signature request to the signature service which issues the signature.
		Sequence Diagram 
		Data INPUT • OASIS-DSS signature request OUTPUT • Issued signature (from signature service)
4	Return results	**Description** The C-PEPS returns the results to the calling instance (citizen's browser, SP).
		External Actors

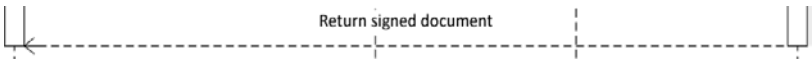
Message sequences (interactions)		*Description*
		• The citizen's browser receives the results of the authentication/signature request.
		Sequence Diagram 
		Data INPUT • Issued signature OUTPUT • OASIS-DSS response

Table 6 – Description sequence Create signature with optional authentication in S-PEPS

2.4.1.5 Document transfer

Document transfer is the process which allows Service Providers to request the user to sign a certain document, redirecting him/her to his national signature creation portal. After creating the signed document, this is returned to the SP. This function supports multiple signatures on one document.

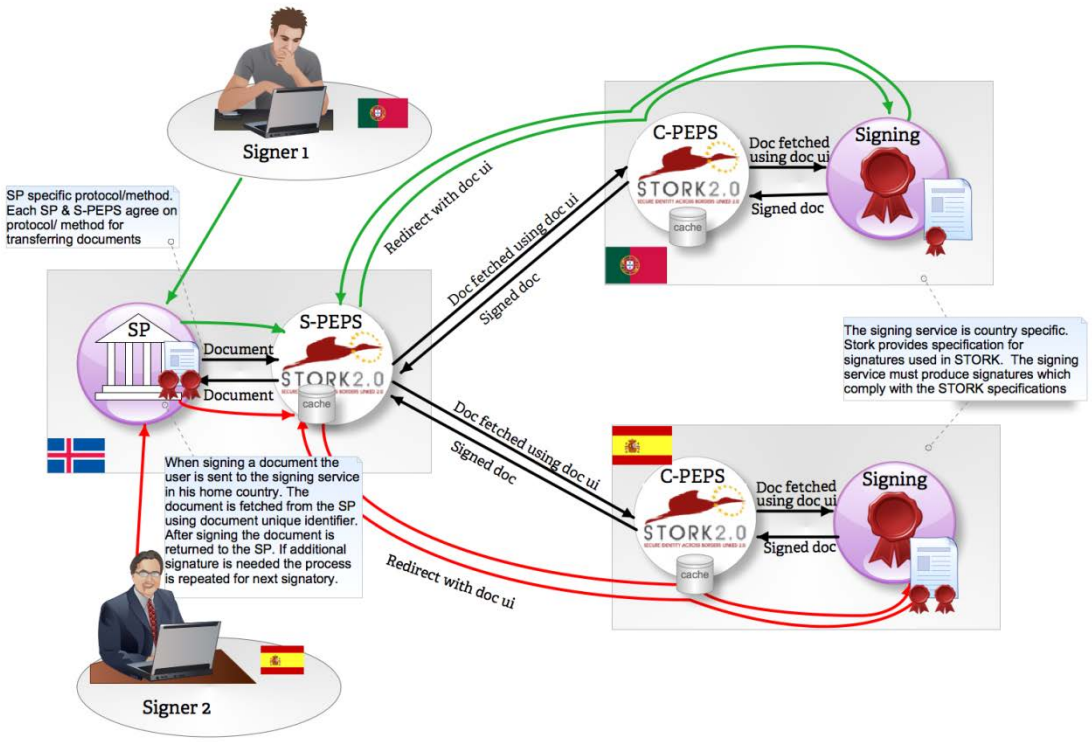


Figure 11: Diagram of a document with two signatures

2.4.1.5.1 Sequence diagram of Document Transfer

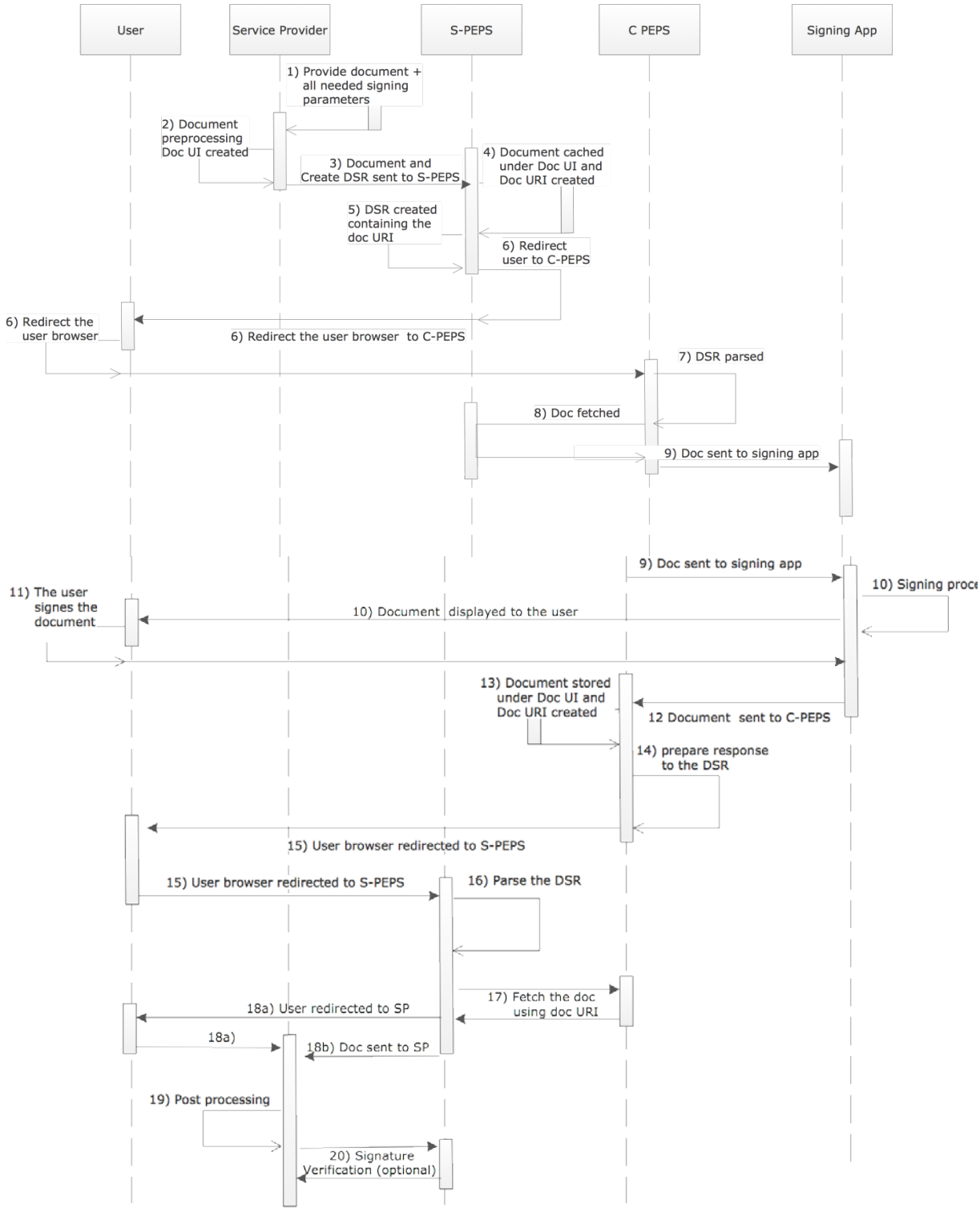


Figure 12: Sequence diagram of document transfer

2.4.1.5.2 Detailed description of Document Transfer

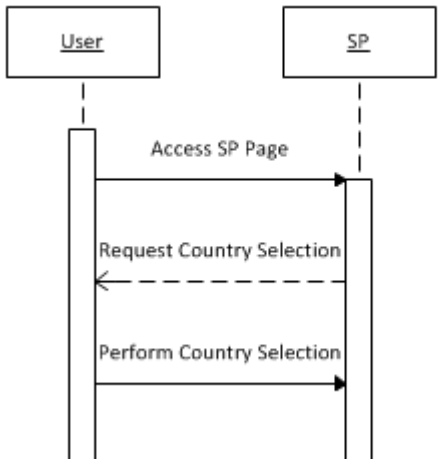
<table>
<tr><th colspan="2">Message sequences (interactions)</th><th>Description</th></tr>
<tr><td>1</td><td>DSR-ACT-1

Create signature</td><td>Description

A presumption for this process is that the service provider has a document which needs to be digitally signed by one or more persons. Where and how this document was created is out of scope for this process. Also the legal preparation of the document text is out of scope, such as clauses on that the document that is digitally signed. The document needs to be prepared and set up for digital signature.

The interaction starts with the user accessing the SP. It is assumed that user has already successfully signed in SP's private area. The service provider must provide all parameters required for signing the document, such as needed information on the signers.</td></tr>
<tr><td></td><td></td><td>External Actors

• Citizen-Browser: The Citizen Browser interacts with SP</td></tr>
<tr><td></td><td></td><td>Sequence Diagram

 
</td></tr>
<tr><td></td><td></td><td>Data

INPUT

• User's country

OUTPUT

• Create Signature Request (CSR) containing OASIS-DSS signature request

COMMON CONFIG FILES NEEDED

• Country profiles</td></tr>
<tr><td></td><td></td><td>Actions

• Validate user's access rights to SP services / authorization
• Gather user's information
• Perform user's country selection</td></tr>
</table>

<table>
<tr><th colspan="2">Message sequences (interactions)</th><th>Description</th></tr>
<tr><td></td><td></td><td>• Prepare document to be signed
• Gather optional information</td></tr>
<tr><td>2</td><td>DSR-ACT-2

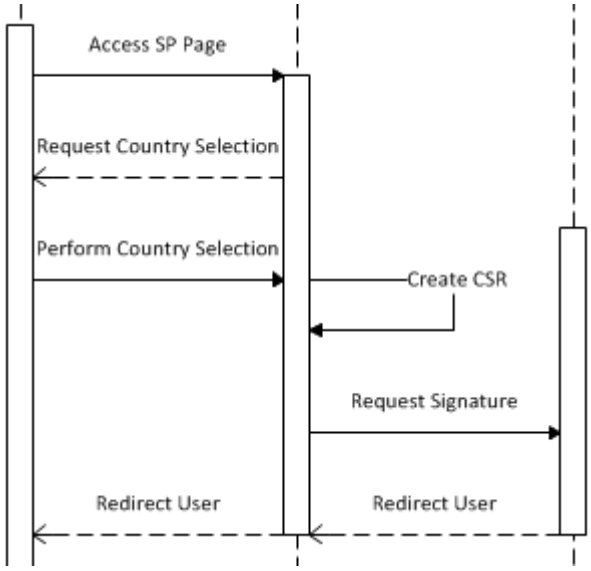
Process signature request</td><td>Description

SP initiates document signing, activates document pre-processing for signing, creates CSR and redirects user to S-PEPS.

Forward Create-Signature-Request (CSR) to S-PEPS (always requesting the signature of one signatory only) and redirect user to the S-PEPS.

The DSR does not contain the payload. Instead, the payload is given by a reference (DocumentURL).</td></tr>
<tr><td></td><td></td><td>External Actors

• Citizen-Browser: The Citizen Browser interacts with SP and S-PEPS</td></tr>
<tr><td></td><td></td><td>Sequence Diagram

 
</td></tr>
<tr><td></td><td></td><td>Data

INPUT

• Citizen’s selected country (the signature fields option)
• The document to be signed
• SP ID
• SP Name
• Redirect URL (SP)

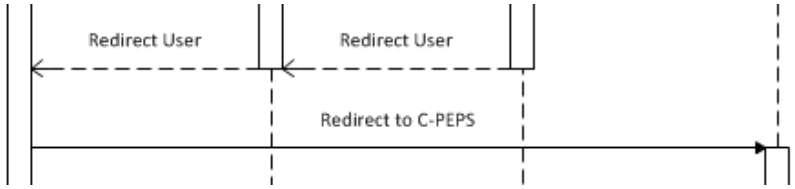
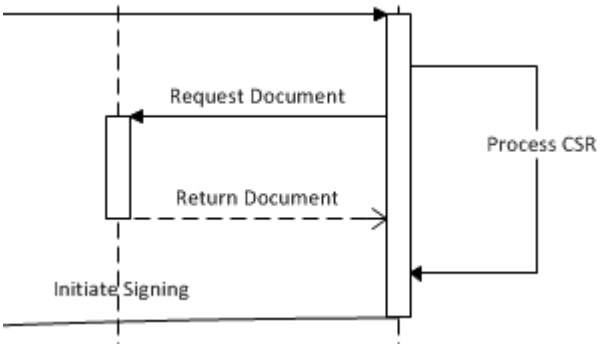
OUTPUT

• CSR containing OASIS-DSS signature request
• Processed document to be signed

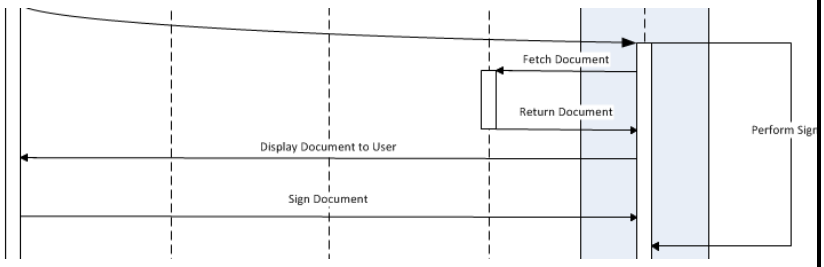
COMMON CONFIG FILES NEEDED

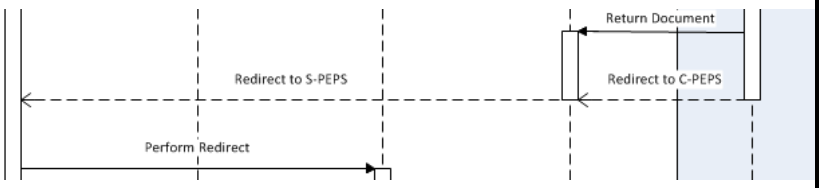
• Version control file (signature fields per MS)</td></tr>
</table>

Message sequences (interactions)		*Description*
		• SP's list • Log configuration
		Actions • Process the document for signing If the document has been signed before, the document has been prepared for signing therefore steps 1 – 3 below be skipped. Using the STORK SP-extensions/Adapters the document must be prepared for signing. 1) Convert the document to the right document format. Structured text should be converted to PDF and simple form to text or XML forms. A detailed specification is given as the STORK Viewer Specification. 2) Save the document using the document unique identifier (d-ui) as the document name. 3) Based on how many signers will sign the document and the origin of the signers, the document must be prepared for signing by creating empty signing boxes for signers from countries where the signing software requires pre generated signing boxes. • Embed current signer information in CSR using the STORK-profile of OASIS-DSS. • Upload processed document to S-PEPS and generate DocumentURL and DocUID, according to the OASIS DSS STORK extension profile. • Prepare redirection to C-PEPS. • Prepare DSR and embed DocUID, DocumentURL. • Validate origin (SP ID) (If SP ID missing → Handle Error SPDTR0401). • Check number of requests (SP ID, 60): number of requests in last period of 60 seconds (If this number is greater than or equal to the maximum number, to avoid DoS → Handle Error SPDTR0404).
3	DSR-ACT-3 Redirect user	**Description** The user is being redirected from S-PEPS to C-PEPS to sign the document at MS signature service. The signature service can by MS-specific operating national eID tokens and signature software. CSR can contain the request for the signature of one signatory only.
		External Actors • Citizen-Browser: The Citizen Browser interacts with S-PEPS and C-PEPS

Message sequences (interactions)		***Description***
		Sequence Diagram 
		Data INPUT • Create Signature Request (CSR)
		Actions • Redirect user to C-PEPS according to previously selected country
4	DSR-ACT-4 Process DSR	**Description** Take/parse request and store the payload (i.e. the document to be signed) in the cache; create an URI pointing to the payload. The user is redirected to C-PEPS. C-PEPS processes CSR, retrieves document to be signed, performs processing if necessary and forwards the user to the MS-specific signature service
		External Actors • Citizen-Browser: The Citizen Browser interacts with S-PEPS and C-PEPS • Signature service – the Member State specific service the user is being redirected to
		Sequence Diagram 
		Data INPUT • Create Signature Request (CSR) • Document to be signed / DocumentURL

Message sequences (interactions)		***Description***
		OUTPUT • Create Signature Request (CSR) • Temporary URI to store the document CONFIG FILES NEEDED • Log configuration • Signature service
		Actions • The data from CSR is extracted. • Create a "complicated" URI to the cached document. • Retrieve document from S-PEPS (If the document cannot be retrieved → Handle Error SPDTR0503). • The document on the S-PEPS is removed after successful retrieval is performed. • The document is stored into temporary storage using the Document ID and C-PEPS specific temporary URI. • DSR is updated with the DocumentURL using new temporary location.
5	DSR-ACT-5 Initiate signing	**Description** Redirect user to the MS signature service to perform document signing.
		External Actors • Citizen-Browser: The Citizen Browser is being forwarded to signature service. • Signature service – the Member State specific service the user is being redirected to.
		Sequence Diagram 
		Data INPUT • Create Signature Request (CSR) OUTPUT • Create Signature Request (CSR)

Message sequences (interactions)		***Description***
		CONFIG FILES NEEDED • Log configuration • Signature service
		Actions • Prepare and perform redirection of the user to Member State specific signature service.
6	DSR-ACT-6 Perform signing	**Description** User signs document using signature service. Optionally, user is able to select the method/application to sign the document with.
		External Actors • Citizen-Browser: The Citizen Browser interacts with signature service
		Sequence Diagram 
		Data INPUT • Create Signature Request (CSR) • Document (DocumentURL) OUTPUT • Create Signature Response (CSR) CONFIG FILES NEEDED • Log configuration • Signature service
		Actions • The CSR is processed • Signature Service retrieves document from C-PEPS and its temporary storage using DocumentURL parameter (If the document cannot be retrieved → Handle Error SPDTR0503). • After the fetching is successfully done, the document is deleted

Message sequences (interactions)		***Description***
		at C-PEPS(If the document cannot be deleted → Handle Error SPDTR0505). • Signature service processes document, if necessary. • The document is displayed to the user (cf. STORK Viewer Specification). • The user signs the document using signature service's facilities and requested signature level. For example if the DSR contains request for NCP signature the signer can sign the document with NCP, NCP+ or QCP certificates. If the DSR request QCP the user can only sign the document using QCP certificate stored on secure signature device.
7	DSR-ACT-7 Return to S-PEPS	**Description** After the document is successfully signed, the document is sent to C-PEPS. The user is redirected to S-PEPS through C-PEPS.
		External Actors • Citizen-Browser: The Citizen Browser interacts with signature service
		Sequence Diagram 
		Data INPUT • Create Signature Response (CSR) • Signed Document OUTPUT • Create Signature Response (CSR) CONFIG FILES NEEDED • Log configuration • Signature service
		Actions • Proving that the user has signed the document, the document is being sent to C-PEPS (If the document cannot be uploaded → Handle Error SPDTR0504)

<table>
<tr><th colspan="2">Message sequences (interactions)</th><th>Description</th></tr>
<tr><td></td><td></td><td>

The document is stored into temporary storage on C-PEPS,
Create a “complicated” URI to the cached document
The payload document is deleted from the signing app (If the document cannot be deleted → Handle Error SPDTR0505).
The user is redirected back to C-PEPS.
C-PEPS updates Create Signature Response with the actual DocumentURL and status, according to STORK OASIS DSS Profile
C-PEPS redirects the user to S-PEPS.

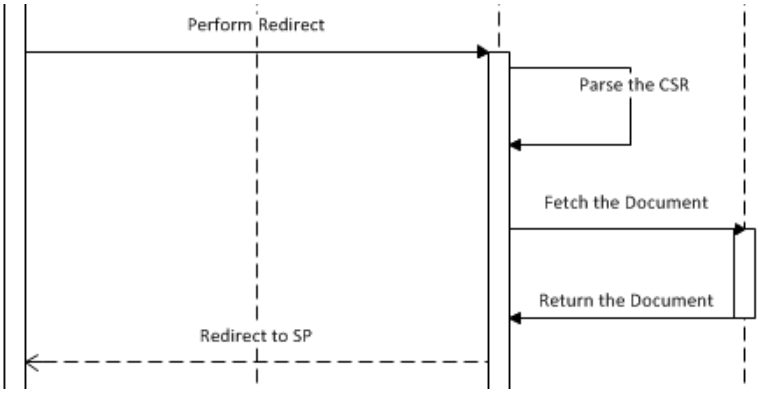
</td></tr>
<tr><td>8</td><td>DSR-ACT-8

Process the signing response</td><td>
Description

The user is sent back to S-PEPS, which processes the Create Signature Response, fetches the signed document using the URL and redirects user to SP.

External Actors

Citizen-Browser: The Citizen Browser interacts S-PEPS and SP

Sequence Diagram


Data

INPUT

Create Signature Response (CSR)
Signed Document

OUTPUT

Create Signature Response (CSR)

CONFIG FILES NEEDED

Log configuration
SP’s list

Actions

The document is fetched from C-PEPS using the DocumentURL

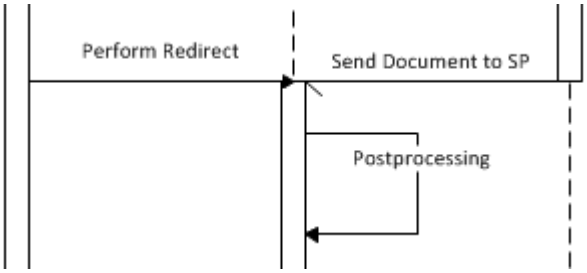
</td></tr>
</table>

<table>
<tr><th colspan="2">Message sequences (interactions)</th><th>Description</th></tr>
<tr><td></td><td></td><td>contained in Document Signing Response (If the document cannot be retrieved → Handle Error SPDTR0503).
• The document is stored into temporary storage at S-PEPS.
• Create a “complicated” URI to the cached document.
• The payload document is deleted from the C-PEPS (If the document cannot be deleted → Handle Error SPDTR0505).
• The DSR is being updated and the user redirected to SP.</td></tr>
<tr><td>9</td><td>DSR-ACT-9

Receive and process signed document</td><td>Description

The user is redirected to SP. SP receives the signed document. After the document is processed by SP, and optionally validated, user is able to consume other services from SP.</td></tr>
<tr><td></td><td></td><td>External Actors

• Citizen-Browser: The Citizen Browser interacts SP</td></tr>
<tr><td></td><td></td><td>Sequence Diagram

 
</td></tr>
<tr><td></td><td></td><td>Data

INPUT

• Create Signature Response (CSR)
• Signed Document

CONFIG FILES NEEDED

• Log configuration
• PEPS configuration file
• SP’s list</td></tr>
<tr><td></td><td></td><td>Actions

• User performs the redirection to SP.
• S-PEPS uploads the document to SP using Document ID and RedirectURL from CSR (If the document cannot be uploaded → Handle Error SPDTR0504).
• The document is removed from the S-PEPS temporary storage after it is successfully sent to SP</td></tr>
</table>

Message sequences (interactions)		*Description*
		(If the document cannot be deleted → Handle Error SPDTR0505). • SP processes the DSR. • SP processes the signed document • SP optionally verifies the signed document (If the document cannot be verified → Handle Error SPDTR0506).

Table 5 – Description sequence document transfer

2.4.1.6 Anonymity

The anonymity is initiated when the user sends his eSurvey, filled in, to the S-PEPS. Please note that, on the contrary of the rest of the STORK 2.0 infrastructure, anyone can send surveys; no mutual trust relation needs to exist. The S-PEPS forwards the request to his colleague PEPSes and V-IDPs, and awaits the results.

2.4.1.6.1 Sequence diagram

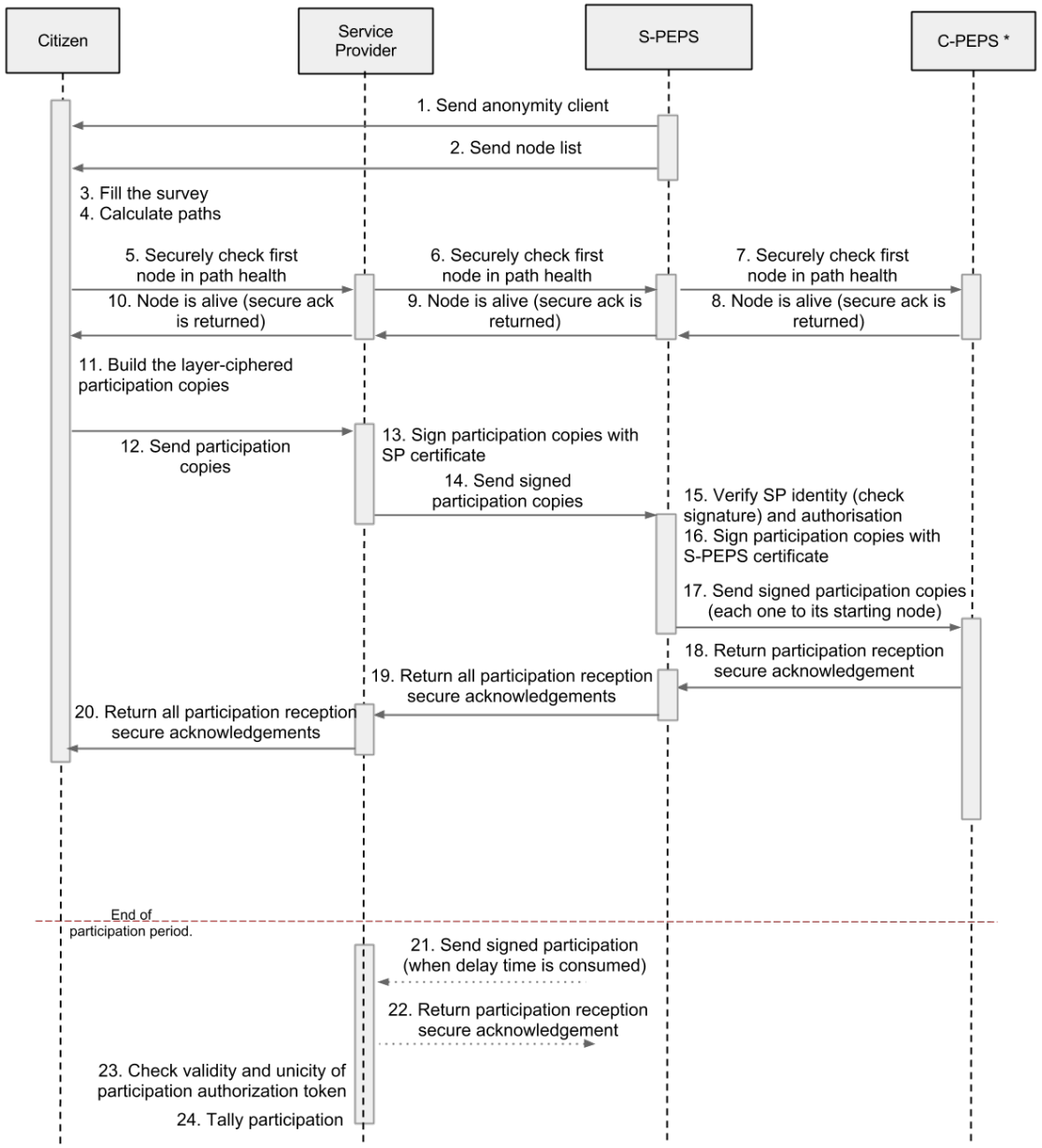
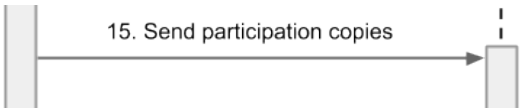
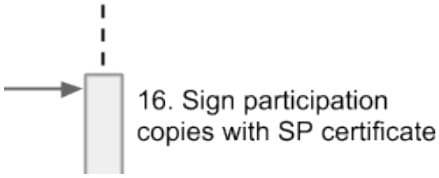


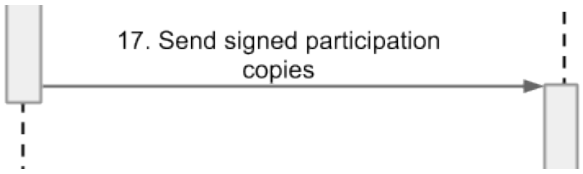
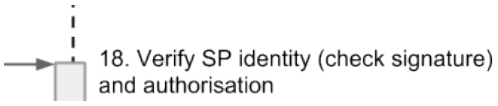
Figure 13: Sequence diagram sending Anonymity of in S-PEPS

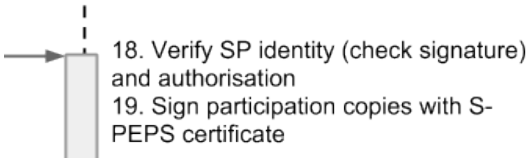
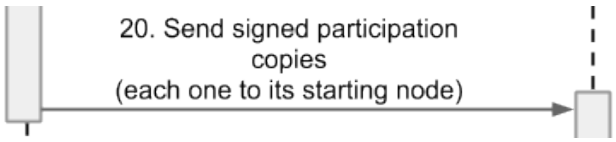
Please note that steps 1-11 are previous to the anonymity, and are only include in the diagram for contextual understanding. The description doesn't describe them, as they are out of scope.

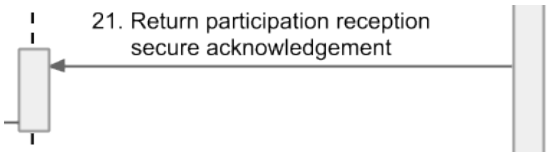
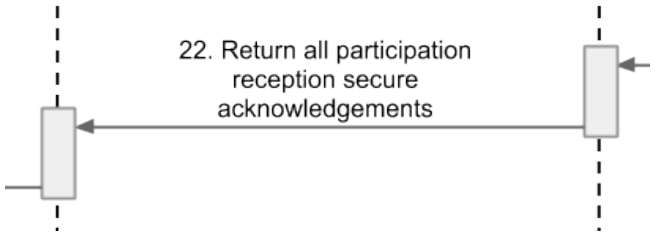
2.4.1.6.2 Description

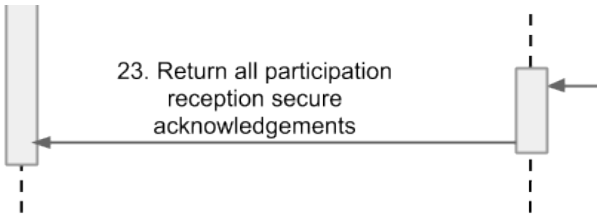
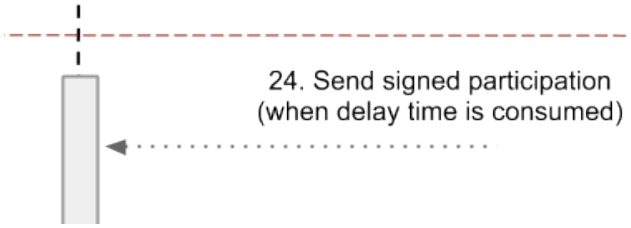
Message sequences (interactions)		***Description***
12	Send participation copies	The client sends the participation packets into the network using an *HTTPS POST* method.
		Sequence Diagram

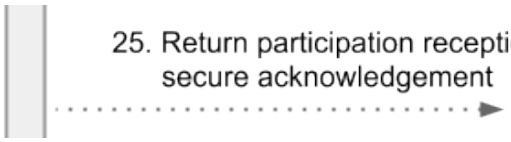
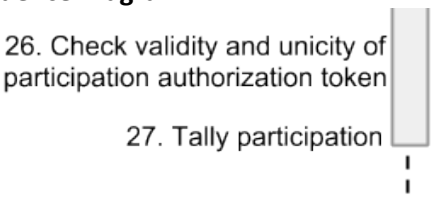
Message sequences (interactions)		***Description***
		
		External Actors • Citizen • SP
		Data INPUT • Layer-ciphered participation copies. OUTPUT •
		Error Communications •
		Actions • Citizen's client sends the packets to the SP
13	Sign participation copies with SP certificate	The *SP* acts as a proxy to the network, it signs each participation using the classic *SAML STORK* profile.
		Sequence Diagram 
		External Actors • SP
		Data INPUT • Layer-ciphered participation copies. OUTPUT • Signed layer-ciphered participation copies.
		Error Communications •
		Actions • The SP signs the layer-ciphered packages.
14	Send signed participation copies	The *SP* bounces the signed participations to the *S-PEPS* using an *HTTPS POST* method.
		Sequence Diagram

Message sequences (interactions)		*Description*
		
		External Actors • SP • S-PEPS
		Data INPUT • OUTPUT • Signed layer-ciphered participation copies.
		Error Communications •
		Actions • SP bounces the packets to the S-PEPS.
15	Verify SP identity (check signature) and authorisation	The *S-PEPS* verifies the *SP* signature to authenticate the source of the ciphered packets. It will also act as a proxy to the network, thus the previous steps (16 and 17) are repeated for the *S-PEPS*.
		Sequence Diagram 
		External Actors • S-PEPS
		Data INPUT • Signed and layer-ciphered packets OUTPUT •
		Error Communications •
		Actions • S-PEPS verifies the signature from the SP.
16	Sign participation copies with S-PEPS	*S-PEPS* signs the participations using its certificate and the classic *SAML STORK* profile.
		Sequence Diagram

Message sequences (interactions)		***Description***
	certificate	
		External Actors • S-PEPS
		Data INPUT • Layer-ciphered participation copies. OUTPUT •
		Error Communications •
		Actions • S-PEPS receives the layer-ciphered participation copies and signs them.
17	Send signed participation copies (each one to its starting node)	The *S-PEPS* bounces the signed participations to each first node on each path using an *HTTPS POST* method.
		Sequence Diagram 
		External Actors • S-PEPS • Each starting node on calculated paths
		Data INPUT • OUTPUT • Signed participations
		Error Communications •
		Actions • The S-PEPS bounces the signed participations to each first node in the path.
18	Return participation reception	Simultaneously the node returns back the Participation Reception Secure Acknowledgement (*PRSA*). This will be returned as a response to the *HTTPS POST* method used by the *S-PEPS*.

Message sequences (interactions)		*Description*
	secure acknowledgement	**Sequence Diagram** 
		External Actors • S-PEPS • Each starting node on calculated paths
		Data INPUT • Layer-ciphered packet OUTPUT • Secure ACK
		Error Communications •
		Actions • The node returns the participation reception secure acknowledgement.
19	Return all participation reception secure acknowledgements	The *S-PEPS* will concentrate all the *PRSA* and will send back to the *SP* as a response to the *HTTPS POST* method used in step 17.
		Sequence Diagram 
		External Actors • Each starting node on calculated paths • S-PEPS
		Data INPUT • OUTPUT • All participation reception secure acknowledgements
		Error Communications •
		Actions • S-PEPS receives all the secure ACKs and bounces back to the SP.

Message sequences (interactions)		***Description***
20	Return all participation reception secure acknowledgements	The *SP* does the same as the *S-PEPS* and bounces back the *PRSA* to the citizen's client as a response to the *HTTPS POST* method used in step 15. The citizen's client checks the challenges match the expected ones.
		Sequence Diagram 
		External Actors • S-PEPS • SP
		Data INPUT • OUTPUT • All participation reception secure acknowledgements
		Error Communications •
		Actions • The SP receives all the secure ACKs and bounces it to the client.
21	Send signed participation (when delay time is consumed)	The last node in the path sends the signed participation to the SP as a final step.
		Sequence Diagram 
		External Actors • Last node in each path • SP
		Data INPUT • Signed participation OUTPUT •

Message sequences (interactions)		***Description***
		Error Communications •
		Actions • Last node in the path sends the signed participation to the SP.
22	Return participation reception secure acknowledgement	The participation is also ciphered for the SP as if it was a node thus; the SP deciphers the *RC4* key with its *RSA* private key and the participation contents with the *RC4* key. The last layer of ciphering includes the survey results themselves.
		Sequence Diagram 
		External Actors • Last node in the path • Penultimate node in the path
		Data INPUT • OUTPUT •
		Error Communications •
		Actions • The SP acts also as the end-point of the paths so the package is also ciphered to it. It deciphers the participation and send back the secure ACK.
23	Check validity and unicity of participation authorisation token	The *SP* checks the validity (check the token's signature) and unicity (check no other results with the same token have been counted in) of the results.
		Sequence Diagram 
		External Actors • SP

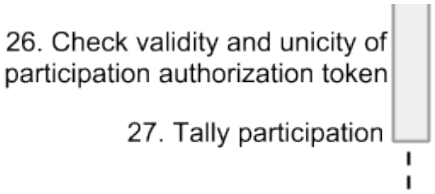
Message sequences (interactions)		*Description*
		Data INPUT • The deciphered participation OUTPUT •
		Error Communications •
		Actions • The SP checks the validity and unicity of this given result set
24	Tally participation	Finally the *SP* tallies the participation to be part of the general survey results.
		Sequence Diagram 
		External Actors • SP
		Data INPUT • Plain participation OUTPUT •
		Error Communications •
		Actions • The SP counts the participation to be part of the general results.

Table 7 – Description sequence Anonymity in S-PEPS

2.4.2 C-PEPS

The C-PEPS is the PEPS in the role of verification of citizen's credentials and obtaining additional data, e.g. from the represented person and mandates. This role is also composed of three business processes:

Authentication on behalf of

Powers (for digital signature)

Domain-specific attributes

However, in the functional design it was found out that first two processes are the same in a PEPS, so both are described in one section.

A consecutive section describes the powers validation process.

2.4.2.1 Authentication on behalf of and Powers

The authentication on behalf of process is initiated when a Service Provider needs to know the user's and the represented person's identity as well as mandate data, and sends an *authentication on behalf of* request to the S-PEPS through the citizen's Web Browser, which the S-PEPS will redirect to the C-PEPS through the citizen's Web Browser as well. As far as additional (business) attributes are requested from foreign countries, these are requested from the A-PEPS.

2.4.2.1.1 Sequence diagram AUB, part 1

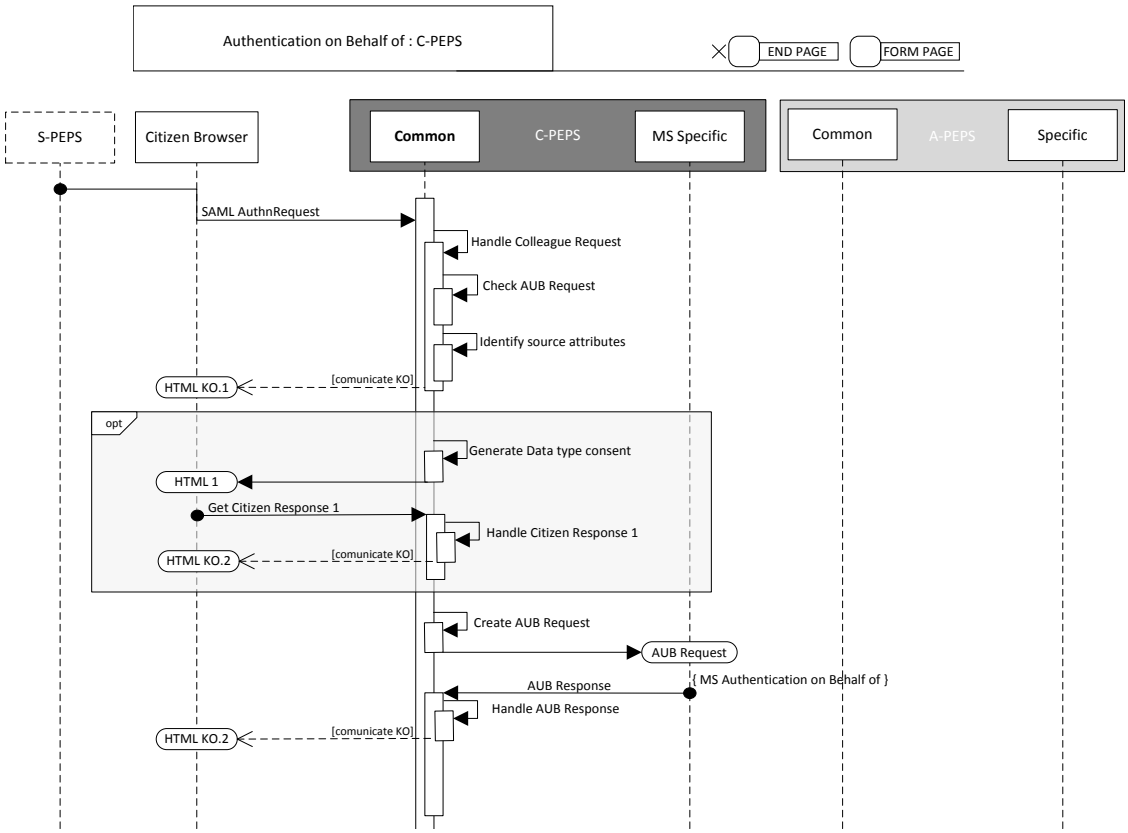
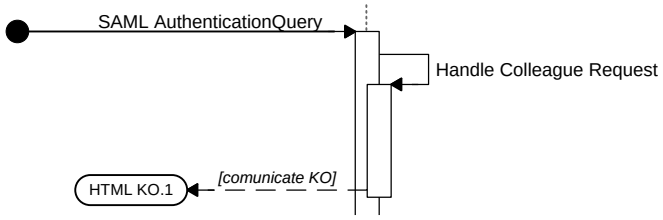
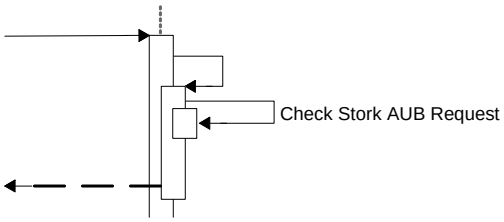
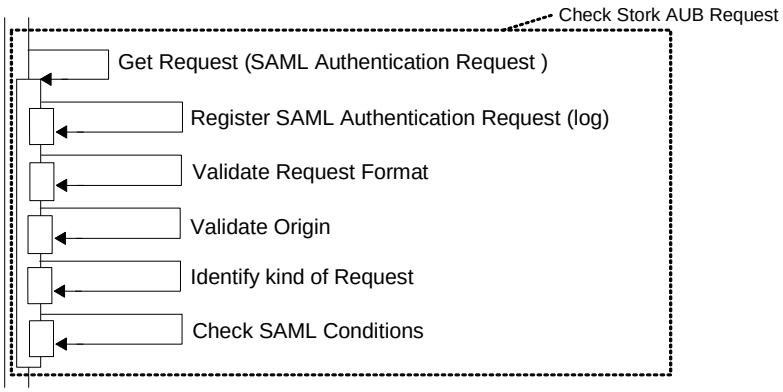
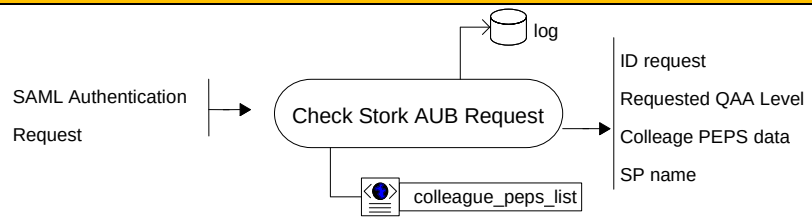


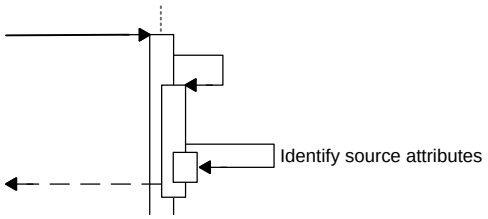
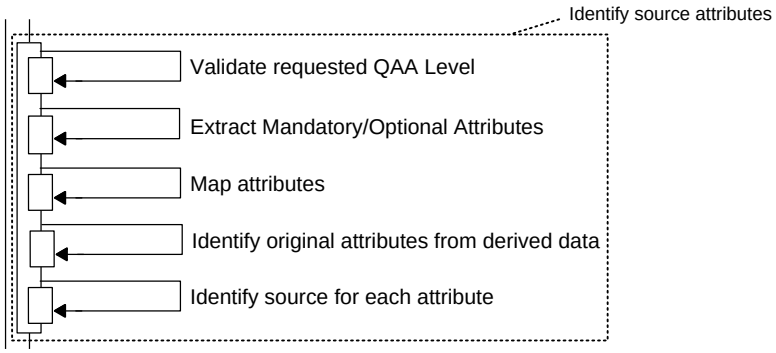
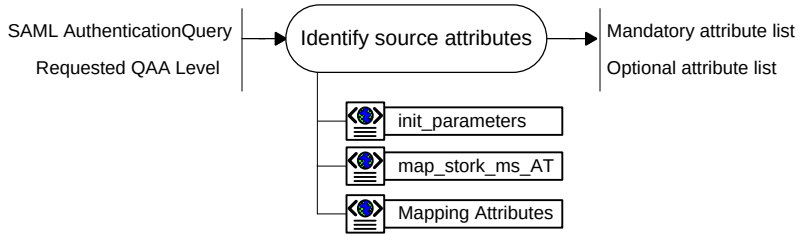
Figure 14: Sequence diagram Authentication on Behalf of, Part 1, in C-PEPS

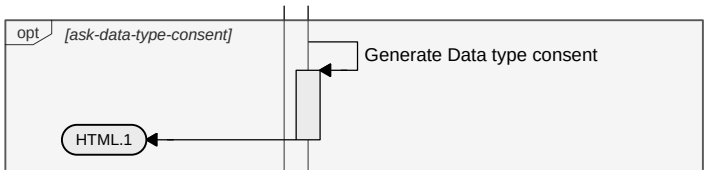
2.4.2.1.2 Description AUB, part 1

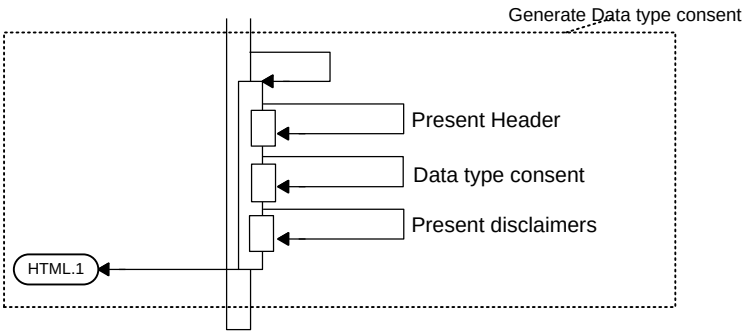
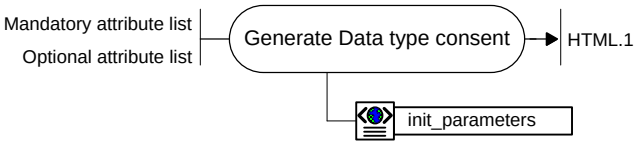
Message sequences (interactions)		***Description***
1	Handle Colleague Request	**Description** Handles request of colleague PEPS or V-IDP This task includes some internal activities to validate the request received and prepare the next steps towards the user authentication. 1.1 Check Authentication Request 1.2 Identify source attributes

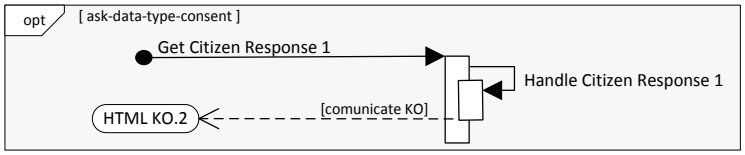
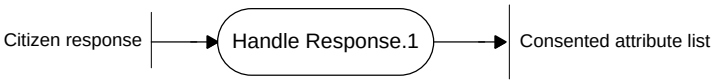
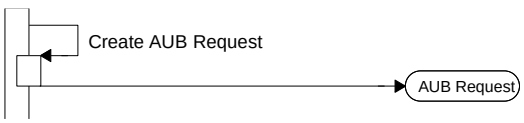
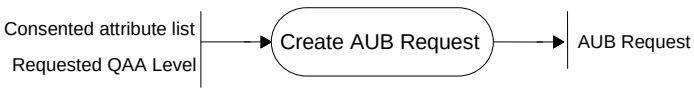
Message sequences (interactions)		*Description*
		External Actors • Citizen-Browser: The C-PEPS/V-IDP receives the Request from a colleague PEPS or V-IDP through the Citizen-Browser.
		Sequence Diagram 
		Error Communications • Send HTML KO.1 to the Citizen-Browser. (If an error succeeds a Handle Error function manages this error).
1.1	Check authentication request	**Description** The objective of this activity is to register the request, check the origin of the request and decide if the request, in this first step, is accepted or not.
		Sequence Diagram 
		Detailed Sequence Diagram 
		Data

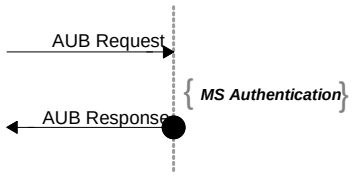
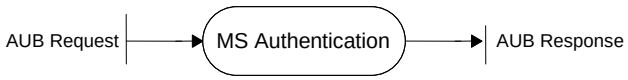
Message sequences (interactions)		***Description***
		 INPUT • SAML Authentication Request OUTPUT • ID request • Requested QAA Level • Colleague PEPS data • SP name CONFIG FILES NEEDED • colleague_peps_list: List of PEPS and V-IDP colleagues that form STORK 2.0
		Actions • Get Request (SAML Authentication Request): Captures any Request from Citizens browser send to the C-PEPS. • Register Request (SAML Authentication Request): Log the SAML request received. • Validate Request Format (SAML) (If the format is not correct → Handle Error CPAUB0101) • Validate Origin (Colleague PEPS): Validate Colleague PEPS Signature. This data is available in the colleague peps list file. (If the origin is not correct → Handle Error CPAUB0102) • Identify kind of Request: Extract Request and identify kind of Request (Authentication Request, AttributeQuery, etc). In this activity we analyse the Authentication Request case. • Check SAML Conditions (notBefore, notAfter, etc.) (If conditions are not met → Handle Error CPAUB0103)
1.2	Identify source attributes	**Description** This activity checks the validity of the request in terms of contents.
		Sequence Diagram

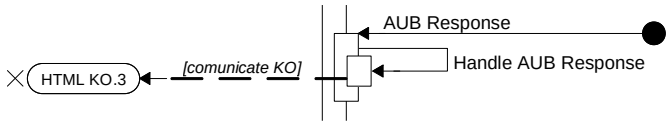
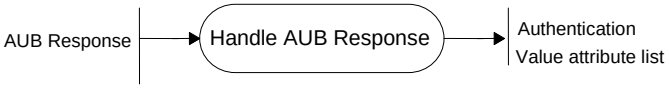
Message sequences (interactions)	*Description*
	
	Detailed Sequence Diagram 
	Data  INPUT • SAML AuthenticationQuery • Requested QAA Level OUTPUT • Mandatory requested attributes in MS Attribute nomenclature • Optional attributes in MS Attribute nomenclature CONFIG FILES NEEDED • init_parameters (initial C-PEPS parameters) • map_stork_ms_attributes (STORK attributes mapped to MS attributes)
	Actions • Validate requested QAA Level: Compare requested QAA Level with the C-PEPS max QAA Level.

Message sequences (interactions)		*Description*
		(If max level is lower than requested → Handle Error CPAUB0104) • Extract mandatory/optional attributes: Identify the mandatory attributes. • Map attributes: Transform and derive national requested attributes in STORK requested attributes. This transformation may include the translation of values included in the request. • Identify original attributes from derived data: This attributes will be added to the attributes to be requested to the national IDPs, APS. This attributes may be also mandatory or optional. They inherit this characteristic from the derived data. • Identify source for each attribute: For each MS attribute obtained, the national source that can disclosure this data is identified: o available o not available (If there is "not source available" for a mandatory attribute → Handle Error CPAUB0105)
2	Generate Data type consent (optional)	**Description** • This task includes some internal activities to generate the page in which the user is requested to give his consent to the transfer of his data (types). It is optional, only it is executed if the ask-data-type-consent is set to YES in the *init_parameters* config file.
		External Actors • Citizen-Browser: Receives a generated HTML
		Sequence Diagram 
		Detailed Sequence Diagram

Message sequences (interactions)		*Description*
		
		Data  INPUT • Requested attributes (mandatory / optional) in C-PEPS's native language OUTPUT • HTML.1 CONFIG FILES NEEDED • init_parameters
		Error Communications • None
		Actions • Evaluate ask-data-type-consent from init parameters file. • Present header (SP name) • Data type consent: Present attribute list (requested attributes: mandatory and optional) and gets citizen's consent. Mandatory attributes cannot be deselected. • Present disclaimers
3	Handle Citizen Response.1 (optional)	**Description** Receives citizen's reply.

Message sequences (interactions)		***Description***
		Sequence Diagram 
		External Actors • Citizen
		Data  INPUT • Citizen reply OUTPUT • Consented attribute list (mandatory and optional) CONFIG FILES NEEDED • None
		Actions • Register citizen reply (log citizen's consent) • Check citizen reply format (If the response is malformed → Handle Error CPAUB0301) • Check consents (If the consent is not given for a mandatory attribute → Handle Error CPAUB0302)
		Error Communications • Send HTML KO.2 to the Citizen-Browser. (If a Handle Error succeed).
4	Create AU Request	**Description** Create an AUB Request and send it to the "MS Authentication".
		Sequence Diagram 
		Data 

Message sequences (interactions)		***Description***
		INPUT • Consented attribute list (mandatory and optional) • Requested QAA Level OUTPUT • Authentication Request CONFIG FILES NEEDED • init_parameters: (localization of the MS Authentication Module)
		Actions • Create Authentication Request • Send Authentication Request to MS Authentication Module
5	MS Authentication	**Description** Select and Perform Authentication in the IDP/NAP. Receive an Authentication Request and return an Authentication Response.
		Sequence Diagram 
		Data  INPUT • AUB Request o Consented Attribute list o Requested QAA Level OUTPUT • AU Response o Authentication (yes/no) o Value Attribute list CONFIG FILES NEEDED • None

<table>
<tr><th colspan="2">Message sequences (interactions)</th><th>Description</th></tr>
<tr><td></td><td></td><td>Actions
Is MS specific.
Authentication on behalf of is a country specific process. Within this process, some of the requested attributes may be collected.</td></tr>
<tr><td>6</td><td>Handle AUB Response</td><td>Description
Handles an authentication response.</td></tr>
<tr><td></td><td></td><td>Sequence Diagram
 
</td></tr>
<tr><td></td><td></td><td>Data
 

INPUT
• AUB Response
OUTPUT
• Authentication (yes/no)
• Value attribute list
o Filled attribute list: attributes with found values
o Empty attribute list: attributes without values
CONFIG FILES NEEDED
• None</td></tr>
<tr><td></td><td></td><td>Actions
• Check AUB Request
(If the authentication fails→ Handle Error CPAUB0601)
• More attributes?: Evaluate if all attributes needed are available or if more attributes should be requested to MS Attribute Supply.
(Also evaluate if some attributes has to be introduced by the user and verified)</td></tr>
<tr><td></td><td></td><td>Error Communications
• Send HTML KO.3 to the Citizen-Browser.
(If a Handle Error succeed).</td></tr>
</table>

Table 8 –Description sequence Authentication on Behalf of, part 1, in C-PEPS

2.4.2.1.3 Sequence diagram AUB, part 2

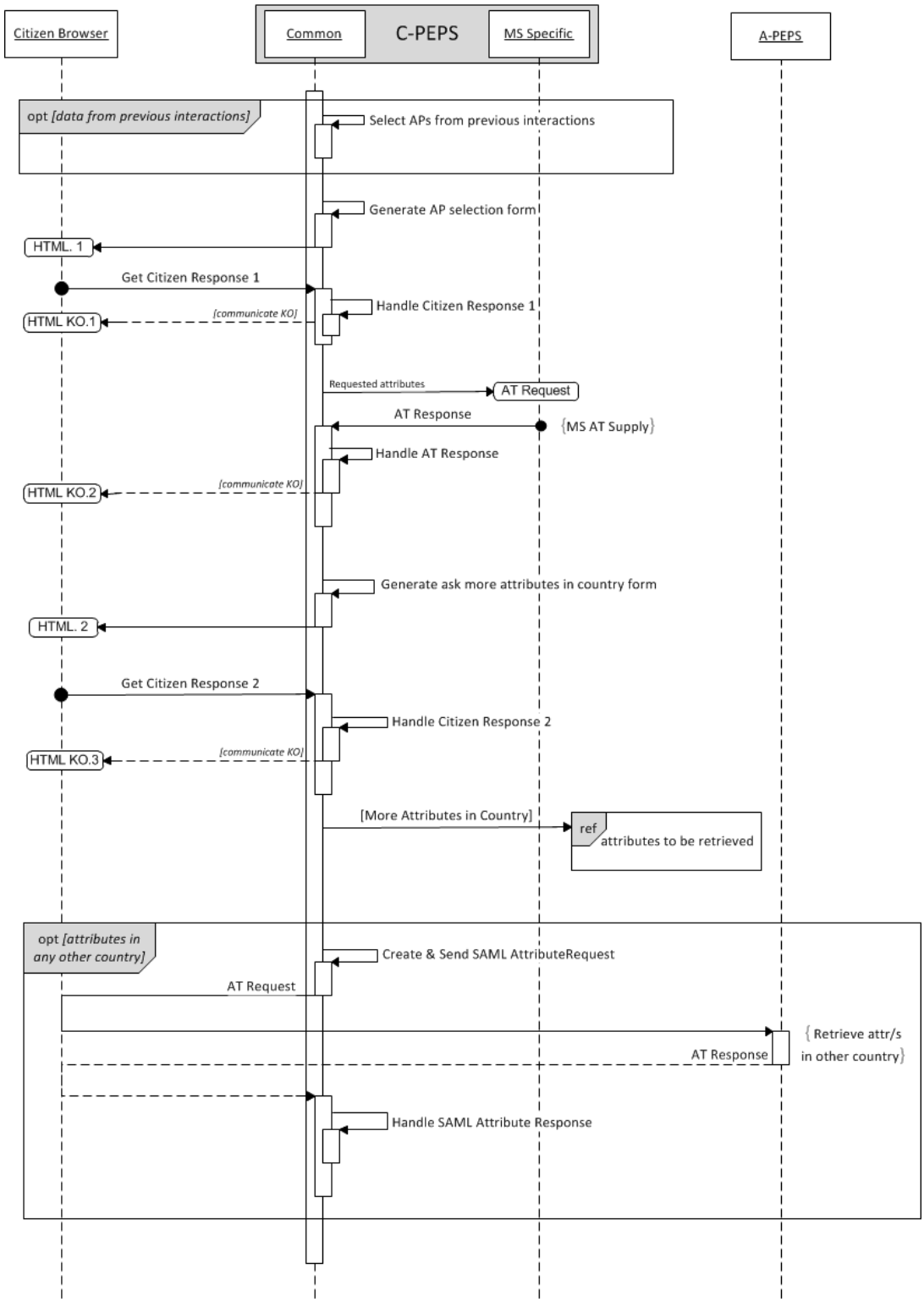
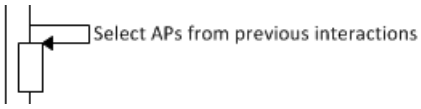
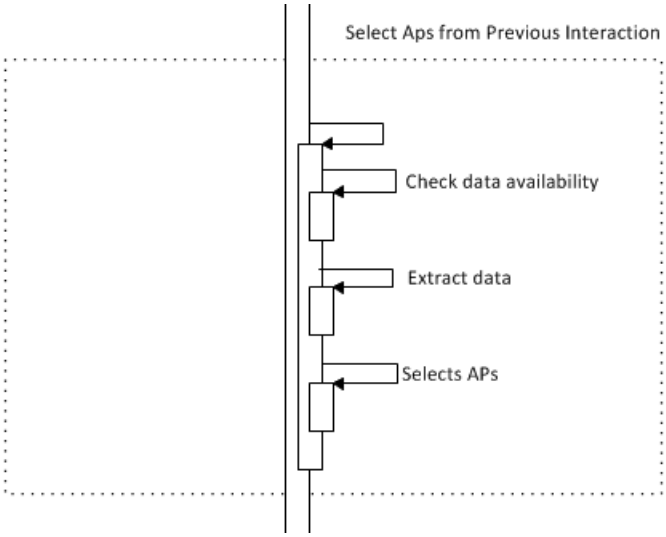
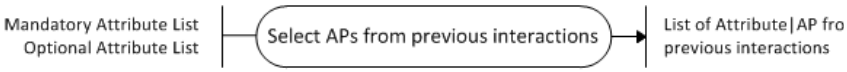
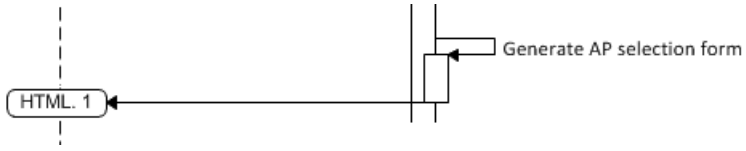
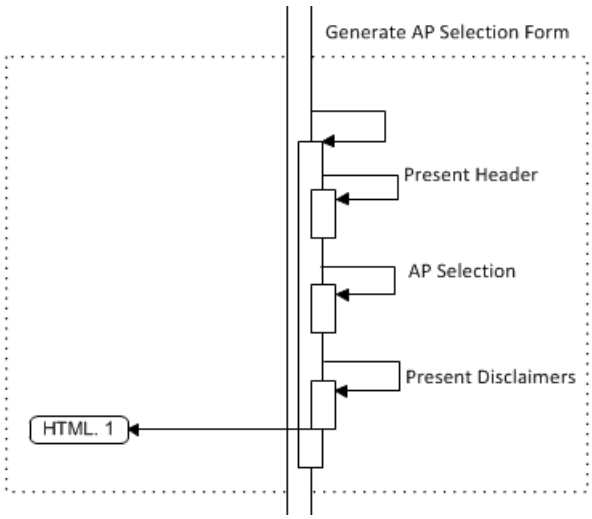
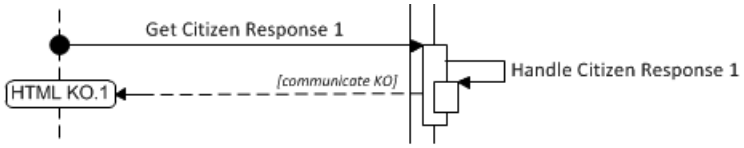
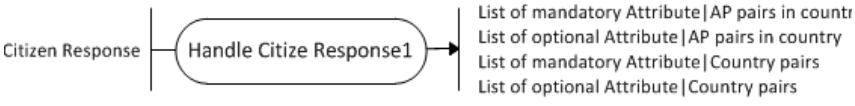


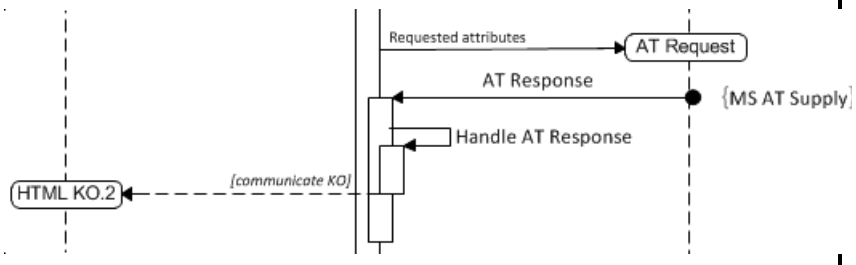
Figure 15: Sequence diagram Authentication on behalf of in C-PEPS, part 2.

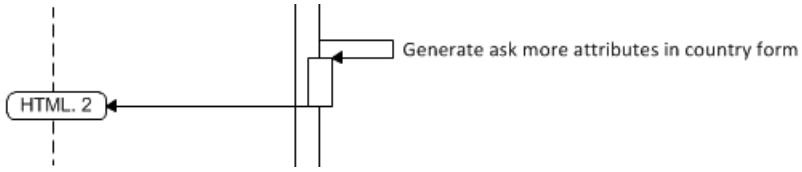
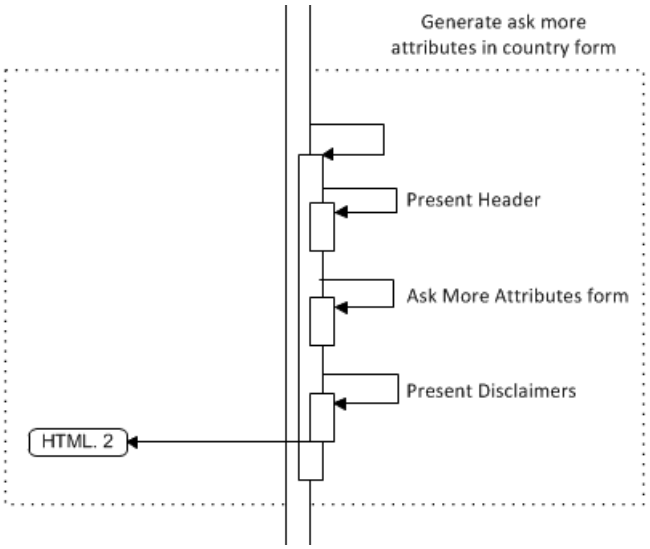
2.4.2.1.4 Description AUB, part 2

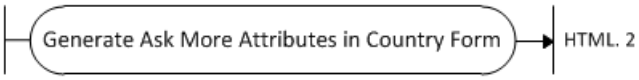
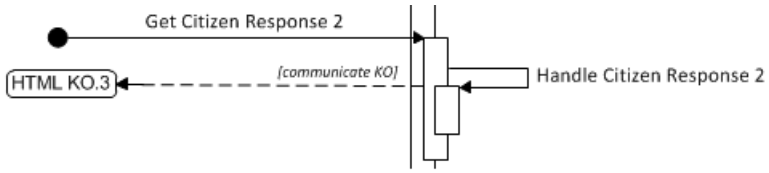
Message sequences (interactions)	*Description*
7 Select APs from previous interactions (optional)	**Description** This task includes activities to check whether data regarding the APs of requested attributes are available from previous interactions of the user with STORK, and extracts them if there are any data available. It is executed only if domain-specific attributes are requested and data from previous interactions with STORK is available.
	Sequence Diagram 
	Detailed Sequence Diagram 
	Data  INPUT • Mandatory domain-specific attribute list • Optional domain-specific attribute list OUTPUT • List of Attribute|AP pairs from previous interactions CONFIG FILES NEEDED • None
	Actions • Check data availability: Checks whether data from previous visits are available. External actors (like user's browser) might

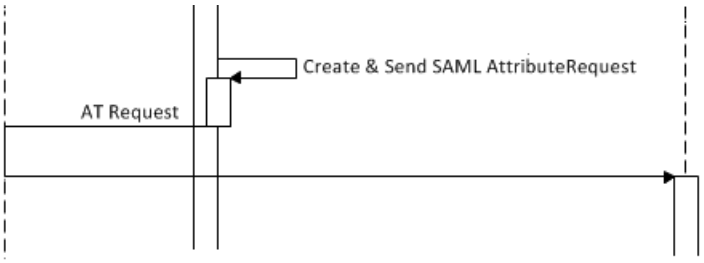
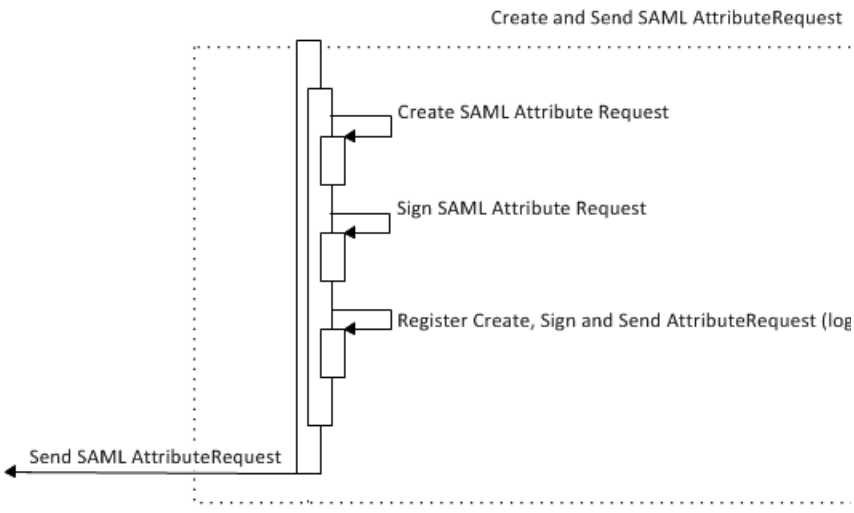
<table>
<tr><th>Message sequences (interactions)</th><th>Description</th></tr>
<tr><td></td><td>be involved in this task, e.g. if the data is stored as cookie in the user's browser it is transparently sent to C-PEPS.
• Extract data: Extracts the data if available
• Select APs: If available in the data, select APs for the requested attributes and generate a list of Attribute|AP pairs from previous interactions</td></tr>
<tr><td>8 Generate AP selection form</td><td>Description
• This task includes some internal activities to generate a page in which the user is prompted with:
a) a list of pre-selected APs for the attributes that have been retrieved in previous interactions with STORK, and should also be allowed to select other APs for these attributes if he wishes so. If the list generated at point a) is empty, the user will select APs as at point b).
b) select APs for the attributes that have not been retrieved in previous interactions with STORK .</td></tr>
<tr><td></td><td>External Actors
• Citizen-Browser: Receives a generated HTML</td></tr>
<tr><td></td><td>Sequence Diagram
 
</td></tr>
<tr><td></td><td>Detailed Sequence Diagram
 
</td></tr>
<tr><td></td><td>Data
 
</td></tr>
</table>

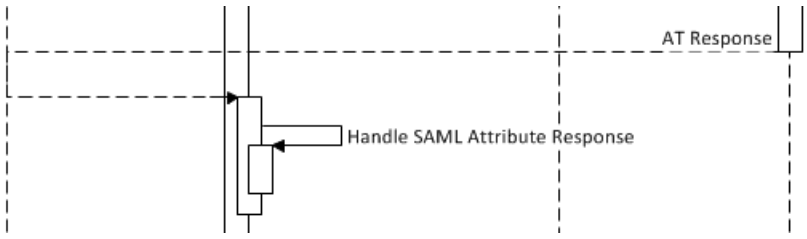
Message sequences (interactions)	***Description***
	INPUT • List of Attribute|AP pairs from previous interactions (if available) • Mandatory domain-specific attribute list in C-PEPS's native language • Optional domain-specific attribute list in C-PEPS's native language OUTPUT • HTML.1 CONFIG FILES NEEDED • List of Attribute Providers in the Country • List of Countries
	Actions • Present header (SP name) • AP Selection: Present attributes list (requested attributes: mandatory and optional) with a listbox next to each attribute for the selection of AP. The listbox lists the Attribute Providers in the country, an option "AP in another country" and an option "Do not retrieve". An option (either AP or "Other Country" is preselected if information about the attribute is available from previous interactions. An AP should be provided for each mandatory attribute. • Present disclaimers
9 Handle Citizen Response 1	**Description** Receives and handles citizen's reply.
	Sequence Diagram 
	External Actors • Citizen
	Data  INPUT • Citizen response

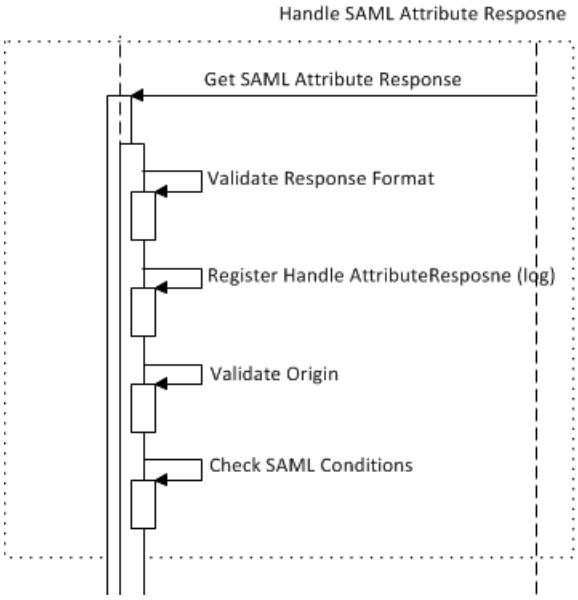
Message sequences (interactions)	***Description***
	OUTPUT • List of mandatory Attribute|AP pairs in country • List of optional Attribute|AP pairs in country • List of mandatory Attribute|Country pairs • List of optional Attribute|Country pairs CONFIG FILES NEEDED • List of Attribute Providers in the Country • List of Countries
	Actions • Register citizen reply (log citizen's selection) • Check citizen reply format (If the response is malformed → Handle Error CPAUB0201) • Check selection (If an AP is not selected for a mandatory attribute → Handle Error CPAU0202)
	Error Communications • Send HTML KO.1 to the Citizen-Browser. (If a Handle Error succeed).
10 MS Attribute Supply	**Description** Retrieve attribute values from attribute providers.
	Sequence Diagram 
	Data  INPUT • List of mandatory Attribute|AP pairs in country • List of optional Attribute|AP pairs in country • Requested QAA Level OUTPUT • Value attribute list

Message sequences (interactions)	***Description***
	CONFIG FILES NEEDED • None
	Actions Is MS Specific If re-authentication takes place at an AP, the AP should also include in the AT Response the name, surname, and date of birth of the user. If values for a mandatory attribute not found → Handle Error CPAUB0301
1 1 Generate ask more attributes in country form	**Description** The objective of this activity is to generate a form asking the user if any more additional attributes will be requested in the country
	External Actors • Citizen-Browser: Receives a generated HTML
	Sequence diagram 
	Detailed Sequence Diagram 

Message sequences (interactions)	***Description***
	Data  INPUT • None OUTPUT • HTML. 2 CONFIG FILES NEEDED • None
	Actions • Present header (SP name) • Ask More Attributes: Present a selection for asking more attributes • Present disclaimers
1 Handle Citizen 2 Response 2	**Description** Receives and handles citizen's reply.
	External Actors • Citizen
	Sequence Diagram 
	Data  INPUT • Citizen Response OUTPUT • More attributes in the country (boolean) CONFIG FILES NEEDED • None

Message sequences (interactions)	*Description*
	Actions • Register citizen reply (log citizen's selection) • Check citizen reply format (If the response is malformed → Handle Error CPAUB0501)
	Error Communications • Send HTML KO.3 to the Citizen-Browser. (If a Handle Error succeed).
13 Create & Send SAML Attribute Request (optional)	**Description** Creates and sends an attribute request to a colleague PEPS/V-IDP. This task is only executed if there are domain-specific attributes in another country.
	External Actors • Citizen-Browser: The C-PEPS sends the SAML AttributeRequest to a colleague C-PEPS/V-IDP through the Citizen-Browser.
	Sequence Diagram 
	Detailed Sequence Diagram 

Message sequences (interactions)	***Description***
	Data  INPUT • Mandatory Attribute|Country list • Optional Attribute|Country list OUTPUT • SAML AttributeRequest CONFIG FILES NEEDED • Log Configuration
	Actions • Create SAML AttributeRequest • Sign SAML AttributeRequest • Register Create, Sign and Send Attribute Request
14 Handle SAML AttributeResponse (optional)	**Description** C-PEPS receives SAML AttributeResponse issued by Colleague PEPS/V-IDP. This task is executed only if there are domain-specific attributes in another country.
	External Actors • Citizen-Browser: The C-PEPS receives the SAML AttributeResponse from a colleague C-PEPS/V-IDP through the Citizen-Browser.
	Sequence Diagram 

Message sequences (interactions)	***Description***
	Detailed Sequence Diagram 
	Data  INPUT • SAML AttributeResponse OUTPUT • Value attribute list CONFIG FILES NEEDED • Log Configuration
	Actions • Receive SAML AttributeResponse • Register Handle SAML AttributeResponse (log) • Validate Response Fomat (SAML) (If the format is not correct → Handle Error CPAUB0701) • Validate Origin: Validate Colleague PEPS/ V-IDP Signature. (If the origin is not correct → Handle Error CPAUB0702) • Check SAML Conditions (notBefore, notAfter, etc.) (If conditions are not fulfilled → Handle Error CPAUB0703)
15 Normalise data	**Description** Normalises all received data (from MS Authentication, MS AT Supply and) mapping the MS names/values to STORK nomenclature.

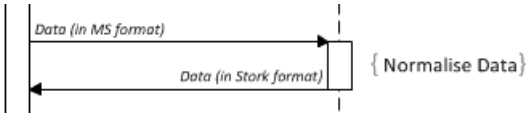
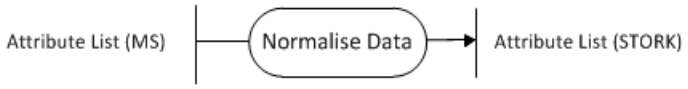
Message sequences (interactions)	*Description*
	Sequence Diagram 
	Data  INPUT • Attribute list (names and data values in MS nomenclature) OUTPUT • Attribute list (names and data values in STORK nomenclature) CONFIG FILES NEEDED • Attribute mapping
	Actions Some parts may be MS Specific

Table 9 –Description sequence Authentication on Behalf of, part 2, in C-PEPS

2.4.2.1.5 Sequence diagram AUB, part 3

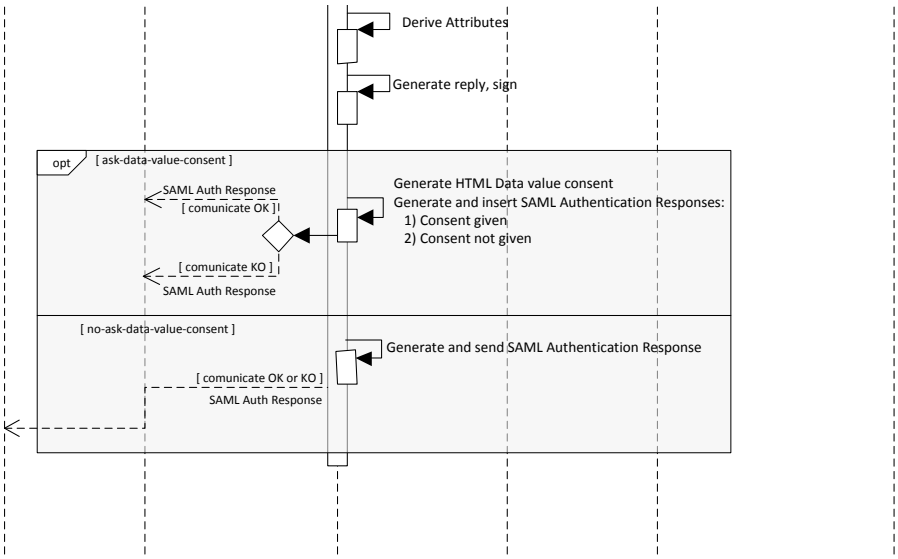
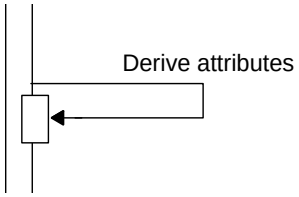
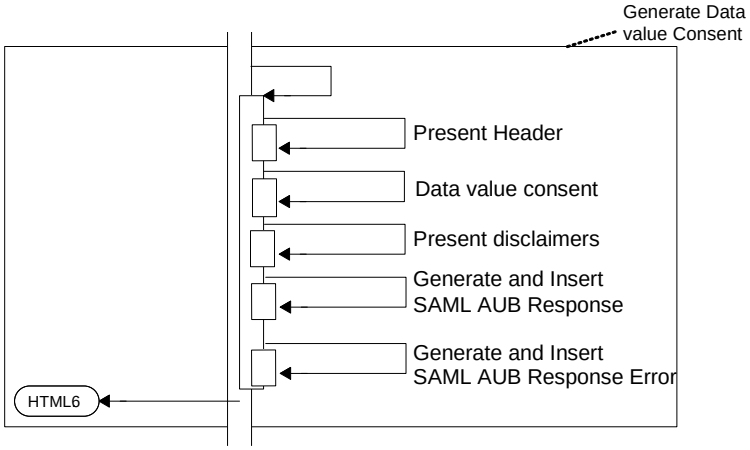
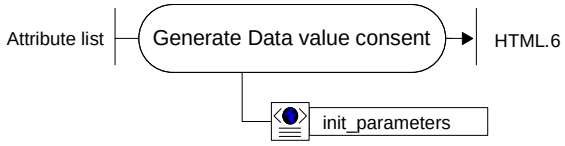
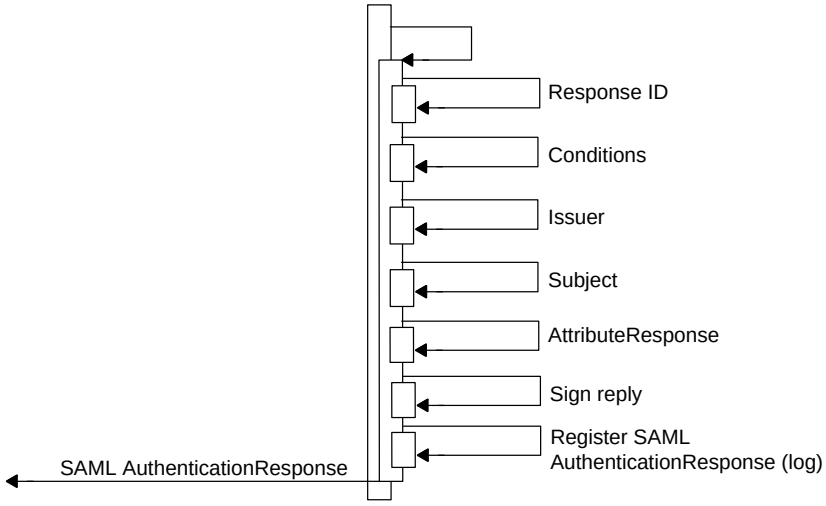
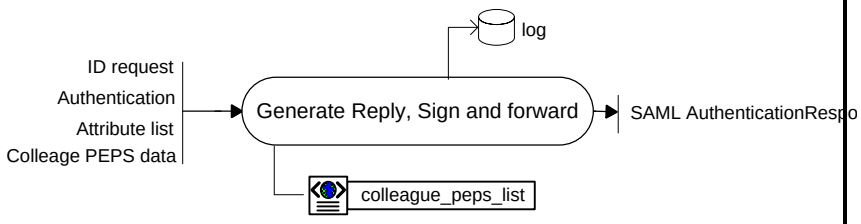


Figure 16: Sequence diagram Authentication on behalf of in C-PEPS, part 3.

2.4.2.1.6 Description AUB, part 3

Message sequences (interactions)		*Description*
20	Derive attributes	**Description** Derived attribute data is constructed
		Sequence Diagram 
		External Actors • None
		Data INPUT • Attributes and values: Date of Birth, Age threshold (s), eIdentifier OUTPUT • Derived data: Age, IsAgeOver <threshold>, eIdentifier CONFIG FILES NEEDED • none
		Actions • Derive age • Compare with threshold (age) • Creates the eIdentifier
21	Generate HTML Data value consent (optional)	**Description** • This task includes some internal activities to generate the page in which the user is requested to give his consent to the transfer of his data types and values in the C-PEPS native language except for derive data. This step is optional, only it is executed if the ask-data-value-consent is set to YES in the *init_parameters* config file. If the Citizen gives data-value-consent, a SAML Authentication Response is sent to the SP, if the data-value-consent is not given another SAML Authentication Response (Error) is sent.
		Sequence Diagram 

Message sequences (interactions)		*Description*
		Detailed Sequence Diagram 
		Data  INPUT • Attributes list in MS nomenclature OUTPUT • HTML.6 CONFIG FILES NEEDED • init_parameters
		Actions • Present destination (SP name) • Present attribute list (requested attributes) with values • Present disclaimers
22	Generate and Send SAML Authenticatio n Response	**Description** Generates the SAML token, and signs it. It is optional, only it is executed if the ask-data-value-consent is set to NO in the *init_parameters* config file.
		Sequence Diagram 
		Detailed Sequence Diagram

Message sequences (interactions)	*Description*
	
	External Actors • S-PEPS, S-V-IDP
	Data  INPUT • ID request • Colleague PEPS data • Attribute list • Attribute list (consented) OUTPUT • SAML AuthenticationResponse CONFIG FILES NEEDED • colleague_peps_list
	Actions • Generate SAML Response ID • Generate SAML Conditions (NotBefore, etc) • Generate Issuer • Generate Subject • Generate AuthenticationResponse (attribute assertions can be included)

Message sequences (interactions)		*Description*
		• Generate PEPS Signature • Send SAML Authentication Response

Table 10 –Description sequence Authentication on behalf of, part 3, in C-PEPS

2.4.2.2 Domain-specific attributes

The Domain-specific attributes process, similar to the *Authentication on Behalf of*, is initiated when a Service Provider needs to know the user's identity, and sends a *Domain-specific attributes* request to the S-PEPS through the citizen's Web Browser. In the same way, the S-PEPS will redirect the request (as a SAML AuthnRequest) to the C-PEPS through the citizen's Web Browser as well.

The authentication process begins when the S-PEPS sends a SAML AuthRequest to the C-PEPS (Sequence diagram BA).

2.4.2.2.1 Sequence diagram BA, part 1

Same as 2.4.2.1.1, however the MS specific authentication function is different from the one in that section. In this process only the user is authenticated, no data are retrieved from any represented person (as there is not any), nor of any mandate.

2.4.2.2.2 Description BA, part 1

Same as 2.4.2.1.2.

2.4.2.2.3 Sequence diagram BA, part 2

Same as 2.4.2.1.3.

2.4.2.2.4 Description BA, part 2

Same as 2.4.2.1.4.

2.4.2.2.5 Sequence diagram BA, part 3

Same as 2.4.2.1.5.

2.4.2.2.6 Description BA, part 3

Same as 2.4.2.1.6.

2.4.2.3 Powers Validation

The Powers Validation process, similar to the *Authentication on Behalf of*, is initiated when a Service Provider needs to know the user's identity, and sends a *Powers Validation* request to the S-PEPS through the citizen's Web Browser. In the same way, the S-PEPS will redirect the request (as a SAML AuthnRequest) to the C-PEPS through the citizen's Web Browser as well.

The authentication process begins when the S-PEPS sends a SAML AuthRequest to the C-PEPS (Sequence diagram PV).

2.4.2.3.1 Sequence diagram BA, part 1

Same as 2.4.2.1.1, however the MS specific authentication function is different from the one in that section. In this process only the user is authenticated, no data are retrieved from any represented person (as there is not any), nor of any mandate.

2.4.2.3.2 Description BA, part 1

Same as 2.4.2.1.2.

2.4.2.3.3 Sequence diagram BA, part 2

Same as 2.4.2.1.3.

2.4.2.3.4 Description BA, part 2

Same as 2.4.2.1.4.

2.4.2.3.5 Sequence diagram BA, part 3

Same as 2.4.2.1.5.

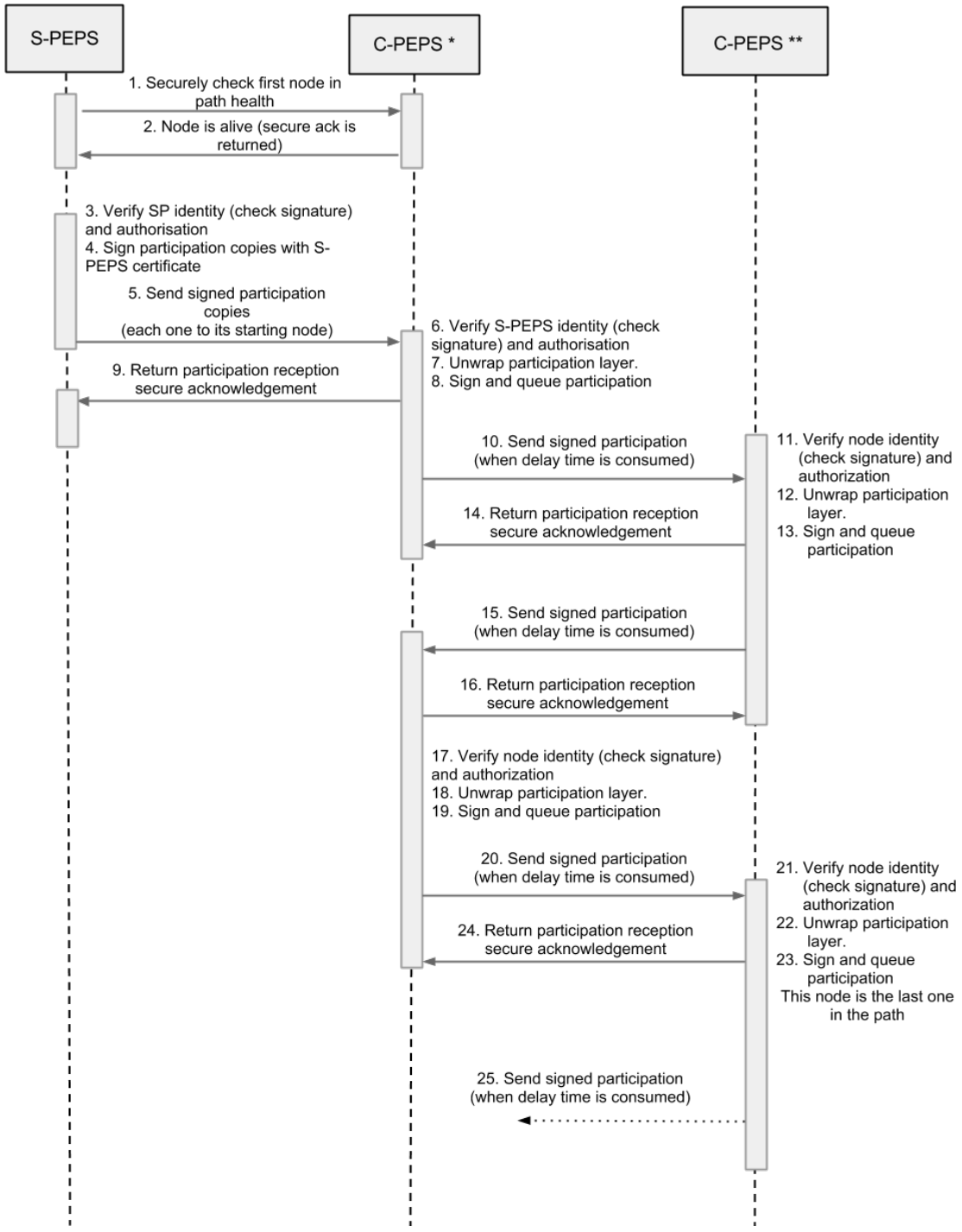
2.4.2.3.6 Description BA, part 3

Same as 2.4.2.1.6.

2.4.2.4 Anonymity – First node

The Anonymity-First node is initiated for each copy of the eSurvey received by the S-PEPS when it sends this copy to the first node mentioned in the copy.

2.4.2.4.1 Sequence diagram Anonymity First Node

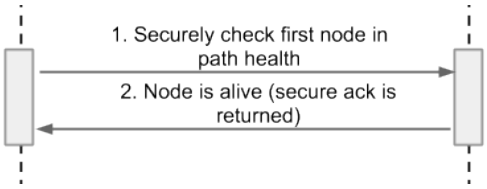
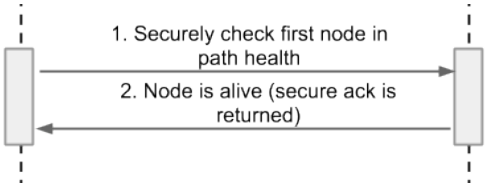


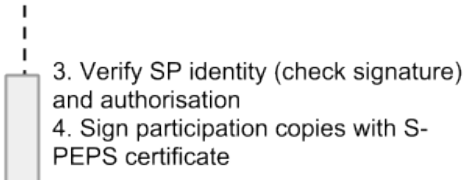
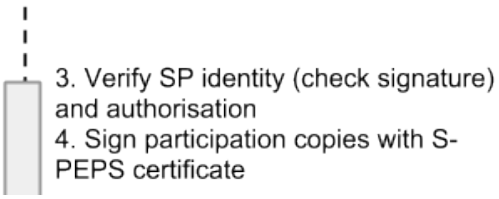
* This node entity will represent the first one on each path.
** This node entity will represent any other node on each path.

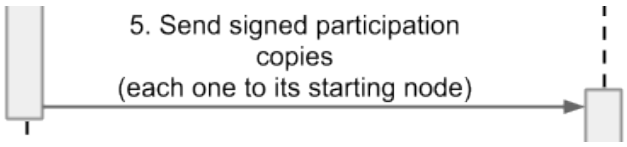
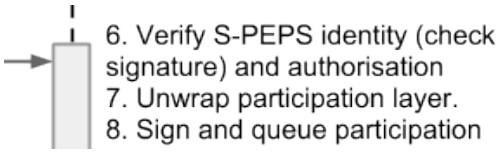
Figure 17: Sequence diagram Anonymity First Node in C-PEPS

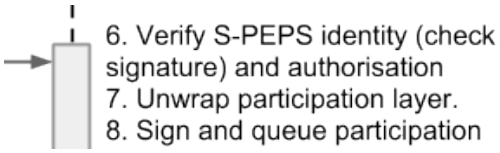
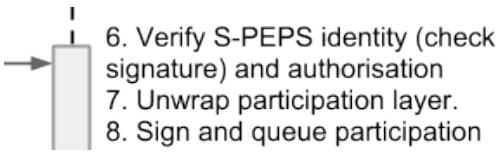
2.4.2.4.2 Description

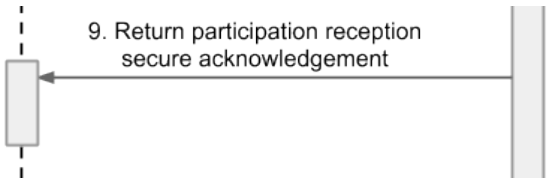
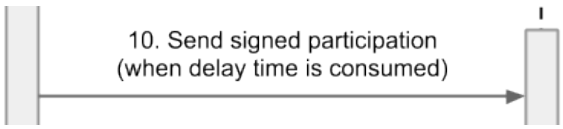
Message sequences (interactions)		***Description***
1	Securely check first node in	For each node that will serve as first node in a given path, the client checks if it is up and can use it. This is done by sending a special

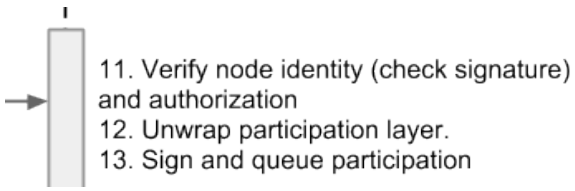
Message sequences (interactions)		***Description***
	path health	packet with a ciphered challenge, this challenge can only be deciphered by the target node. If the node is up and the software is running correctly, it will be able to return the challenge in plain text. This packet is sent using an *HTTPS POST* method.
		Sequence Diagram 
		External Actors • Citizen • SP • S-PEPS • Each first node in each path
		Data INPUT • Dummy packet with a challenge OUTPUT •
		Error Communications • Invalid secure ping
		Actions • Citizen's client calculates a packet with a challenge for each of the first node on each path. • Next, it delivers each packet to the SP. • The SP signs the packet and bounces the packet to the S-PEPS. • The S-PEPS signs the packet and bounces to the destination node.
2	Node is alive (secure ACK is returned)	If the node has been able to send back the plain challenge, the client will count it; otherwise, the client will choose a new first node for this path in a random way.
		Sequence Diagram 
		External Actors • Citizen • SP • S-PEPS • Each first node in each path

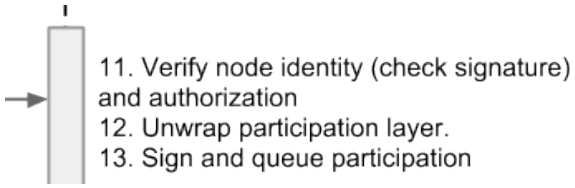
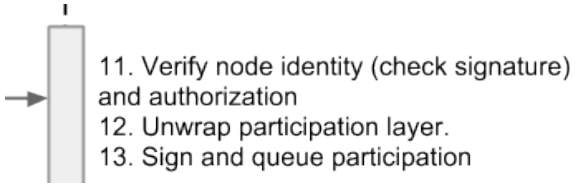
Message sequences (interactions)		***Description***
		Data INPUT • OUTPUT • Secure ACK deciphered
		Error Communications •
		Actions • The node deciphers the packet, signs the challenge and sends the result back to the S-PEPS. • The result is bounced back to the SP, and then backs to the citizen.
3	Verify SP identity (check signature) and authorisation	The *S-PEPS* verifies the *SP* signature to authenticate the source of the ciphered packets. It will also act as a proxy to the network, thus the previous steps (16 and 17) are repeated for the *S-PEPS*.
		Sequence Diagram 
		External Actors • S-PEPS
		Data INPUT • Signed and layer-ciphered packets OUTPUT •
		Error Communications •
		Actions • S-PEPS verifies the signature from the SP.
4	Sign participation copies with S-PEPS certificate	*S-PEPS* signs the participations using its certificate and the classic *SAML STORK* profile.
		Sequence Diagram 

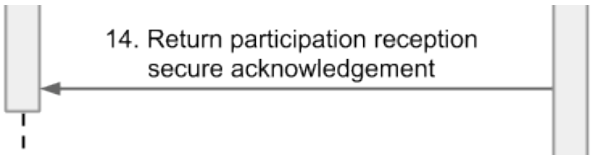
Message sequences (interactions)		***Description***
		External Actors • S-PEPS
		Data INPUT • Layer-ciphered participation copies. OUTPUT •
		Error Communications •
		Actions • S-PEPS receives the layer-ciphered participation copies and signs them.
5	Send signed participation copies (each one to its starting node)	The *S-PEPS* bounces the signed participations to each first node on each path using an *HTTPS POST* method.
		Sequence Diagram 
		External Actors • S-PEPS • Each starting node on calculated paths
		Data INPUT • OUTPUT • Signed participations
		Error Communications •
		Actions • The S-PEPS bounces the signed participations to each first node in the path.
6	Verify S-PEPS identity (check signature) and authorisation	Each first node on each path verifies the *S-PEPS* signature and also that the certificate used is trusted.
		Sequence Diagram 

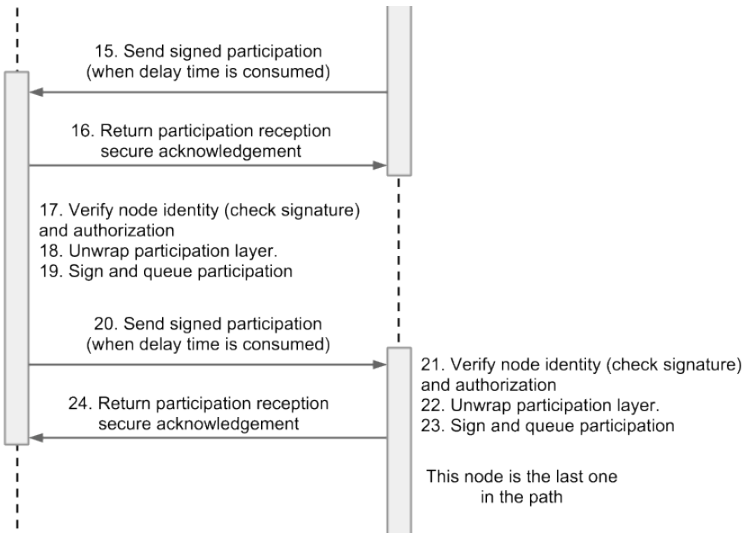
Message sequences (interactions)		***Description***
		External Actors • Each starting node on calculated paths
		Data INPUT • Layer-ciphered and signed packet OUTPUT •
		Error Communications •
		Actions • The node verifies the S-PEPS identity and authorisation (by checking the signature.
7	Unwrap participation layer	Each first node on each path unwraps the layer of the packet addressed to it. To achieve this, it will decipher an *RC4* key using its *RSA* private key and will use this *RC4* key to decipher the upper most layer of the packet. In this layer, the challenge must send back and has to wait for the next node, the time to wait and the *URL* of the next node are present.
		Sequence Diagram 
		External Actors • Each starting node on calculated paths
		Data INPUT • Layer-ciphered and signed packet OUTPUT •
		Error Communications •
		Actions • Each starting node on calculated paths unwraps the upper-most participation layer to access the information addressed to it.
8	Sign and queue participation	The node signs the participation and queue for the indicated time.
		Sequence Diagram 

Message sequences (interactions)		***Description***
		External Actors • Each starting node on calculated paths
		Data INPUT • Layer-ciphered packet OUTPUT • Layer-ciphered and signed packet
		Error Communications •
		Actions • Each starting node on calculated paths sign and queue the packet.
9	Return participation reception secure acknowledgement	Simultaneously the node returns the Participation Reception Secure Acknowledgement (*PRSA*). This will be returned as a response to the *HTTPS POST* method used by the *S-PEPS*.
		Sequence Diagram 
		External Actors • S-PEPS • Each starting node on calculated paths
		Data INPUT • Layer-ciphered packet OUTPUT • Secure ACK
		Error Communications •
		Actions • The node returns the participation reception secure acknowledgement.
10	Send signed participation (when delay time has passed)	When the delay time has passed, the node sends the signed participation to the next node in the path.
		Sequence Diagram 

Message sequences (interactions)		***Description***
		External Actors • Each starting node on calculated paths • Second node in the path
		Data INPUT • Layer-ciphered participation OUTPUT • Layer-ciphered signed participation
		Error Communications •
		Actions • Each starting node on calculated paths send the signed participation when the indicated delay time has passed.
11	Verify node identity (check signature) and authorisation	The node verifies the prior node signature and also that the certificate used is trusted.
		Sequence Diagram 
		External Actors •
		Data INPUT • Layer-ciphered and signed participation OUTPUT •
		Error Communications •
		Actions • The node verifies the signature produced by the previous node and checks its trust.
12	Unwrap participation layer	The node unwraps the upper most layer of the participation by deciphering a *RC4* key using its *RSA* private key. This *RC4* key will be used to decipher the upper most layer of the participation. In this layer, the challenge must be sent back, the challenge has to wait for the next node, the time to wait and the *URL* of the next node are present.

Message sequences (interactions)		***Description***
		Sequence Diagram 
		External Actors • Node
		Data INPUT • Layer-ciphered participation OUTPUT •
		Error Communications •
		Actions • The node unwraps the upper-most layer of the participation getting the information that is addressed to it.
13	Sign and queue participation	The node signs the participation using the classic *SAML STORK* profile and, when the time sends to the next node in the path using an *HTTPS POST* method.
		Sequence Diagram 
		External Actors • Node
		Data INPUT • Layer-ciphered participation OUTPUT • Layer-ciphered and signed participation
		Error Communications •
		Actions • The node performs a signature over the package and stores it for the time that is indicated in the information it has obtained.

<table>
<tr><th colspan="2">Message sequences (interactions)</th><th>Description</th></tr>
<tr><td>14</td><td>Return participation reception secure acknowledgement</td><td>The node sends back the PRSA to the sender node as a response to the HTTPS POST method initiated at step 27. The sender node checks that the challenge matches the expected one.</td></tr>
<tr><td></td><td></td><td>Sequence Diagram
 
</td></tr>
<tr><td></td><td></td><td>External Actors
• Node</td></tr>
<tr><td></td><td></td><td>Data
INPUT
•
OUTPUT
• The secure ACK obtained in the deciphering step</td></tr>
<tr><td></td><td></td><td>Error Communications
•</td></tr>
<tr><td></td><td></td><td>Actions
• The node sends back the secure ACK obtained when it has deciphered the participations.</td></tr>
<tr><td>15</td><td>Send signed participation (when delay time is consumed)</td><td rowspan="5">From that point, the participation will follow the client’s calculated path within the network. Steps 15 to 24 are repeated for each node in the path.</td></tr>
<tr><td>16</td><td>Return participation reception secure acknowledgement</td></tr>
<tr><td>17</td><td>Verify node identity (check signature) and authorisation</td></tr>
<tr><td>18</td><td>Unwrap participation layer</td></tr>
<tr><td>19</td><td>Sign and queue participation</td></tr>
</table>

Message sequences (interactions)		***Description***
20	Send signed participation (when delay time is consumed)	
21	Verify node identity (check signature) and authorisation	
22	Unwrap participation layer	
23	Sign and queue participation	
24	Return participation reception secure acknowledgement	
		Sequence Diagram 
		External Actors • Nodes
		Data INPUT • Layer-ciphered (signed) participation OUTPUT • Layer-ciphered (signed) participation
		Error Communications •

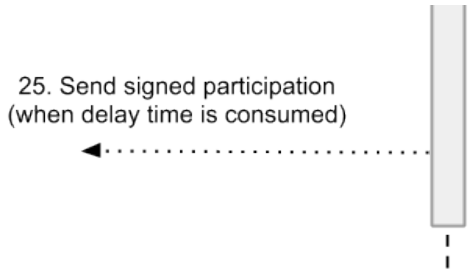
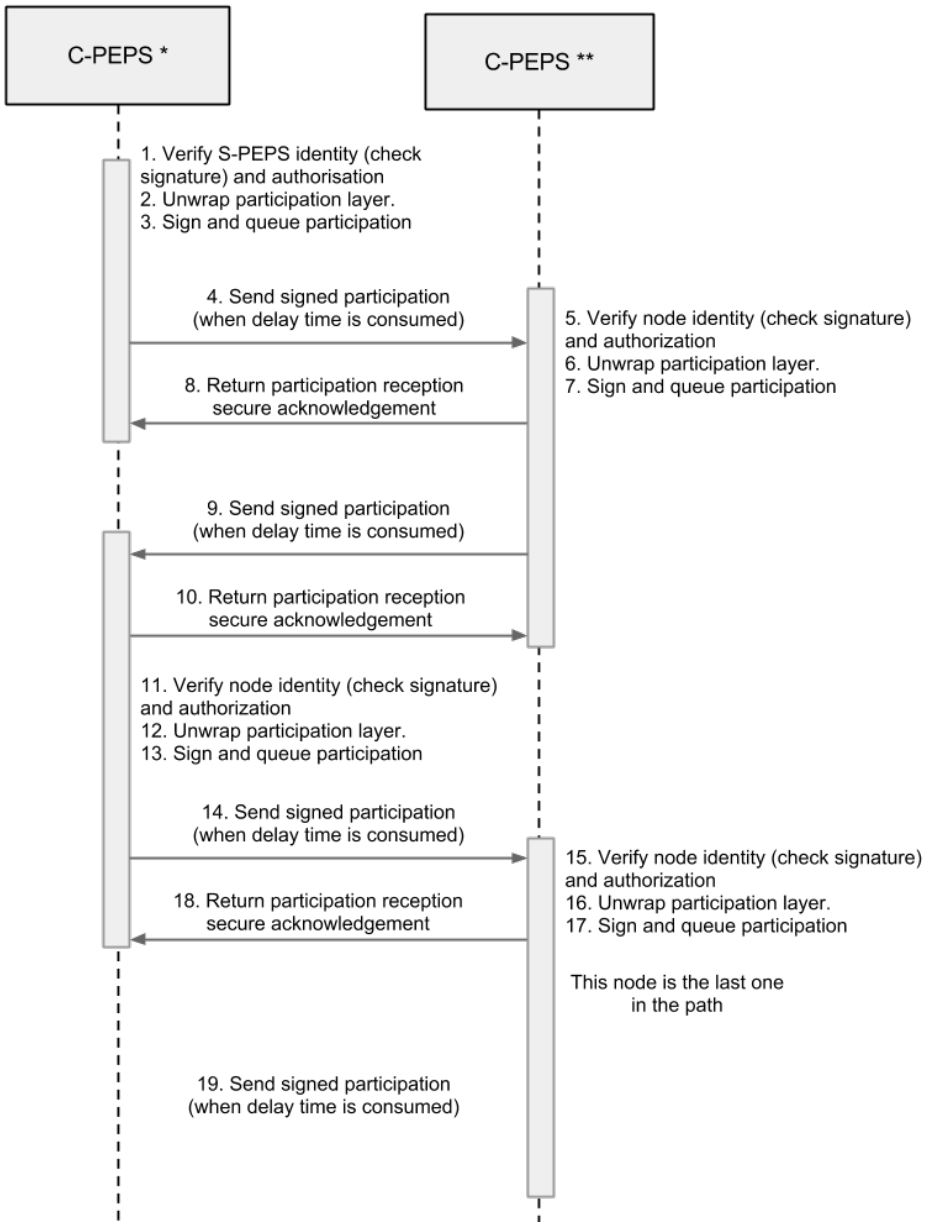
Message sequences (interactions)		*Description*
		Actions • For each one of the remaining nodes, steps 27 to 32 are repeated in a loop until the last node in the path is reached.
25	Send signed participation (when delay time is consumed)	The last node in the path sends the signed participation to the SP as a final step.
		Sequence Diagram 
		External Actors • Last node in each path • SP
		Data INPUT • Signed participation OUTPUT •
		Error Communications •
		Actions • Last node in the path sends the signed participation to the SP.

Table 11 –Description sequence Anonymity First Node in C-PEPS

2.4.2.5 Anonymity – Other node

The Other node function receives the eSurvey and simply returns it after the requested delay has passed.

2.4.2.5.1 Sequence diagram

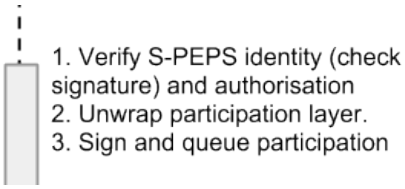
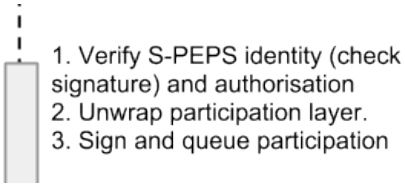


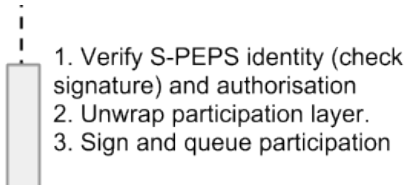
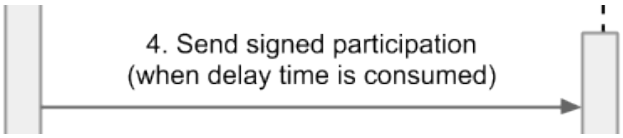
* This node entity will represent the first one on each path
** This node entity will represent any other node on each path.

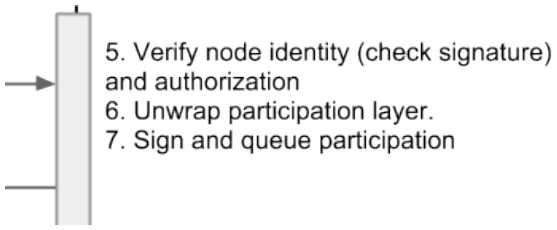
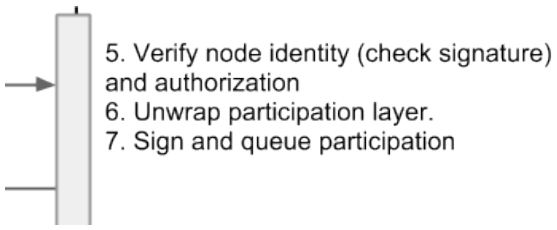
Figure 18: Sequence diagram Anonymity Other Node in C-PEPS

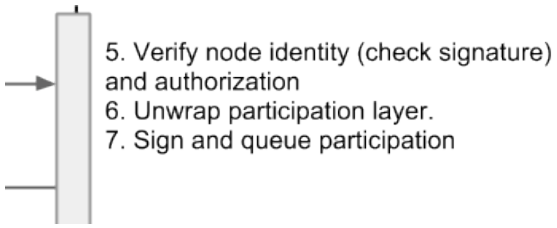
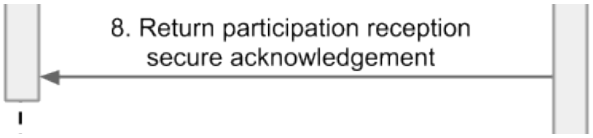
2.4.2.5.2 Description

Message sequences (interactions)		***Description***
1	Verify S-PEPS identity (check signature)	Each other node on each path verifies the *S-PEPS* signature and also that the certificate used is trusted.
		Sequence Diagram

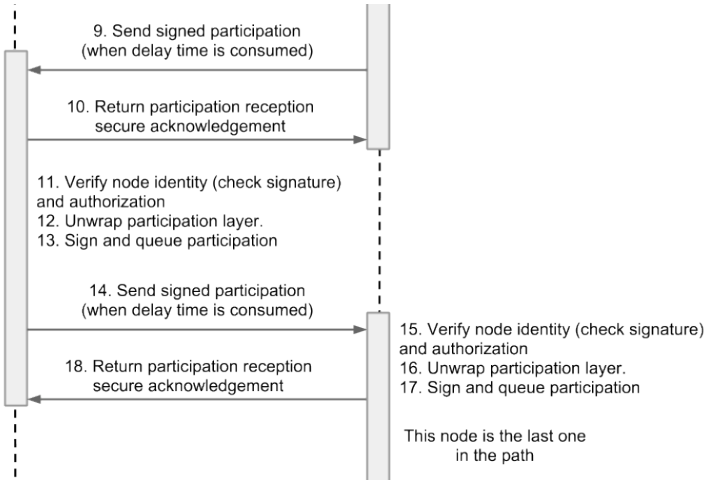
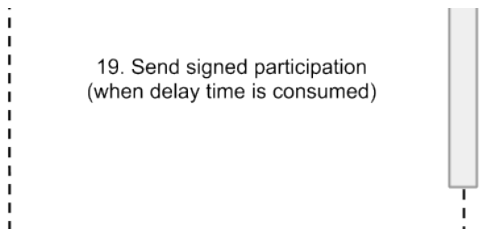
Message sequences (interactions)		***Description***
	and authorisation	
		External Actors • Each starting node on calculated paths
		Data INPUT • Layer-ciphered and signed packet OUTPUT •
		Error Communications •
		Actions • The node verifies the S-PEPS identity and authorisation (by checking the signature.
2	Unwrap participation layer	Each other node on each path unwraps the layer of the packet sent to it. To achieve this it will decipher an *RC4* key using its *RSA* private key and will use this *RC4* key to decipher the upper most layer of the packet. In this layer, the challenge must be sent back, and has to wait for the next node, the time to wait and the *URL* of the next node are present.
		Sequence Diagram 
		External Actors • Each starting node on calculated paths
		Data INPUT • Layer-ciphered and signed packet OUTPUT •
		Error Communications •
		Actions • Each starting node on calculated paths unwraps the upper-most participation layer to access the information addressed to it.

Message sequences (interactions)		***Description***
3	Sign and queue participation	The node signs the participation and queue for the indicated time.
		Sequence Diagram 
		External Actors • Each starting node on calculated paths
		Data INPUT • Layer-ciphered packet OUTPUT • Layer-ciphered and signed packet
		Error Communications •
		Actions • Each starting node on calculated paths sign and queue the packet.
4	Send signed participation (when delay time has passed)	When the delay time has passed the node sends the signed participation to the next node in the path.
		Sequence Diagram 
		External Actors • Each starting node on calculated paths • Second node in the path
		Data INPUT • Layer-ciphered participation OUTPUT • Layer-ciphered signed participation
		Error Communications •
		Actions • Each starting node on calculated paths send the signed participation when the indicated delay time is consumed.
5	Verify node identity	The node verifies the prior node signature and also that the certificate used is trusted.

Message sequences (interactions)		***Description***
	(check signature) and authorisation	**Sequence Diagram** 
		External Actors •
		Data INPUT • Layer-ciphered and signed participation OUTPUT •
		Error Communications •
		Actions • The node verifies the signature produced by the previous node and checks its trust.
6	Unwrap participation layer	The node unwraps the upper layer of the participation by deciphering a *RC4* key using its *RSA* private key. This *RC4* key will be used to decipher the upper layer of the participation. In this layer, the challenge must be sent back, and has to wait for the next node, the time to wait and the *URL* of the next node are present.
		Sequence Diagram 
		External Actors • Node
		Data INPUT • Layer-ciphered participation OUTPUT •
		Error Communications •

Message sequences (interactions)		***Description***
		Actions • The node unwraps the upper-most layer of the participation getting the information that is addressed to it.
7	Sign and queue participation	The node signs the participation using the classic *SAML STORK* profile and, when the time sends to the next node in the path using an *HTTPS POST* method.
		Sequence Diagram 
		External Actors • Node
		Data INPUT • Layer-ciphered participation OUTPUT • Layer-ciphered and signed participation
		Error Communications •
		Actions • The node performs a signature over the package and stores it for the time that is indicated in the information it has obtained.
8	Return participation reception secure acknowledgement	The node sends back the *PRSA* to the sender node as a response to the *HTTPS POST* method initiated at step 27. The sender node checks that the challenge matches the expected one.
		Sequence Diagram 
		External Actors • Node
		Data INPUT • OUTPUT • The secure ACK obtained in the deciphering step

<table>
<tr><th colspan="2">Message sequences (interactions)</th><th>Description</th></tr>
<tr><td colspan="2" rowspan="2"></td><td>Error Communications</td></tr>
<tr><td>Actions
The node sends back the secure ACK obtained when it has deciphered the participations.</td></tr>
<tr><td>9</td><td>Send signed participation (when delay time is consumed)</td><td rowspan="9">From that point, the participation will follow the client’s calculated path within the network. Steps 9 to 18 are repeated by each node in the path interacting with the next node in the same path. The description of these actions has the same step description as the previous actions given that they are equivalent.</td></tr>
<tr><td>10</td><td>Return participation reception secure acknowledgement</td></tr>
<tr><td>11</td><td>Verify node identity (check signature) and authorisation</td></tr>
<tr><td>12</td><td>Unwrap participation layer</td></tr>
<tr><td>13</td><td>Sign and queue participation</td></tr>
<tr><td>14</td><td>Send signed participation (when delay time is consumed)</td></tr>
<tr><td>15</td><td>Verify node identity (check signature) and authorisation</td></tr>
<tr><td>16</td><td>Unwrap participation layer</td></tr>
<tr><td>17</td><td>Sign and queue participation</td></tr>
</table>

Message sequences (interactions)		***Description***
18	Return participation reception secure acknowledgement	
		Sequence Diagram 
		External Actors • Nodes
		Data INPUT • Layer-ciphered (signed) participation OUTPUT • Layer-ciphered (signed) participation
		Error Communications •
		Actions • For each one of the remaining nodes, steps 27 to 32 are repeated in a loop until the last node in the path is reached.
19	Send signed participation (when delay time is consumed)	The last node in the path sends the signed participation to the SP as a final step.
		Sequence Diagram 

Message sequences (interactions)	*Description*
	External Actors • Last node in each path • SP
	Data INPUT • Signed participation OUTPUT •
	Error Communications •
	Actions • Last node in the path sends the signed participation to the SP.

Table 12 –Description sequence Anonymity First Node in C-PEPS

2.4.3 A-PEPS

2.4.3.1 Authentication on behalf of, Powers (for digital signature) and Domain-specific attributes

The *authentication on behalf of* process is carried out using the citizen's Web Browser as a gateway for every message that needs to be exchanged between two STORK entities. Thereby, the request a SP has to send to the S-PEPS will be performed through the citizen's Web Browser. In the same way, the S-PEPS will redirect (if needed) the authentication request (as a SAML AuthnRequest) to the C-PEPS through the citizen's Web Browser as well, which on its turn may forward the request to the A-PEPS.

The *Powers (for Digital Signature)* and *Domain-specific attributes* are – within the A-PEPS, exactly the same as *Authentication on behalf of*.

The authentication process begins when the C-PEPS sends a SAML AuthRequest to the A-PEPS (Sequence diagram AUB).

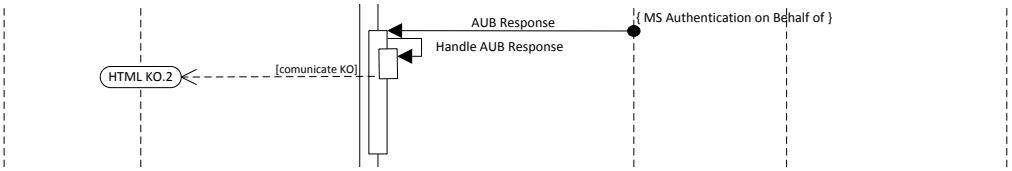
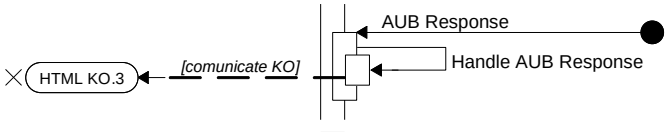


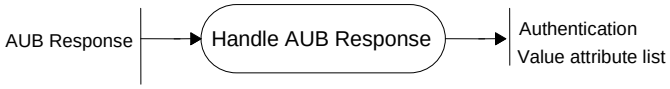
Figure 19: Sequence diagram Authentication on Behalf of, Part 1, in A-PEPS

2.4.3.1.1 Description AUB, part 1

Message sequences (interactions)		
1	Handle AUB Response	**Description** Handles an authentication response.
		Sequence Diagram



Data



INPUT

- AUB Response

OUTPUT

- Authentication (yes/no)
- Value attribute list
  - Filled attribute list: attributes with found values
  - Empty attribute list: attributes without values

CONFIG FILES NEEDED

- None

Actions

- Check AUB Request

  (If the authentication not success → Handle Error CPAUB0601)
- More attributes?: Evaluate if all attributes needed are available or if more attributes should be requested to MS Attribute Supply.

  (Also evaluate if some attributes have to be introduced by the user and verified)

Error Communications

- Send HTML KO.3 to the Citizen-Browser.

  (If a Handle Error succeed).

Table 13 –Description sequence Authentication on behalf of (part 1) in A-PEPS

2.4.3.1.2 Sequence diagram AUB, part 2

Same as 2.4.2.1.3

2.4.3.1.3 Description part 2

Same as 2.4.2.1.4

2.4.4 Version Control (PEPS)

The version control verifies the software and configuration versions of PEPS colleagues and publishes the results to its colleagues and the national service providers.

The *version control* process is carried out under the standard facility that launches processes periodically, like Task Manager in Windows or cron under Unix. If any changes are found in its colleague's or SP configurations, an alert is sent to the administrator(s), in order to inform him/her that some compatibility tests should be performed.

In any case the version control files are generated, which allow on one hand corresponding service providers to be informed of relevant changes and adapt their country selector, and on the other hand allows colleagues to inform their administrators in order to execute relevant tests.

2.4.4.1 Sequence diagram VCP

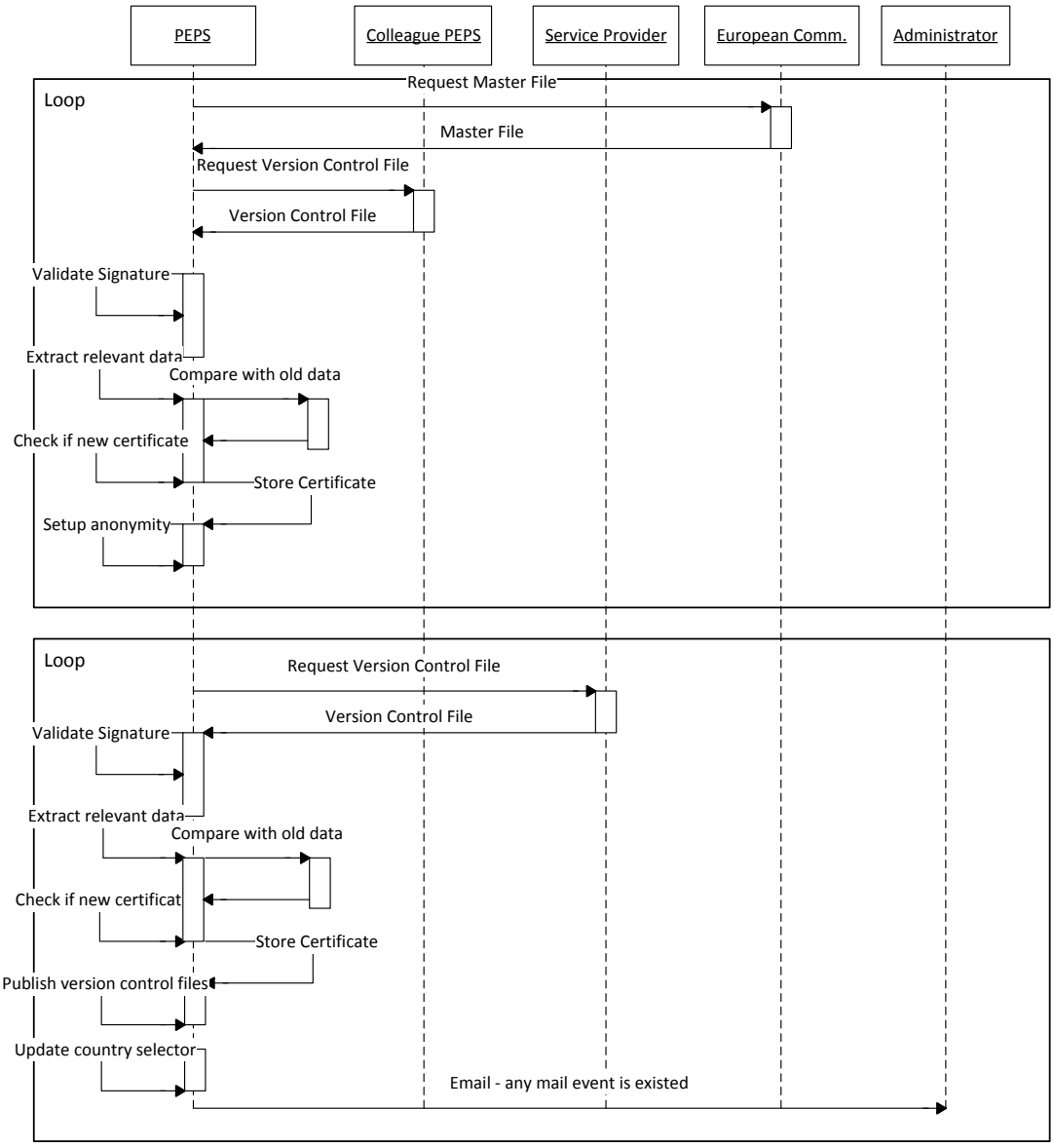
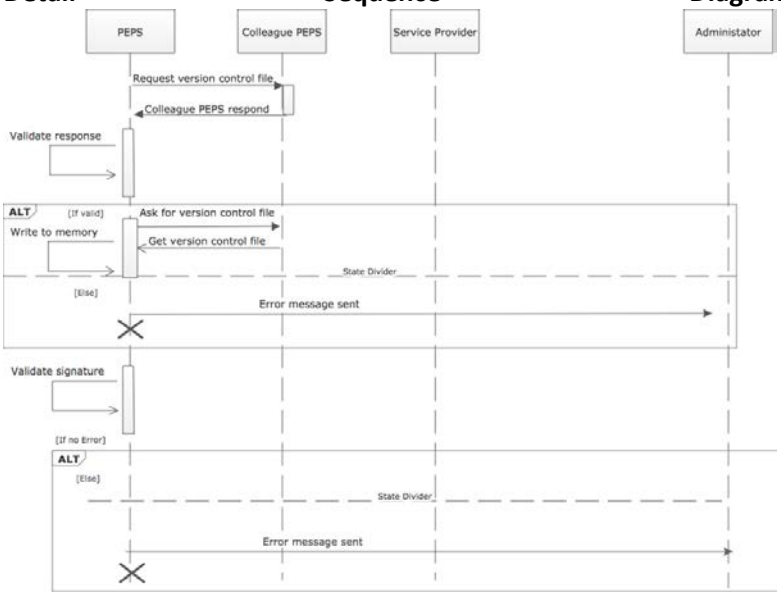


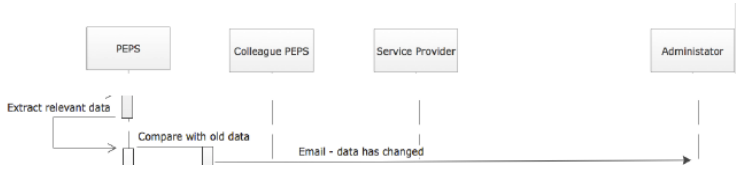
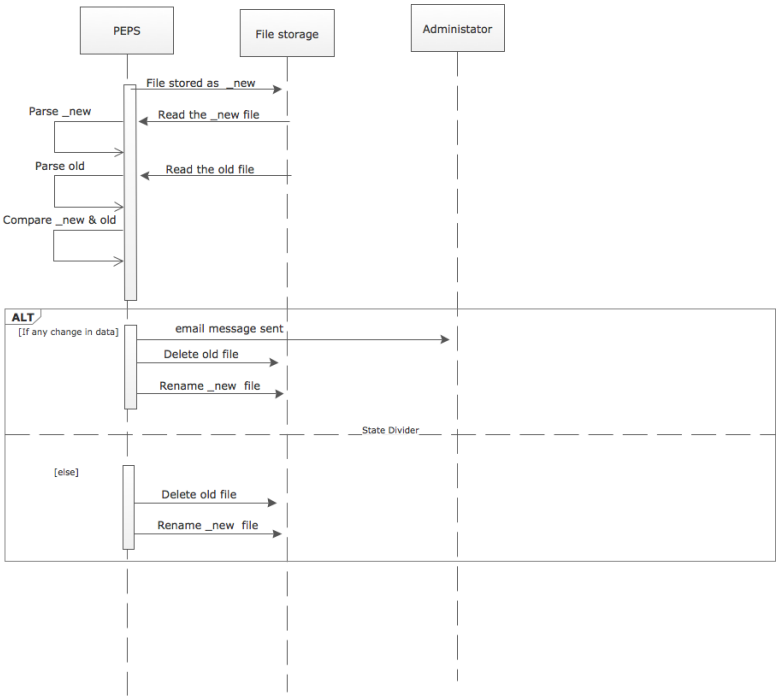
Figure 20: Sequence diagram Version Control, in PEPS

2.4.4.2 Description VCP

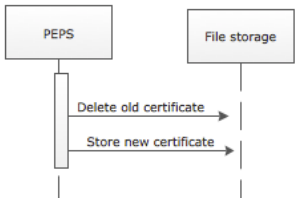
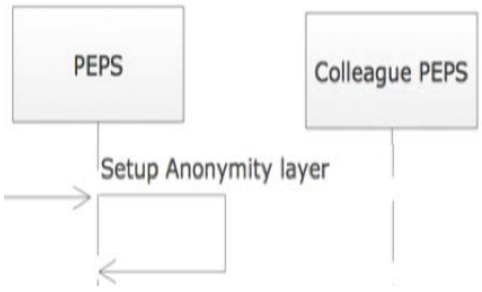
Message sequences (interactions)		***Description***
1	VCP-ACT-1	**Description** The PEPS/V-IDP sends scheduled request for Master File to European Commission. The link to each version control file is stored in the Master File stored at http://ec.europa.eu/STORK/PEPS-Master.xml, the exact location is still to be determined. The master file is signed by the European Commission. If a master file is in place, European Commission responds with the master file (PEPS-Master.xml) which contains all countries version control file links. It also contains countries id and name. If the master file is not in place the results in an error. The PEPS/V-IDP validates the signature of the master file. In case the signature is invalid the PEPS/V-IDP sends an error message to the PEPS/V-IDP Administrator and obtains countries version control file links from Peps.xml.
		External Actors PEPS/V-IDP job scheduled
		Action 1 **Function Validate Master file** Request master file If file is available Get file Validate signature Save file to memory Gets PEPS trusted list from master file. Else Return error Send error to Administrator Write to error log Gets PEPS trusted list from Peps.xml. **Data** INPUT Master file path(ex: http://ec.europa.eu/STORK/PEPS-Master.xml.) OUTPUT Lisf of trusted PEPS Version Control File paths.

Message sequences (interactions)		***Description***
		Success/error (VCP_ERR_1 / VCP_ERR_2) CONFIG FILES NEEDED No config file needed
2	VCP-ACT-2	**Description** The PEPS/V-IDP sends a request for configuration to a colleague PEPS. The configuration is stored in the Version Control File (VCF) stored at the location indicated in the masterfile, normally https://xxx.xxx.xx/PEPS/XX-info.xml. The version control file is signed by the Colleague PEPS/V-IDP (XX-info.xml where XX is the country code of the colleague PEPS. i.e Spain: ES-info.xml). If a version control file is in place, the colleague PEPS/V-IDP responds with the version control file (ES-info.xml) which contains configuration settings and all needed certificates. It also contains the configuration settings and certificate for its colleague PEPS to act as an anonymity node and all the certificates and settings for its subordinate non authoritative nodes. If the version control file is not in place the request for configuration results in an error. The PEPS/V-IDP validates the signature and format of the version control file. In case the signature is invalid the PEPS/V-IDP sends an error message to the PEPS/V-IDP Administrator and stops the process for current PEPS/VIDP's version control file.
		Sequence Diagram 
		Detail Sequence Diagram 

Message sequences (interactions)		***Description***
		Action 1 **Function Validate PEPS Version control file** Request version control file If file is available Get file Validate signature Save file to memory Else Return error Send error to Administrator Write to error log **Data** INPUT Version control file path (ex: https://xxx.xxxx.xx/PEPS/XX-info.xml) OUTPUT Success/error (VCP_ERR_3 / VCP_ERR_4) CONFIG FILES NEEDED No config file needed
3	VCP-ACT-3	**Description** If this is the first time a version control file is fetched from the Colleague PEPS/V-IDP and the signature and format are correct and trusted, the PEPS stores the Version control file and the process continues directly to VCP-ACT-5. If this is not the first time the version control file is fetched and the signature and format of the file are correct and trusted the PEPS/V-IDP stores the version control file adding _new to the file name. The PEPS/V-IDP reads the previous version control file and the _new version control file; extracts relevant data from the files and compares the new data to the data in the previous version control file. If there is no change in the data, the PEPS/V-IDP deletes the previous version control file and removes "_new" from the name of the new version control file (the new file is renamed). If there is any change in the data the PEPS/V-IDP adds change information to email that will be sent to the PEPS/V-IDP Administrator. If the change in data includes no new certificates, the process continues with VCP-ACT-5.

Message sequences (interactions)	*Description*
	Sequence Diagram 
	Detail Sequence Diagram 
	External Actors • No external actors
	Action 1 **Function check for previous version control file for colleague PEPS** If no previous file is available Go to step VCP-ACT-5 Else Store the version control file as XX-info.xml_new **Data** INPUT Previous version control file OUTPUT XX-info.xml_new / **Go to step VCP-ACT-5** CONFIG FILES NEEDED

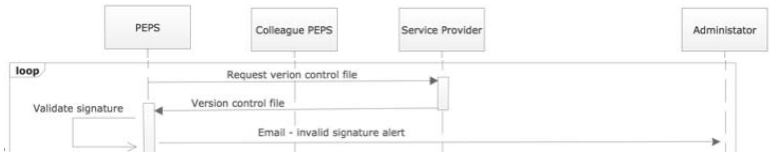
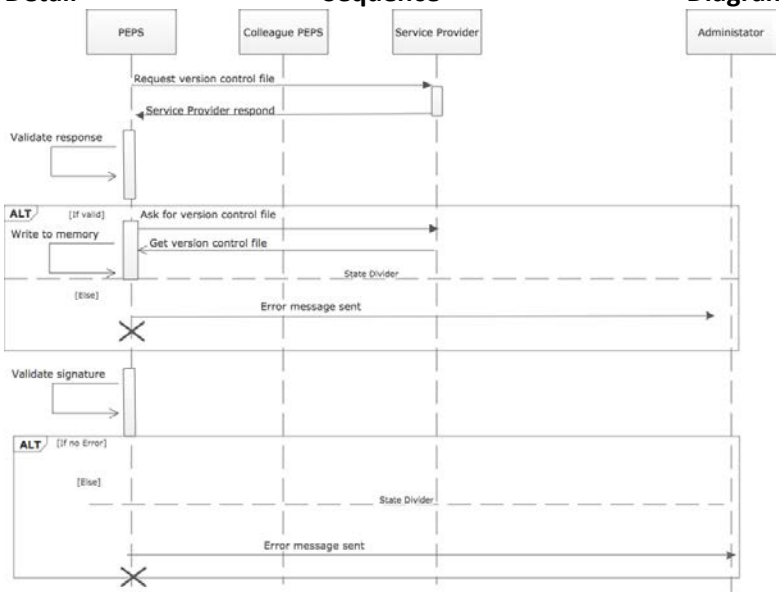
Message sequences (interactions)		*Description*
		No config files needed _____ **Action 2** **Compare data** Store the version control file as XX-info.xml_new Parse the file XX-info.xml_new into variables Read the old version control file XX-info.xml Parse the file XX-info.xml into variables Compare all data from XX-info.xml_new with the data in XX-info.xml If any change in data Add information about changes to email that will be sent to administrator Delete XX-info.xml Rename XX-info.xml_new to XX-info.xml If no new certificates Go to step VCP-ACT-5 Else Retrieve the name of old certificate file from the old version control file Delete XX-info.xml (the old version control file) Rename XX-info.xml_new to XX-info.xml Go to step VCP-ACT-4 **Data** INPUT Previous version control files OUTPUT New version of colleague PEPS version control file CONFIG FILES NEEDED No config files needed
4	VCP-ACT-4	**Description** If the change in data includes any new certificate, the PEPS/V-IDP checks and removes any out dated certificate before it stores all new certificates.

Message sequences (interactions)		*Description*
		Sequence Diagram 
		External Actors • No external actor
		Action **Function check for out-dated certificates** Look for old certificate If old certificate is found Delete old certificate Store new certificate **Data** INPUT Name of the old certificate file from old version control file New Certificates retrieved in previous step OUTPUT Success / Error (VCP_ERR_6, VCP_ERR_7) CONFIG FILES NEEDED No config files needed
5	VCP-ACT-5	**Description** The PEPS/V-IDP determines if it has the anonymity layer software installed and configured. If so, it determines the version of the software and configuration files
		Sequence Diagram 
		Action 1 **Determine if Anonymity is installed** **If Anonymity is installed**

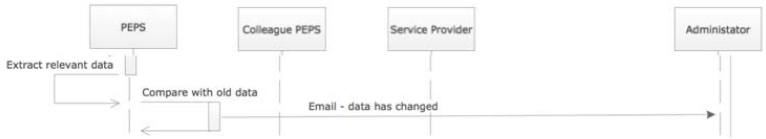
Message sequences (interactions)	*Description*

	Determine version of the Anonymity software
	Determine version of configuration files
	Data
	INPUT
	Software installation status
	OUTPUT
	True (and Anonymity software version) / False
	CONFIG FILES NEEDED
	No config files needed
	_____
	Action 2
	Function update PEPS Anonymity Data
	if PEPS is not anonymity node:
	if PEPS was an anonymity node:
	Mark the node for deletion.
	Mark all of this node's subordinate non-authoritative nodes for deletion.
	end
	Insert or update node certificate on the certificate store
	Insert or update node settings on the database
	for each subordinate non-authoritative node:
	Insert or update node certificate on the certificate-store
	Insert or update node settings on the database
	for each subordinate non-authoritative node that is on the database but is no longer on the Control Version File:
	Delete node certificate from the certificate store
	Delete node settings from the database
	Data
	INPUT
	PEPS node settings
	PEPS node certificate
	List of subordinate non-authoritative nodes with:
	Node certificate

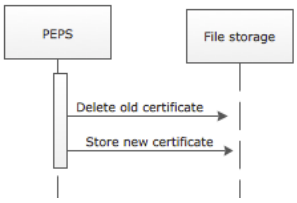
Message sequences (interactions)	*Description*
	Non-authoritative node settings Status OUTPUT CONFIG FILES NEEDED Anonymity config file _____ **Action 3** **Function clean node database** for each node marked for deletion whose exclusion period has expired: Delete node certificate from the certificate store Delete node settings from the database **Data** INPUT Database OUTPUT CONFIG FILES NEEDED Anonymity config file _____ **Action 4** **Function update own control version file** if own anonymity certificate or settings change, own Control Version File must be updated. if subordinate non-authoritative node list changes (additions, deletions, setting or certificate changes), own Control Version File must be updated. **Data** INPUT New settings / certificates OUTPUT New version control file CONFIG FILES NEEDED

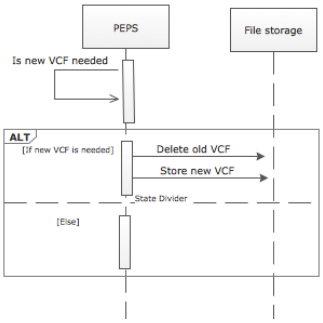
Message sequences (interactions)		*Description*
		Anonymity config file
6	VCP-ACT-6	**Description** The PEPS/V-IDP sends scheduled request for configuration to a service provider (SP). The configuration is stored in Version Control File (VCF) stored at https://xxx.xxx.xx/PEPS/XX-XXX-info.xml. The version control file is signed by the service provider (XX-info.xml were XX is the country code of the service provider and XXX is unique service provider code. For example for Arion bank in Iceland IS-AB1-info.xml). If version control file is in place, the service provider responds with the version control file (xx-xxx-info.xml) file which contains configuration settings and all needed certificates. If version control file is not in place the request for configuration results in error. The PEPS/V-IDP validates the signature and format of the version control file. In case the signature is invalid the PEPS/V-IDP sends an error message to the PEPS/V-IDP Administrator and stops the process for current SP's version control file.
		Sequence Diagram 
		Detail Sequence Diagram 

Message sequences (interactions)		*Description*
		External Actors • **PEPS/V-IDP scheduled job**
		Action 1 **Function** **Obtain** **SP** **list** **Data** INPUT Peps.xml path. OUTPUT Success/error (VCP_ERR_8 / VCP_ERR_9) CONFIG FILES NEEDED List of trusted SP _____ **Action 2** **Function Validate SP Version control file** Request version control file If file is available Get file Validate signature Save file to memory Else Return error Add information about error to email that will be sent to AdministratorWrite to error log **Data** INPUT Version control file path (ex: https://xxx.xxx.xx/PEPS/XX-XXX-info.xml.) OUTPUT Success/error (VCP_ERR_3 / VCP_ERR_4) CONFIG FILES NEEDED No config file needed
7	VCP-ACT-7	**Description** If this is the first time a version control file is fetched from the SP and the signature and format are correct and trusted, the PEPS stores the Version control file and the process continues directly

Message sequences (interactions)	*Description*
	to VCP-ACT-10. If this is not the first time the version control file is fetched and the signature and format of the file is correct and trusted the PEPS/V-IDP stores the version control file adding _new to the file name. The PEPS/V-IDP reads the previous version control file and the _new version control file; extracts relevant data from the files and compares the new data to the data in the previous version control file. If there is no change in the data, the PEPS/V-IDP deletes the previous version control file and removes "_new" from the name of the new version control file (the new file is renamed). If there is any change in the data the PEPS/V-IDP sends an email to the PEPS/V-IDP Administrator. If the change in data does not include new certificates, the process continues with VCP-ACT-9.
	Sequence Diagram 
	External Actors No external actors
	Action 1 **Function check for previous version control file for SP** If no previous file is available Go to step VCP-ACT-9 Else Store the version control file as XX-XXX-info.xml_new **Data** INPUT Previous version control file OUTPUT XX-XXX-info.xml_new / **Go to step VCP-ACT-9** CONFIG FILES NEEDED No config files needed

Message sequences (interactions)		***Description***
		Action 2 **Compare data** Store the version control file as XX-XXX-info.xml_new Parse the file XX-XXX-info.xml_new into variables Read the old version control file XX-XXX-info.xml Parse the file XX-XXX-info.xml into variables Compare all data from XX-XXX-info.xml_new with the data in XX-XXX-info.xml If any change in data Add information about changes to email that will be sent to administrator Delete XX-XXX-info.xml Rename XX-XXX-info.xml_new to XX-info.xml If no new certificates Go to step VCP-ACT-9 Else Retrieve the name of old certificate file from the old version control file. Delete XX-XXX-info.xml (the old version control file) Rename XX-XXX-info.xml_new to XX-XXX-info.xml Go to step VCP-ACT-8 **Data** INPUT Previous version control files OUTPUT New version of SP version control files CONFIG FILES NEEDED No config files needed
8	VCP-ACT-8	**Description** If the change in data includes any new certificate, the PEPS/V-IDP checks and removes any out dated certificate before it stores all new certificates.

Message sequences (interactions)		***Description***
		Sequence Diagram 
		External Actors No external actor
		Action **Function check for out-dated certificates** Look for old certificate If old certificate is found Delete old certificate Store new certificate **Data** INPUT Name of the old certificate file from old version control file New Certificates retrieved in previous step OUTPUT Success / Error (VCP_ERR_6, VCP_ERR_7) CONFIG FILES NEEDED No config files needed
9	VCP-ACT-9	**Description** The PEPS/V-IDP checks if any mail event exists, the PEPS/V-IDP prepares a table formatted email. Then the PEPS/V-IDP sends this email to Administrator.
		Sequence Diagram
		External Actors No external actor
		Action **Function check for email event** If mail event is existed Prepare table formatted email Send email to Administrator **Data**

Message sequences (interactions)		***Description***
		INPUT Mail events(certificate changes, VCF data changes, any error..) OUTPUT Success / Error (VCP_ERR_6, VCP_ERR_7) CONFIG FILES NEEDED
10	VCP-ACT-10	**Description** The PEPS/V-IDP determines if new version control files are needed based on if there has been a change in configuration files which are referred in the Version Control File and Colleague PEPS version control files. The PEPS/V-IDP has two type version control files which one of them is for Colleague PEPS and the other is for SPs in own country.
		Sequence Diagram 
		External Actors No external actors
		Action 1 **Store Current Version Control File values** Read current Version Control File Parse current Version Control File Put values into variables **Data** INPUT Own version control file from local storage OUTPUT Variables containing the Version Control File values CONFIG FILES NEEDED Own Version Control File

Message sequences (interactions)	*Description*
	Action2 **Determine if certificates have changed** Read newest Certificate files from certificate storage Compare new certificates with certificates in VCF If own certificates have changed Update corresponding variable value **Data** INPUT Own Certificates stored in local certificate storage OUTPUT Updated value's for variables containing VCF certificates CONFIG FILES NEEDED No config files needed
	Action 3 **Determine if configuration files have changed** Read configuration files and compare if data has changed If data in configuration files has changed Update corresponding variable value **Data** INPUT Configuration files containing info for the VCF OUTPUT Updated variable values CONFIG FILES NEEDED Configuration files containing VCF info
	Action 4 **Update information from Colleague PEPS'es/V-IDP's** Loop for all colleague PEPS/V-IDP VCF files Read from file storage colleague PEPS/V-IDP VCF file Compare values in Colleague PEPS/V-IDP VCF file with values in own VCF file If data has changed Update corresponding variable value

Message sequences (interactions)		*Description*
		End loop **Data** INPUT Colleague PEPS VCF files OUTPUT Updated variable values CONFIG FILES NEEDED Colleague PEPS VCF files
		Action 5 **Build new Version Control files** Use stored variables to build new VCFs Delete old Version control files Store new Version control files(PEPS VCF and SPs VCF) **Data** INPUT Stored variables OUTPUT New version control files(PEPS VCF and SPs VCF) CONFIG FILES NEEDED No config files needed
11	VCP-ACT-11	**Description** Based on the new PEPS VCF file, country selector is updated.
		Sequence Diagram 
		External Actors No external actors
		Action 1 **Update country selector** Read current Version Control File Parse country selector values into variables Read the country selector Update the country selector Remove old country selector

Message sequences (interactions)		***Description***
		Save new Country selector **Data** INPUT Own version control file from local storage OUTPUT New country selector CONFIG FILES NEEDED Own PEPS Version Control File

Table 14 –Description Version Control in PEPS

3 V-IDP Architecture design

V-IDPs fulfil basically the same objectives as the PEPSs, but for the decentralised model (formerly "MW countries"):

- they form anchors of trust which allow to elevate the national circles of trust to European level and;
- they hide country specific things like organisation, available ID providers and national and domain-specific attribute providers to the outside world and just offer standardised data;

In this sense, V-IDPs are also described in this chapter although most functions and structures are common to both approaches. The main difference is that it is a distributed approach (no central instance, but each SP operating a V-IDP), fully irrelevant for this document.

3.1 System Context

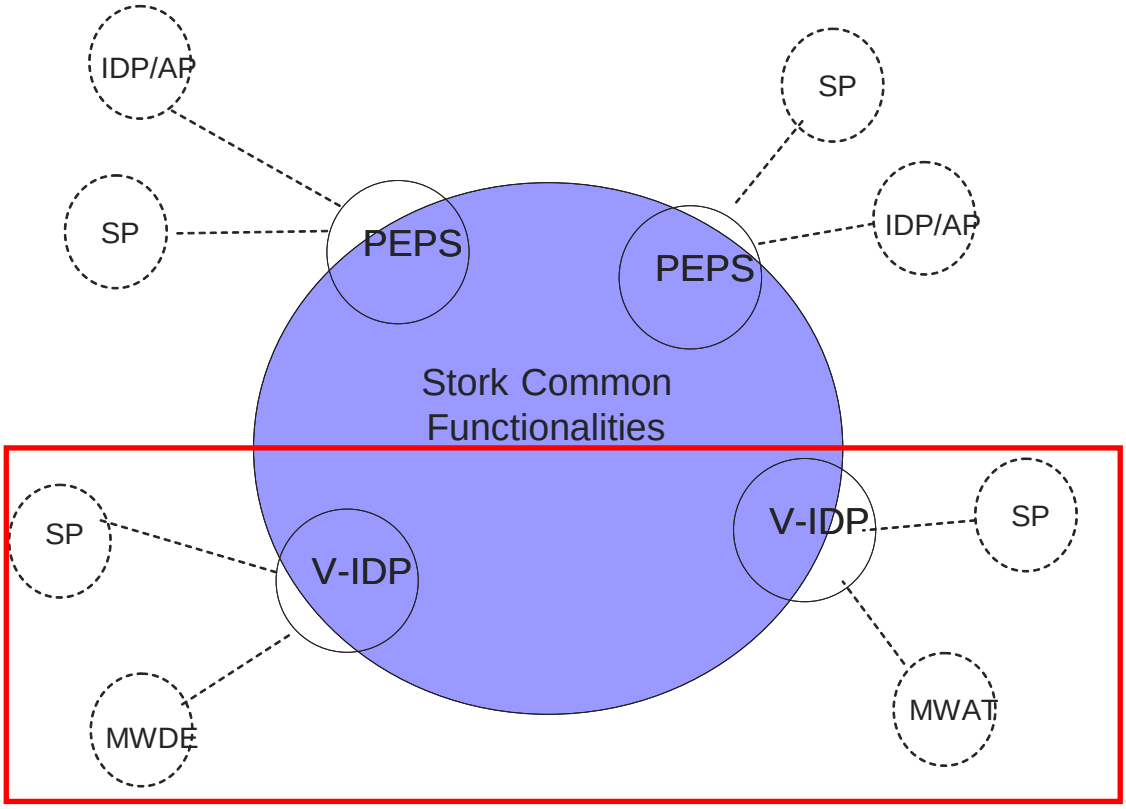


Figure 21: V-IDP System Context Diagram

3.2 Logical view

The main goal of the logical view is the decomposition of the system into subsystems. This can be done by component and/or class diagrams, showing the architecturally important components and their relationships. The sequence diagrams show the sequence of messages passed between objects using a vertical timeline.

Name	***Description***
V-IDP@SP	V-IDP@SP represents the V-IDP on the service provider side (comparable to S-PEPS).

V-IDP@PEPS	V-IDP@PEPS represents the V-IDP on the PEPS side.
Table Cell 7	Table Cell 8

Table 15 –Meaning of the different V-IDPs

The following sequence diagrams are similar to the diagrams in section 2.4, but are slightly modified and replicated here to maintain the document's clarity. The main difference is that – depending on the case – either the S-PEPS or the C-PEPS is represented by a V-IDP.

3.2.1 Authentication on behalf of

For description refer to section 2.4. Please note that for the V-IDP the Powers (for digital signature) process is the same as the Authentication on behalf of process.

3.2.1.1 Sequence Diagram Authentication on behalf of, mixed-model: UC-AUB-MP

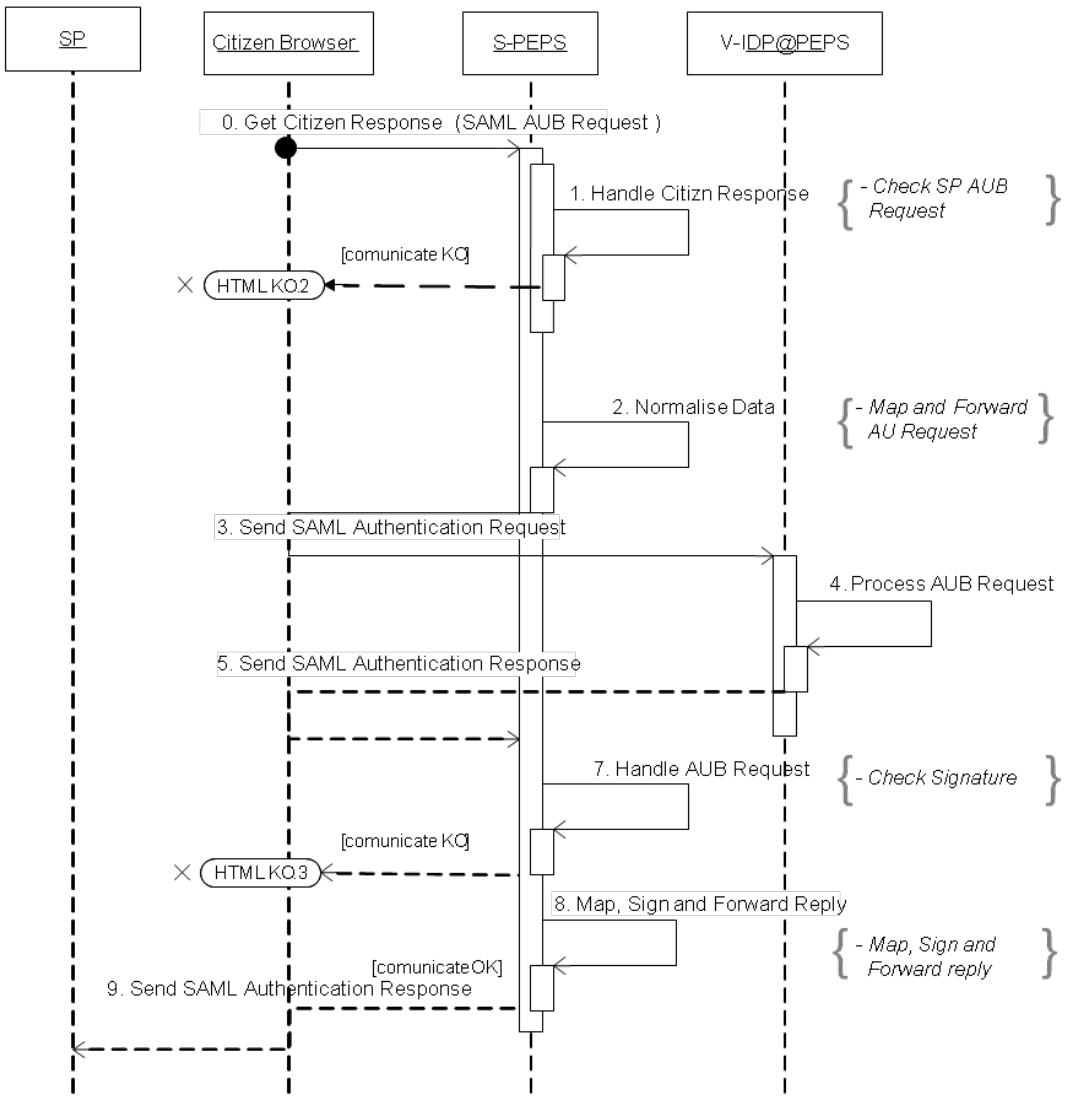
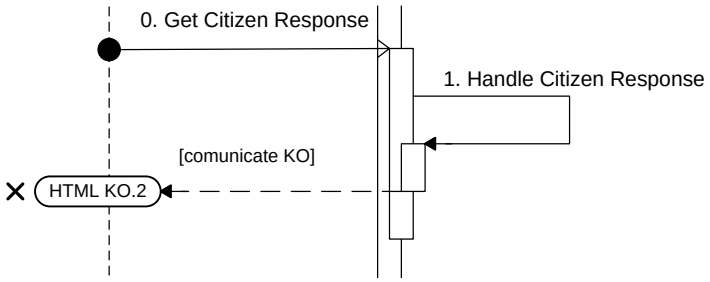
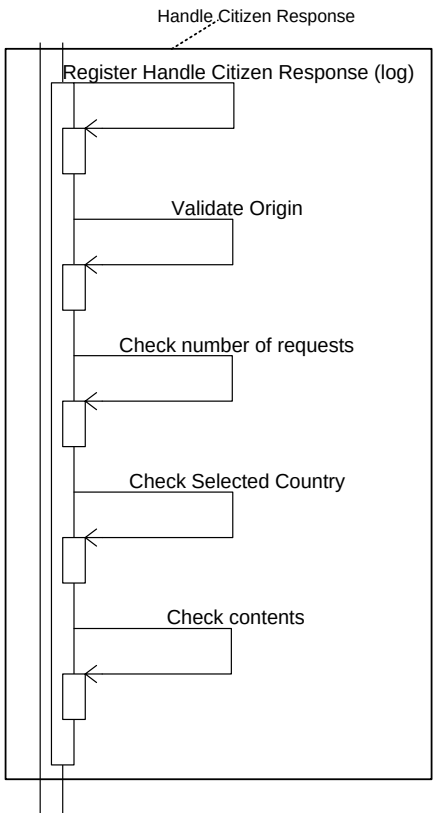
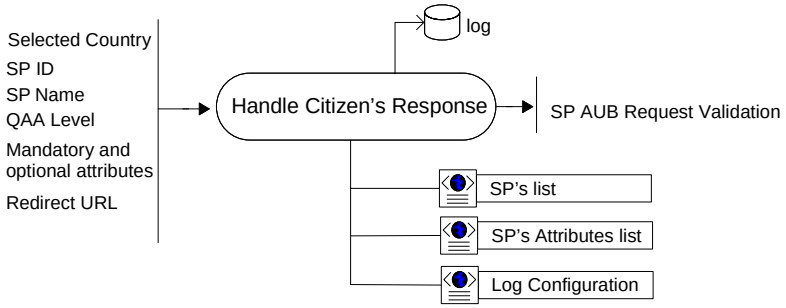
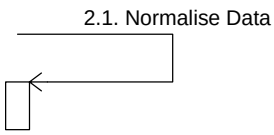


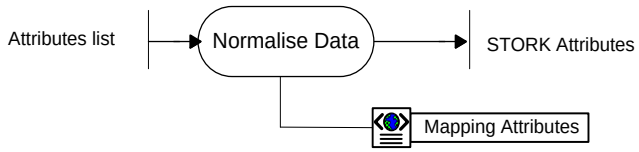
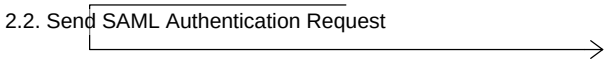
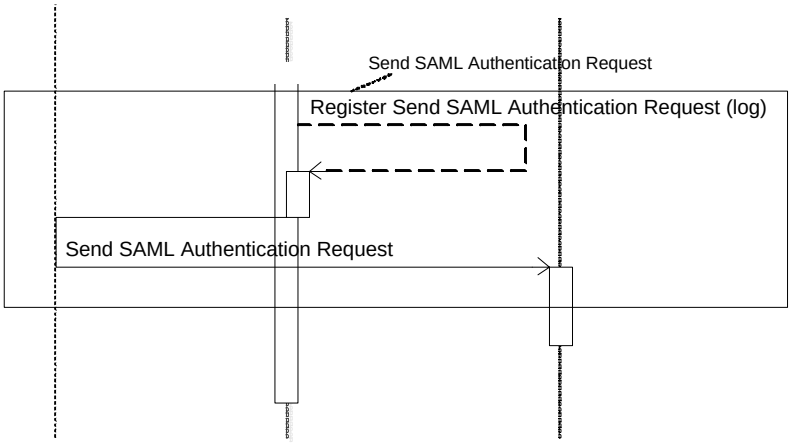
Figure 22: V-IDP Sequence Diagram UC-AUB-MP

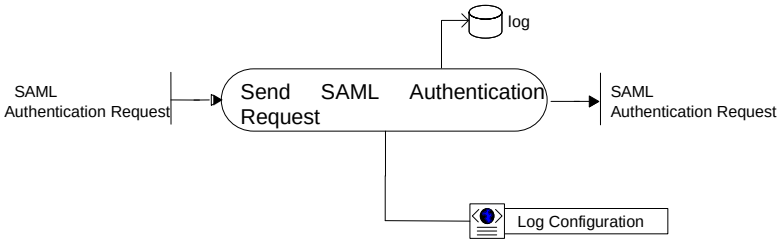
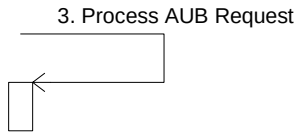
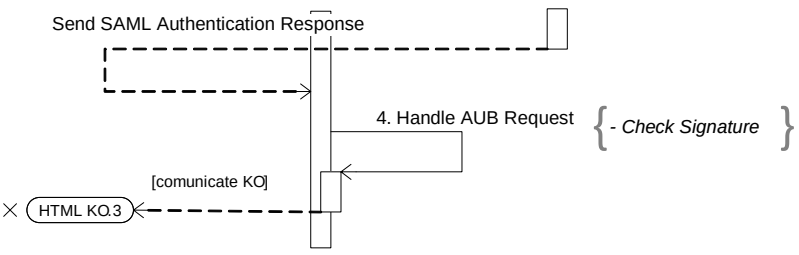
3.2.1.2 Description

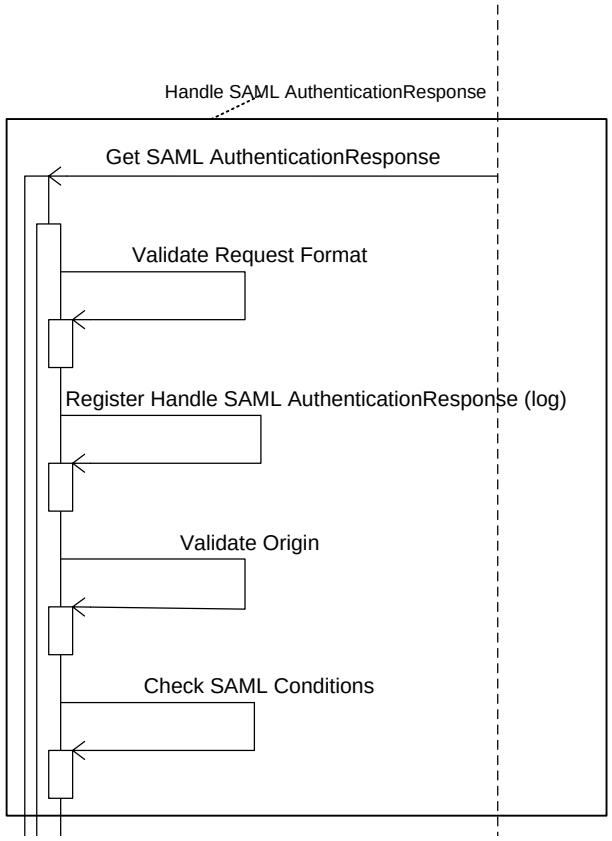
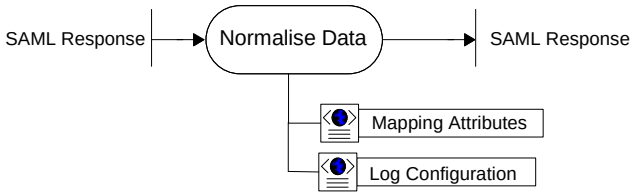
Message sequences (interactions)		*Description*
0-1	Handle Citizen's Response	**Description** Receives Citizen's reply, adds it to the AUB Request and checks AUB request validation (this task includes log activity). **Sequence Diagram**  **Detailed Sequence Diagram**  **External Actors** • Citizen-Browser

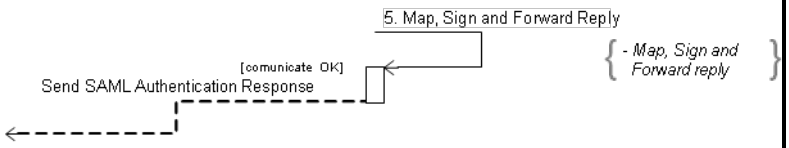
Message sequences (interactions)	*Description*
	Data  INPUT • Citizen's selected country • SP ID • SP Name • QAA Level • Mandatory and optional Attributes list includes the SP representation/mandate requirements • Redirect URL OUTPUT • SP AUB Request Validation COMMON CONFIG FILES NEEDED • SP's list • SP's Attributes List • Log Configuration
	Error Communications • Send HTML KO.2 to the Citizen-Browser. If a Handle Error succeed.
	Actions • Get Citizen's Country Selected (Citizen's Reply) • Validate Origin (SP ID, QAA Level). (If SP ID and QAA Level are missing → Handle Error SPAUB0401) o If Authentication Type is based on SP's list: ✓ Check SP (SP ID, SP List): SP in SP's List. (If SP is not in the list → Handle Error SPAUB0402)

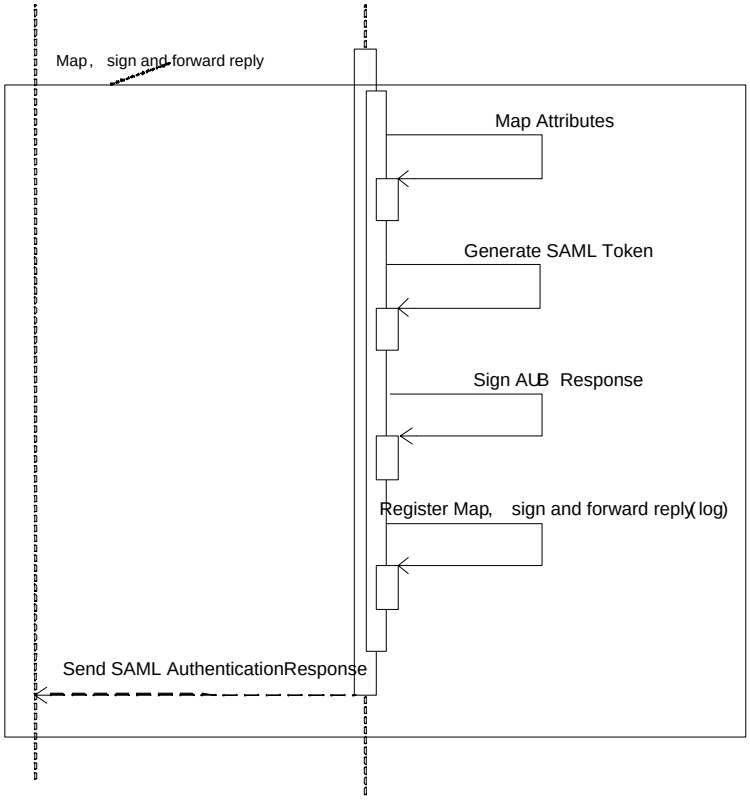
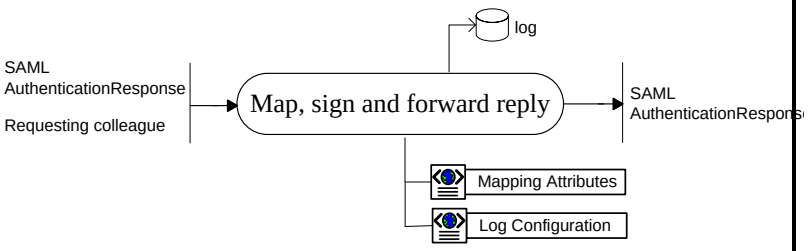
Message sequences (interactions)		*Description*
		✓ Else, check SP Domain (SP Request Domain, Redirect URL): validate if the request domain and Redirect URL matches the registered SP Domain. (If the request domain or Redirection URL is not correct → Handle Error SPAUB0403) o Check number of requests (SP ID, 60): number of requests in the last period of 60 seconds. (If this number is greater than or equal to the maximum number - to avoid DoS → Handle Error SPAUB0404) o Check Selected Country (SelectedCountry) (If selected country is not a valid country → Handle Error SPAUB0405) o Check contents (Mandatory and optional Attributes list). (If any Mandatory or optional Attribute isn't in SP's Attributes List → Handle Error SPAUB0406) o Register Handle Citizen Response (Citizen, S-PEPS, "Select Country")
2	Map and Forward AUB Request	**Normalise data and send AUB request to colleague V-IDP@PEPS** This task includes some internal activities to normalise all the received data, log activity and send to Colleague V-IDP@PEPS. 2.1 Normalise data 2.2 Get SAML Authentication Request
2.1	Normalise data	**Description** Normalise all received data, mapping of the MS values to STORK nomenclature. For AUB this, for instance, includes mapping of the SP requesting a MS-specific definition of powers to the STORK taxonomy of mandates/representations.
		Sequence Diagram 

Message sequences (interactions)		*Description*
		Data  INPUT • Attribute list with values OUTPUT • Attribute names and data values in STORK nomenclature. CONFIG FILES NEEDED • Mapping Attributes: Map between STORK names/values and MS names/values for the attributes (text values are not included)
2.2	Send SAML Authentication Request	**Description** Send AUB Request to Colleague V-IDP@PEPS.
		Sequence Diagram 
		Detailed Sequence Diagram 

Message sequences (interactions)		*Description*
		Data  INPUT • SAML Authentication Request OUTPUT • SAML Authentication Request CONFIG FILES NEEDED • Log Configuration
		Actions • Register Send SAML Authentication Request (S-PEPS, V-IDP@PEPS, "Map and Forward AUB Request")
3	Process AUB Request	**Description** The request is processed in the Colleague V-IDP@PEPS. The authentication on behalf of is performed with the user and the security token created. This is MS specific.
		Sequence Diagram 
4	Handle AUB Request	**Description** S-PEPS receives SAML AuthenticationResponse through the Citizen-Browser issued by Colleague V-IDP@PEPS. S-PEPS validates SAML AuthenticationResponse signature.
		Sequence Diagram 

Message sequences (interactions)	*Description*
	Detailed Sequence Diagram 
	External Actors • Citizen-Browser: The S-PEPS receives the SAML AUB Response from a Colleague V-IDP@PEPS through the Citizen-Browser.
	Data  INPUT • SAML Response OUTPUT • SAML Response CONFIG FILES NEEDED • Mapping attributes • Log Configuration

Message sequences (interactions)		*Description*
		Error Communications • Send HTML KO.3 to the Citizen-Browser. If a Handle Error succeed.
		Actions • Receive SAML AuthenticationResponse ( ) • Register Handle SAML AuthenticationResponse (V-IDP@PEPS, S-PEPS, "Check Signature") • Validate Response Format (SAML) (If the format is not correct → Handle Error SPAUB2201) • Validate Origin (Colleague V-IDP@PEPS): Validate Colleague V-IDP@PEPS Signature. (If the origin is not correct → Handle Error SPAUB2202) • Check SAML Conditions (notBefore, notAfter, etc.) (If conditions are not fulfilled → Handle Error SPAUB2203)
5	Map, Sign and forward reply	**Description** Maps, generates the SAML token, signs and sends it to SP.
		Sequence Diagram 

Message sequences (interactions)	*Description*
	Detailed Sequence Diagram 
	External Actors • SP
	Data  INPUT • SAML AuthenticationResponse • Requesting colleague PEPS OUTPUT • SAML AuthenticationResponse. CONFIG FILES NEEDED • Mapping Attributes: Map between STORK names/values and MS values for the attributes (text values are not included)

Message sequences (interactions)	*Description*
	• Log Configuration
	Error Communications • None.
	Actions • Map STORK values to MS values (STORK Values, MS Values), e.g. STORK mandate/representation values to a MS specific mandate schema. The original source mandate also gets delivered to the SP for its auditing acceptability. • Generate SAML Response (Colleague SAML Request) • Sign SAML Authentication Response (SAML) • Send SAML Authentication Response ( ) • Register Map, sign and forward reply (S-PEPS, V-IDP@PEPS, "Map, sign an forward reply")

Table 16 – Description sequence Authentication on Behalf of, UC-AUB-MP

3.2.1.3 Sequence Diagram Authentication on behalf of, mixed-model: UC-AUB-PM

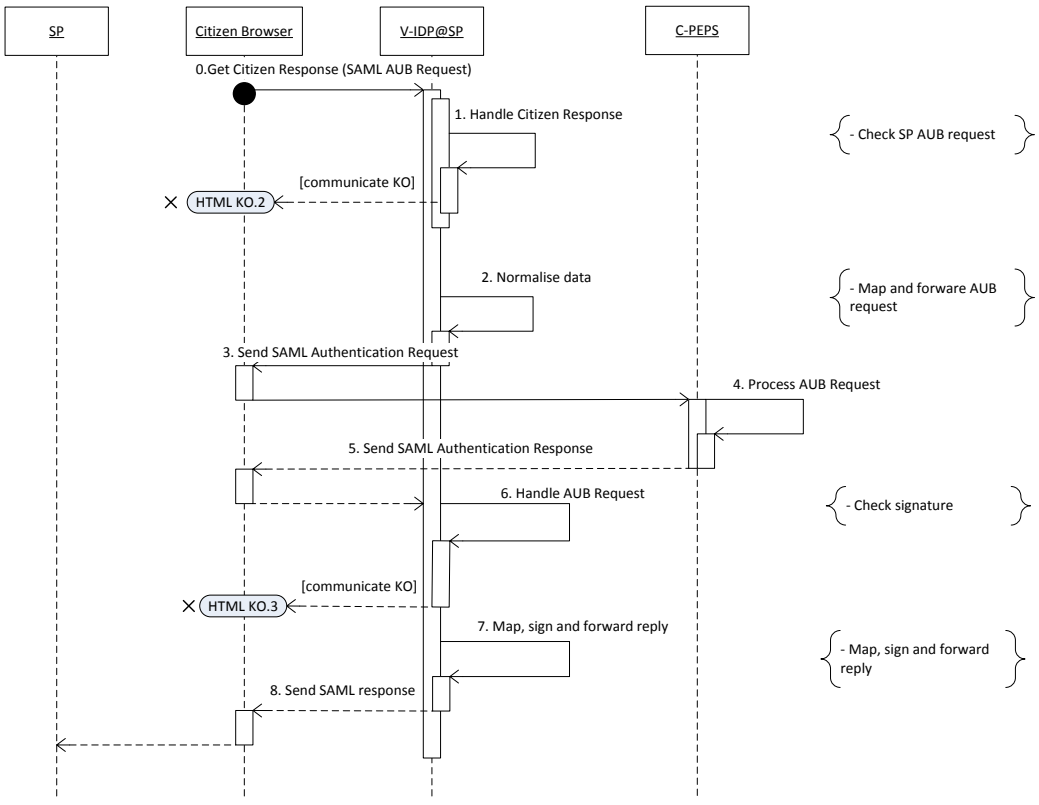
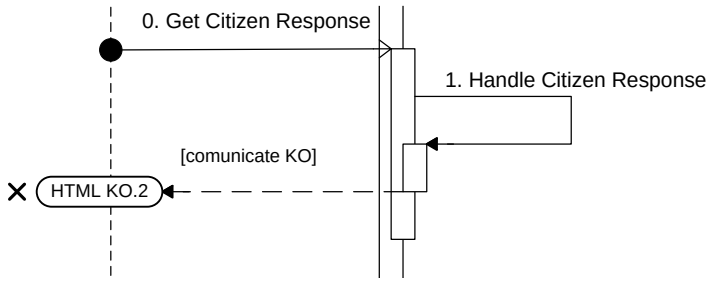
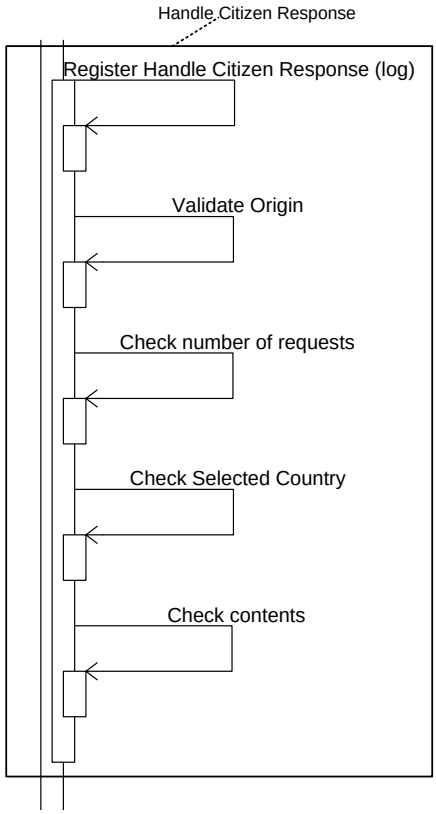
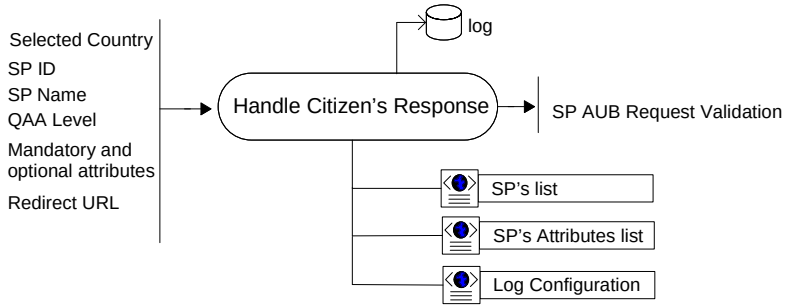


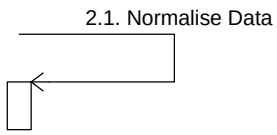
Figure 23: V-IDP Sequence Diagram UC-AUB-PM

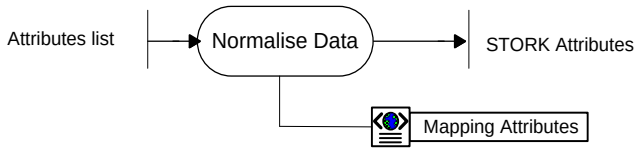
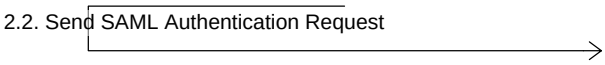
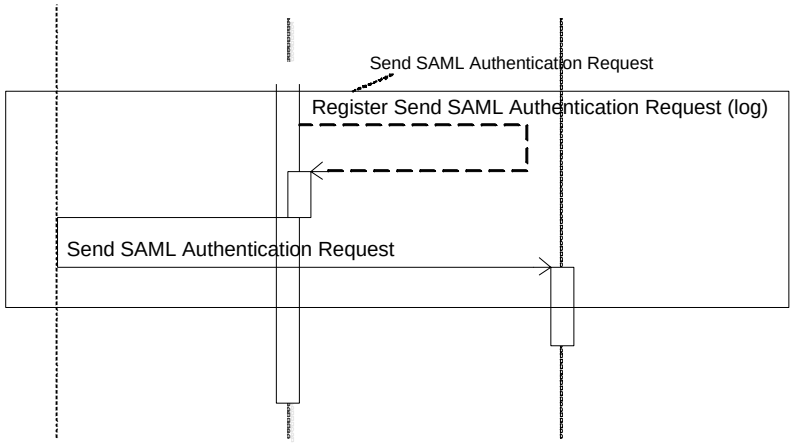
Please note that the V-IDP@PEPS is included with the C-PEPS. It behaves on both sides as a PEPS. Its function is to concentrate the requests, achieving that changes (new SPs) in a MW country do not affect any PEPS. As it does not contribute functionally, for ease of reading this component is omitted.

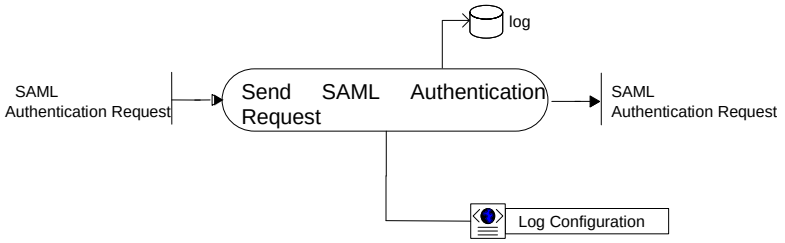
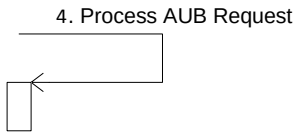
3.2.1.4 Description

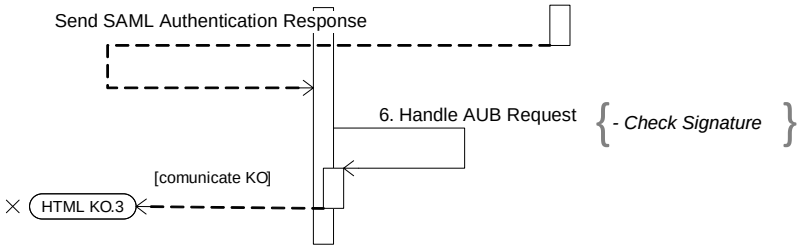
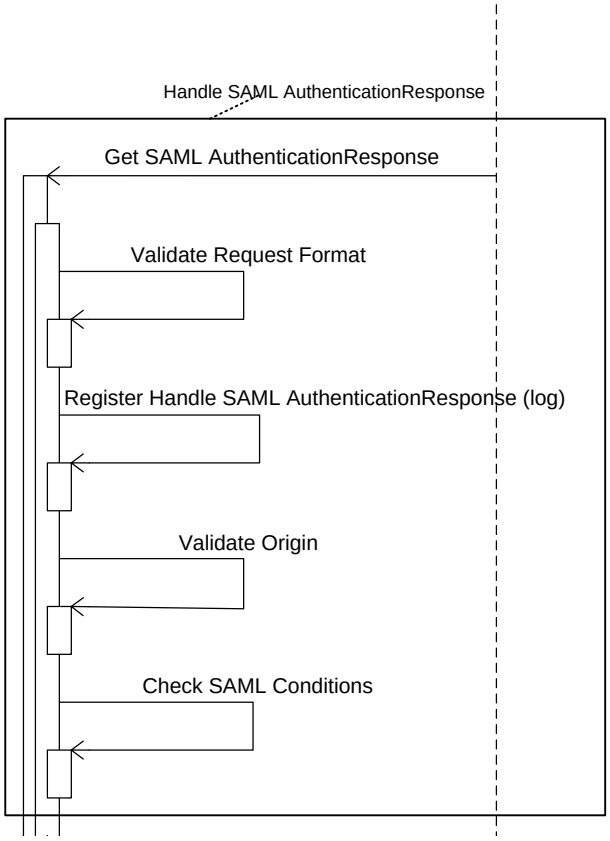
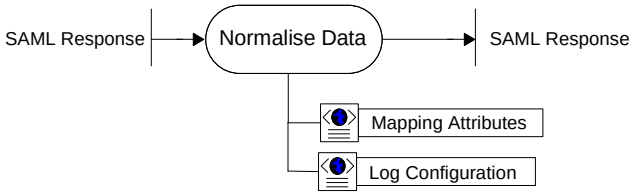
Message sequences (interactions)		***Description***
0-1	Handle Citizen's Response	**Description** Receives Citizen's reply, adds it to the AUB Request and checks AUB request validation (this task includes log activity). **Sequence Diagram**  **Detailed Sequence Diagram**  **External Actors** • Citizen-Browser

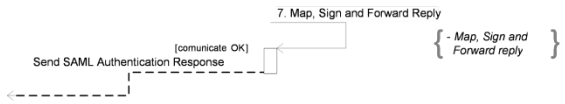
Message sequences (interactions)	*Description*
	Data  INPUT • Citizen's selected country • SP ID • SP Name • QAA Level • Mandatory and optional Attributes list includes the SP representation/mandate requirements • Redirect URL OUTPUT • SP AUB Request Validation COMMON CONFIG FILES NEEDED • SP's list • SP's Attributes List • Log Configuration
	Error Communications • Send HTML KO.2 to the Citizen-Browser. If a Handle Error succeed.
	Actions • Get Citizen's Country Selected (Citizen's Reply) • Validate Origin (SP ID, QAA Level). (If SP ID and QAA Level are missing → Handle Error SPAUB0401) o If Authentication Type is based on SP's list: ✓ Check SP (SP ID, SP List): SP in SP's List. (If SP is not in the list → Handle Error SPAUB0402)

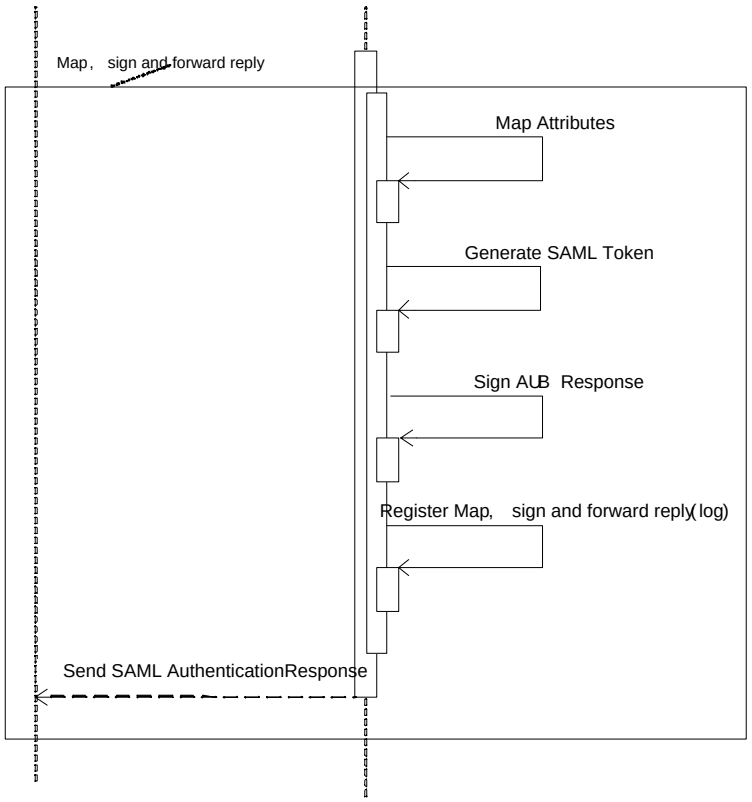
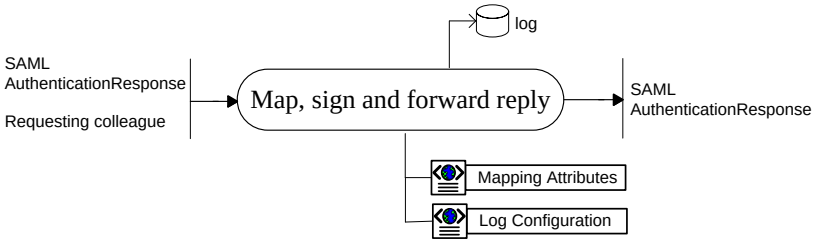
Message sequences (interactions)		*Description*
		✓ Else, check SP Domain (SP Request Domain, Redirect URL): validate if the request domain and Redirect URL matches the registered SP Domain. (If the request domain or Redirection URL is not correct → Handle Error SPAUB0403) o Check number of requests (SP ID, 60): number of requests in the last period of 60 seconds. (If this number is greater than or equal to the maximum number - to avoid DoS → Handle Error SPAUB0404) o Check Selected Country (SelectedCountry) (If selected country is not a valid country → Handle Error SPAUB0405) o Check contents (Mandatory and optional Attributes list). (If any Mandatory or optional Attribute is not in SP's Attributes List → Handle Error SPAUB0406) o Register Handle Citizen Response (Citizen, V-IDP@SP, "Select Country")
2	Map and Forward AUB Request	**Normalise data and send AUB request to colleague C-PEPS** This task includes some internal activities to normalise all the received data, log activity and send to Colleague C-PEPS. 2.1 Normalise data 2.2 Get SAML Authentication Request
2.1	Normalise data	**Description** Normalise all received data, mapping the MS values to STORK nomenclature. For AUB this for instance includes mapping of the SP requesting a MS-specific definition of powers to the STORK taxonomy of mandates/representations.
		Sequence Diagram 

Message sequences (interactions)		*Description*
		Data  INPUT • Attribute list with values OUTPUT • Attribute names and data values in STORK nomenclature. CONFIG FILES NEEDED • Mapping Attributes: Map STORK names/values to MS names/values for the attributes (text values are not included)
3	Send SAML Authentication Request	**Description** Send AUB Request to Colleague C-PEPS.
		Sequence Diagram 
		Detailed Sequence Diagram 

Message sequences (interactions)		*Description*
		Data  INPUT • SAML Authentication Request OUTPUT • SAML Authentication Request CONFIG FILES NEEDED • Log Configuration
		Actions • Register Send SAML Authentication Request (V-IDP@SP, C-PEPS, "Map and Forward AUB Request")
4	Process AUB Request	**Description** The request is processed in the Colleague C-PEPS. The authentication on behalf of is performed with the user and the security token created. This is MS specific.
		Sequence Diagram 
5	Send SAML Authentication Response	**Description** Redirects the Response to the V-IDP@SP.
		Sequence Diagram 
6	Handle AUB Request	**Description** V-IDP@SP receives SAML AuthenticationResponse through the Citizen-Browser issued by Colleague C-PEPS. V-IDP@SP validates SAML AuthenticationResponse signature.
		Sequence Diagram

Message sequences (interactions)	*Description*
	
	Detailed Sequence Diagram 
	External Actors • Citizen-Browser: The V-IDP@SP receives the SAML AUB Response from a Colleague C-PEPS through the Citizen-Browser.
	Data  INPUT • SAML Response

Message sequences (interactions)		*Description*
		OUTPUT • SAML Response CONFIG FILES NEEDED • Mapping attributes • Log Configuration
		Error Communications • Send HTML KO.3 to the Citizen-Browser. If a Handle Error succeed.
		Actions • Receive SAML AuthenticationResponse ( ) • Register Handle SAML AuthenticationResponse (C-PEPS, V-IDP@SP "Check Signature") • Validate Response Format (SAML) (If the format is not correct → Handle Error SPAUB2201) • Validate Origin (Colleague C-PEPS): Validate Colleague C-PEPS Signature. (If the origin is not correct → Handle Error SPAUB2202) • Check SAML Conditions (notBefore, notAfter, etc.) (If conditions are not fulfilled → Handle Error SPAUB2203)
7	Map, Sign and forward reply	**Description** Maps, generates the SAML token, signs and sends it to SP.
		Sequence Diagram 

Message sequences (interactions)	*Description*
	Detailed Sequence Diagram 
	External Actors • SP
	Data  INPUT • SAML AuthenticationResponse • Requesting colleague PEPS OUTPUT • SAML AuthenticationResponse. CONFIG FILES NEEDED • Mapping Attributes: Map STORK names/values to MS values for the attributes (text values are not included) • Log Configuration

Message sequences (interactions)		*Description*
		Error Communications • None.
		Actions • Map STORK values to MS values (STORK Values, MS Values) , e.g. STORK mandate/representation values to a MS specific mandate schema. The original source mandate also gets delivered to the SP for its auditing acceptability. • Generate SAML Response (Colleague SAML Request) • Sign SAML Authentication Response (SAML) • Send SAML Authentication Response ( ) • Register Map, sign and forward reply (V-IDP@SP, C-PEPS, "Map, sign an forward reply")

Table 17 – Description sequence Authentication on Behalf of, UC-AUB-PM

3.2.1.5 Authentication on behalf of, middleware-model: UC-AUB-MM

In the pure middleware model, SP-requests are handled by the V-IDP@SP internally by routing it to its SPware. There is no difference to STORK 1. Given that just one country applying the decentralized deployment model (formerly referred to as "MW") is in STORK 2.0 and, thus the "on behalf" scenario between two such countries can not be piloted, no addotional description is needed here.

However, the pure solution will be developed, and as far as possible it will be tested.

4 Commodities

Commodities are processes, functions or other components which have arisen during the discussions in the project, and are (partially) described in earlier documents, especially in D4.9 Final version of the Functional Design [13], but do not fit into the common functionalities of the core processes. Some of them do not even fit in the central STORK structure (PEPS and V-IDP); but they will be designed for the Service Providers. Their appearance in this document is motivated by the fact that many (nearly all) service providers will need to use such functions, so common development and test will reduce the efforts spent on these issues.

4.1 eIdentifier encryption (National Identifier Privacy)

The common objectives for a global functioning of eIdentifier encryption are described in the functional design. The only pending issue to be described is the implementation. The following sections describe the four standard algorithms to be implemented.

4.1.1 Symmetric encryption

A probably secure pseudo-random permutation based on *Feistel* networks (symmetric encryption) was proposed by Luby, Michael, Rackoff, and Charles (see [3]). The security of this construction is proved provided its round function is a pseudorandom function. In practice, AES-128 is probably an easy choice as round function.

This construction is based on several linked encryptions, each with an independent key – the number of keys depends on the length of the input message and the encryption algorithm. That implies, for instance, 64 AES-128 encryptions for a 128 bytes message[4]. It is advised to keep the unique root identifier under 64 bytes to limit the number of keys to 16.

Figure 24 below shows the algorithm that is described below:

First, suppose that the message is shorter or equal to 256 bits.

1. The message is padded to 256 bits– any padding may be used
2. The message is split in two parts of 128 bits
3. Each F function represents one AES-128 encryption with a different key
4. The result is two 128 bits buffers that we concatenate to get the pseudo-random result

Let's call the function described above F256.

If the message is longer than 256 bits, but shorter or equal to 512 bits:

1. The message is padded to 512 bits
2. The message is split in two parts of 256 bits
3. Each F function represents the whole function F256 described in the previous step (aimed at a 256 bits message). {k1…k4} represent here four sets of 4 AES-128 keys used in the previous steps.

Let's call the function described above F512.

The AES encryption scheme should be used in ECB mode, with no padding.

[4] (#bytes/16)^2 encryptions

A function F_{1024} can be used by combining four functions F_{512}; a function F_{2048} can be used by combining four functions F_{1024}, etc. Different keys have to be used for every AES-128 encryption.

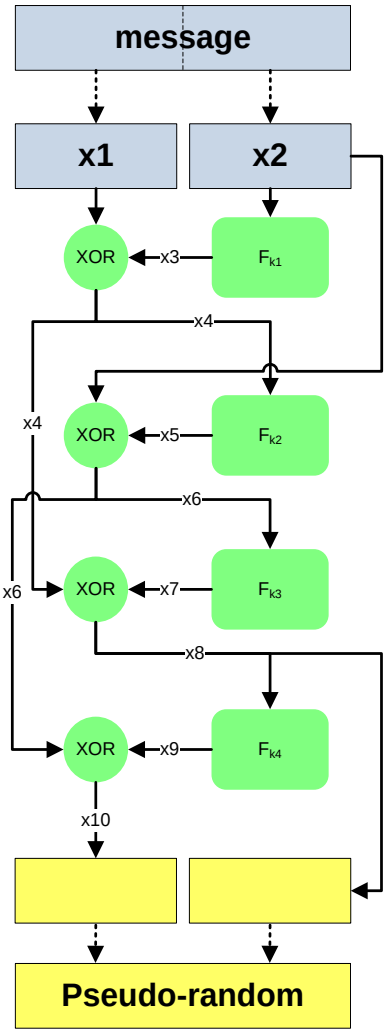


Figure 24: Symmetric encryption

4.1.2 Asymmetric encryption

PKCS#1 v2.1 (RSAES-OAEP) is a proven and easy to implement solution.

The problematic part in using RSAES-OAEP in this context is that RSA-OAEP is probabilistic: the message is scrambled with pseudo-random padding derived from a random seed before being RSA-encrypted. This would obviously void the reproducibility of the solution; a solution is needed where the padding is identical for each identical message5. Two solutions are possible:

1. As a (different) pseudo-random padding is needed for each different message, a function of the temporary unique root identifier could be used as the encryption

[5] This is called "deterministic public key encryption scheme" (see [6] for references)

seed, which will be different for (almost[6]) every root identifier. For example, the following data can be used as a seed:

- the 512 first bits of the reversible ID; this is a real pseudo-random number;
- a 512 bits hash of the temporary unique root identifier, encrypted with the above symmetric algorithm; this solution has the advantage of reducing the maximum number of keys (16);
- a MAC of temporary unique root identifier; this solution has the advantage of requiring only one key;

2. Firstly the temporary unique root identifier can be extended with a MAC of itself, and then the result can be encrypted using a standard construction from deterministic public key encryption. Attaching the MAC essentially makes the encrypted message "unpredictable". Concretely, m=[temporary unique root identifier||MAC(GURI,SK)] can be taken for some symmetric secret key stored by the PEPS, and then encrypt the message as follows. First the encryption seed is calculated as H(PK||m), and then the RSAES-OAEP encryption of m is calculated using the previous seed. This has the disadvantage of resulting in longer identifiers, but it is a more standard construction (see [7] for an in-depth analysis); the padding is, in this case, usually a hash of the public key concatenated with the message: H(PK||m)

The key length must be at least 130 bytes[7] longer than the message to encrypt. By using a 4096 bits key, a root identifier of 382 bytes can be encrypted (potentially even a bit more if it is compressed somehow before encryption).

Please note that using 2048 bits keys – which is considered nowadays as a minimal – would lead to a 342 bytes identifier, which is much longer than the allowed length of a STORK identifier. Therefore, this solution cannot be used inside STORK.

4.1.3 MAC

A proben secure MAC algorithm should be used, e.g. HMAC with a good cryptographic hash function, against which there is no known attack that improves on a birthday attack.

Furthermore, adding the public key used for encryption into the MAC [8] would be a good solution in order to have a different padding when the key pair is migrated to a stronger one: MAC(SK, PK||m).

4.1.4 Hash

SHA-256 or SHA-512, depending on the desired length, are good candidates for the usages envisioned in this document.

4.2 Version Control (SPs)

The version control verifies the software and configuration versions of its PEPS / V-IDP and publishes the results to the national PEPS/V-IDP service providers.

The *version control* process is carried out under the standard facility to the launch of processes periodically, like Task Manager in Windows or cron under Unix. If any changes are found in its PEPS' configurations, an alert is sent to the administrator(s), in order to inform him that some compatibility tests should be performed.

[6] As it is only a seed for an encryption, there is no problem if it there are some collisions

[7] The exact value is 2 * (*hLen* + 1) where *hLen* is the length (in bytes) of the hash function used in the algorithm

In any case the version control files are generated, which allow on one hand, national Service providers to be informed of relevant changes and adapt their country selector, and on the other hand, allows the colleagues to inform their administrators to execute relevant tests.

4.2.1 Sequence diagram VCS

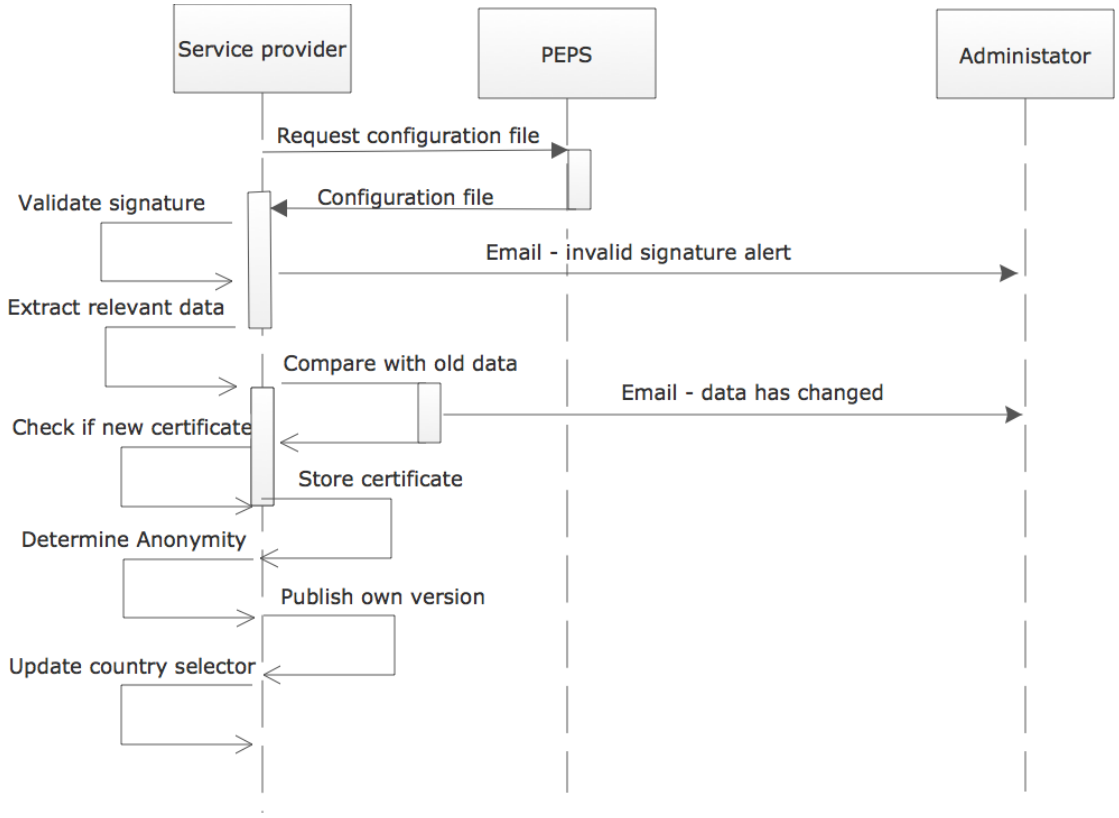
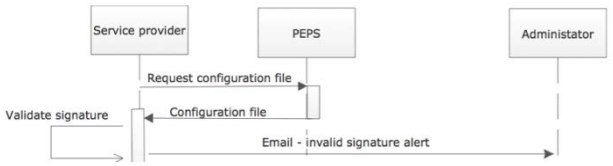
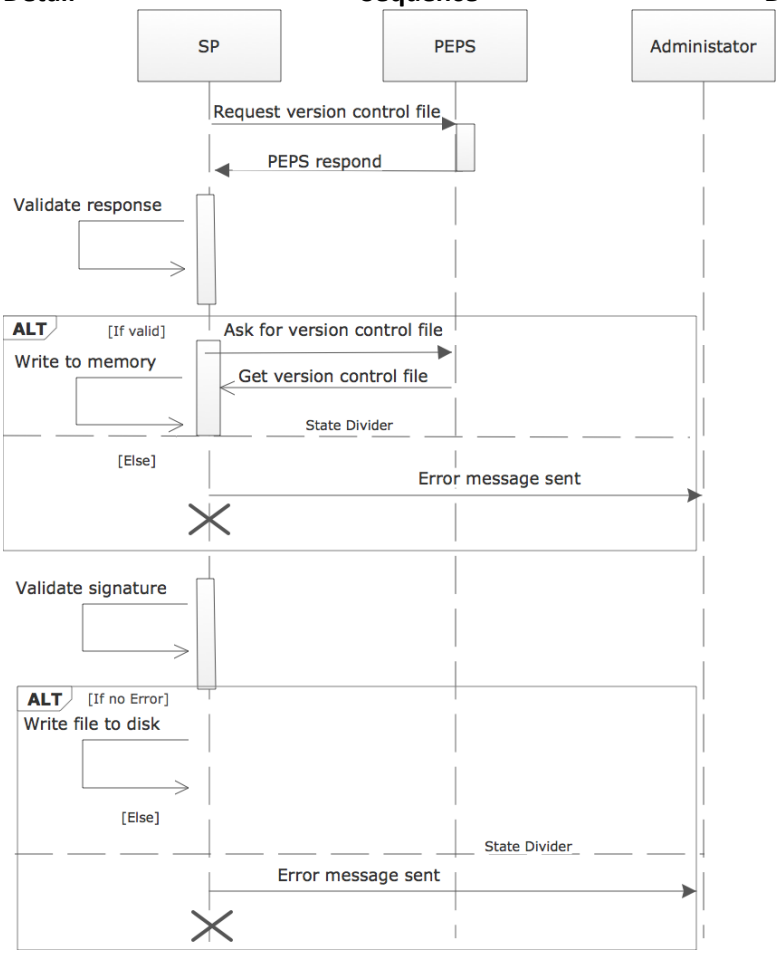


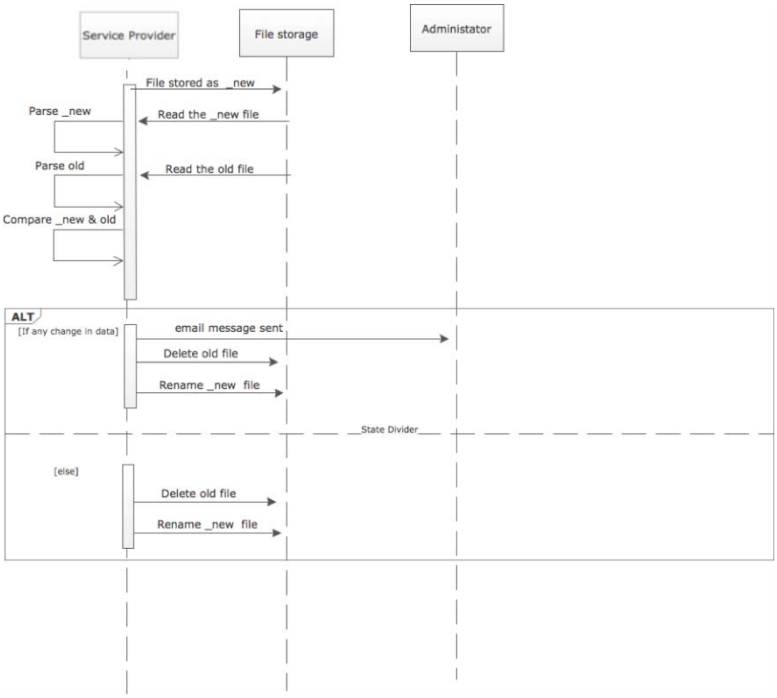
Figure 25: Sequence diagram Version Control, in SP

4.2.2 Description VCS

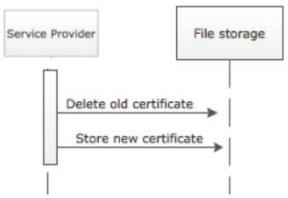
Message sequences (interactions)		***Description***
1	VCP-ACT-1	**Description** The SP sends scheduled request for configuration to a S-PEPS/V-IDP. The configuration is stored in Version Control File (VCF) stored at https://xxx.xxx.xx/PEPS/SPs-XX-info.xml. The version control file is signed by the S-PEPS/V-IDP (SPs-XX-info.xml were XX is the country code of the S-PEPS/V-IDP. For example for Spain SPs-ES-info.xml). If version control file is in place, the PEPS/V-IDP responds with the version control file (SPs-xx-info.xml) file which contains configuration settings and all needed certificates, also the configuration settings and certificate for this S-PEPS/V-IDP to act as an anonymity node and all the certificates and settings for its subordinate non-authoritative nodes. If the version control file is not in place the request for configuration results in error. The SP validates the signature and format of the version control file. In

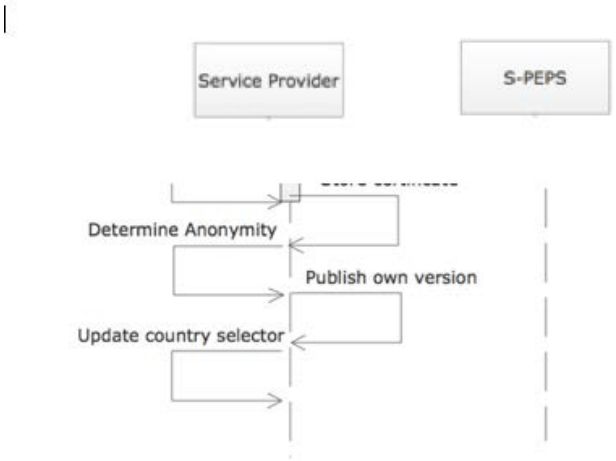
Message sequences (interactions)		*Description*
		case the signature and format are invalid the SP sends error message to the SP Administrator and stops the process.
		Sequence Diagram 
		Detail Sequence Diagram 
		External Actors • SP job scheduler
		Action 1 **Function Validate URI to PEPS/V-IDP Version control file** **Data** INPUT URI to PEPS/V-IDP Version control file OUTPUT

Message sequences (interactions)		*Description*
		Success/error (VCP_ERR_1 / VCP_ERR_2) CONFIG FILES NEEDED No config file needed _____ **Action 2** **Function Validate PEPS Version control file** Request version control file If file is available Get file Validate signature Save file to memory Else Return error Send error to Administrator Write to error log **Data** INPUT Version control file path (ex: https://xxx.xxx.xx/PEPS/SPs-XX-info.xml.) OUTPUT Success/error (VCP_ERR_3 / VCP_ERR_4) CONFIG FILES NEEDED No config file needed
2	VCP-ACT-2	**Description** If this is the first time a version control file is fetched from the PEPS/V-IDP and the signature and format are correct and trusted, the SP stores the Version control file and the process continues directly to VCP-ACT-4. If this is not the first time the version control file is fetched and the signature and format of the file is correct and trusted, the SP stores the version control file adding _new to the file name. The SP reads the previous version control file and the _new version control file; extracts relevant data from the files and compares the new data to the data in the previous version control file. If there is no change in the data, the SP deletes the previous version control file and removes "_new" from the name of the new version control file (the new file is renamed). If there is any change in the data, change information are added to

Message sequences (interactions)		*Description*
		email that will be sent to the SP Administrator. If the change in data includes no new certificates, the process continues with VCP-ACT-4.
		Sequence Diagram 
		Detail Sequence Diagram 
		External Actors • No external actors
		Action 1 **Function check for previous version control file for PEPS** If no previous file is available Go to step VCP-ACT-4 Else Store the version control file as SPs-XX-info.xml_new **Data** INPUT Previous version control file

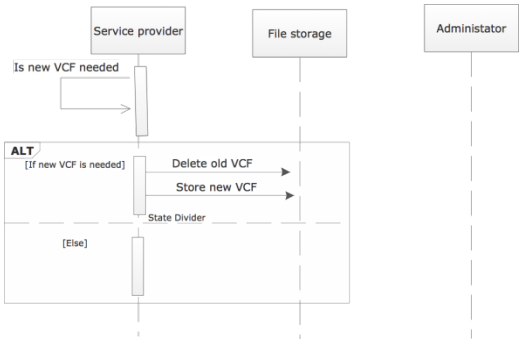
Message sequences (interactions)		*Description*
		OUTPUT SPs-XX-info.xml_new / **Go to step VCP-ACT-4** CONFIG FILES NEEDED No config files needed _____ **Action 2** **Compare data** Store the version control file as SPs-XX-info.xml_new Parse the file SPs-XX-info.xml_new into variables Read the old version control file SPs-XX-info.xml Parse the file SPs-XX-info.xml into variables Compare all data from SPs-XX-info.xml_new with the data in SPs-XX-info.xml If any change in data Add information about changes to email that will be sent to administrator Delete SPs-XX-info.xml Rename SPs-XX-info.xml_new to SPs-XX-info.xml If no new certificates Go to step VCP-ACT-4 Else Retrieve the name of old certificate file from the old version control file Delete SPs-XX-info.xml (the old version control file) Rename SPs-XX-info.xml_new to SPs-XX-info.xml Go to step VCP-ACT-3 **Data** INPUT Previous version control files OUTPUT New version of PEPS version control file CONFIG FILES NEEDED No config files needed

Message sequences (interactions)		***Description***
3	VCP-ACT-3	**Description** If the change in data includes any new certificate, the SP checks and removes any out dated certificate before it stores all new certificates.
		Sequence Diagram 
		External Actors • No external actor
		Action **Function check for out-dated certificates** Look for old certificate If old certificate is found Delete old certificate Store new certificate **Data** INPUT Name of the old certificate file from old version control file New Certificates retrieved in previous step OUTPUT Success / Error (VCP_ERR_6, VCP_ERR_7) CONFIG FILES NEEDED No config files needed
4	VCP-ACT-4	**Description** The SP determines if it has the anonymity layer software installed and configured. If so, it determines the version of the software and configuration files.

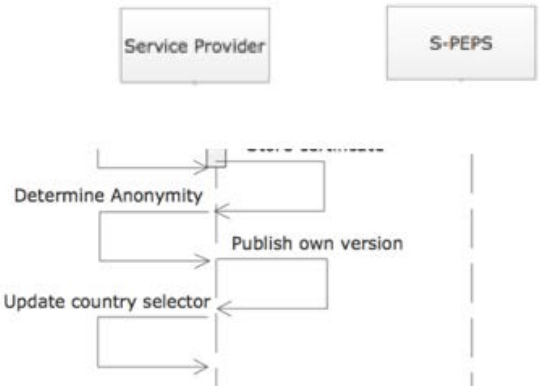
Message sequences (interactions)		*Description*
		Sequence Diagram 

		Action 1
		Determine if Anonymity is installed
		If Anonymity is installed
		Determine version of the Anonymity software
		Determine version of configuration files
		Data
		INPUT
		Software installation status
		OUTPUT
		True (and Anonymity software version) / False
		CONFIG FILES NEEDED
		No config files needed
		_____
		Action 2
		Function update PEPS Anonymity Data
		if SP is not anonymity node:
		if SP was an anonymity node:
		Mark the node for deletion.
		Mark all of this node's subordinate non-authoritative nodes for deletion.
		end

Message sequences (interactions)	*Description*
	Insert or update node certificate on the certificate store Insert or update node settings on the database for each subordinate non-authoritative node: Insert or update node certificate on the certificate store Insert or update node settings on the database for each subordinate non-authoritative node that is on the database but is no longer on the Control Version File: Delete node certificate from the certificate store Delete node settings from the database **Data** INPUT SP node settings SP node certificate List of subordinate non-authoritative nodes with: Node certificate Non-authoritative node settings Status OUTPUT CONFIG FILES NEEDED Anonymity config file _____ **Action 3** **Function clean node database** for each node marked for deletion whose exclusion period has expired: Delete node certificate from the certificate store Delete node settings from the database **Data** INPUT Database

Message sequences (interactions)		*Description*
		OUTPUT CONFIG FILES NEEDED Anonymity config file _____ **Action 4** **Function update own control version file** if own anonymity certificate or settings change, own Control Version File must be updated. if subordinate non-authoritative node list changes (additions, deletions, setting or certificate changes), own Control Version File must be updated. **Data** INPUT New settings / certificates Database OUTPUT New version control file CONFIG FILES NEEDED Anonymity config file
5	VCP-ACT-5	**Description** The SP determines if new version control file is needed based on if there has been a change in configuration files which are referred in the Version Control.
		Sequence Diagram 
		External Actors No external actors
		Action 1

Message sequences (interactions)		*Description*
		Store Current Version Control File values Read current Version Control File Parse current Version Control File Put values into variables **Data** INPUT Own version control file from local storage OUTPUT Variables containing the Version Control File values CONFIG FILES NEEDED Own Version Control File **Action2** **Determine if certificates have changed** Read newest certificate files from certificate storage Compare new certificates with certificates in VCF If own certificates have changed Update corresponding variable value **Data** INPUT Own certificates stored in local certificate storage OUTPUT Updated value's for variables containing VCF certificates CONFIG FILES NEEDED No config files needed **Action 3** **Determine if configuration files have changed** Read configuration files and compare if data has changed If data in configuration files has changed Update corresponding variable value **Data** INPUT

Message sequences (interactions)		*Description*
		Configuration files containing info for the VCF OUTPUT Updated variable values CONFIG FILES NEEDED Configuration files containing VCF info **Action 4** **Build new Version Control file** Use stored variables to build new VCF Delete old Version control file Store new Version control file **Data** INPUT Stored variables OUTPUT New version control file CONFIG FILES NEEDED No config files needed
6	VCP-ACT-6	**Description** Based on the new VCF file, country selector is updated.
		Sequence Diagram 
		External Actors No external actors

Message sequences (interactions)		***Description***
		Action 1 **Update country selector** Read current Version Control File Parse country selector values into variables Read the country selector Update the country selector Remove old country selector Save new Country selector **Data** INPUT Own version control file from local storage OUTPUT New country selector CONFIG FILES NEEDED Own Version Control File
7	VCP-ACT-7	**Description** The SP checks if any mail event is existed, the SP prepares a table formatted email. Then the SP sends this email to Administrator.
		Sequence Diagram
		External Actors No external actor
		Action **Function check for email event** If mail event is existed Prepare table formatted email Send email to administrator **Data** INPUT Mail events(certificate changes, VCF data changes, any error..) OUTPUT Success / Error (VCP_ERR_6, VCP_ERR_7) CONFIG FILES NEEDED

Table 18 –Description Version Control in SP

4.3 Personal Data comparison (for re-authentication)

4.3.1 Introduction to the problem

Sometimes people can be authenticated in foreign countries with their home-country eID. This would be the ideal situation: the university where someone studies should store his data under his foreign citizen number.

For such cases, the C-PEPS, when requesting domain-specific attributes from an A-PEPS, includes the citizen's eIdentifier in the request, to allow the national Attribute Providers to retrieve the citizen's data from the database. In order to allow such attribute providers to reverse look-up these data based on his real data, instead of his citizen number, his givenName, surname and dateOfBirth are also included in the request. The AP should apply complementary mechanisms to verify that this person is the one he/she says.

4.3.1.1 Re-authentication

However, in many systems user's data are not stored under any citizen number; instead a student-number, employee-number or any other <role>-number is used, especially in countries where citizen numbers have not culture, like The Netherlands, Greece and Germany. Such <role> number's were frequently also used as a user-id, and corresponding accounts were usually protected with passwords.

In such cases, when attending a request from a SP, the AP solving this request will not be able to retrieve any data related to the foreign citizen number, so it will have to authenticate the user again with the authentication method of the AP, typically username / password schemes.

Such re-authentication not only applies to foreign citizens; it may be required also for national ones.

4.3.1.2 Consequences for QAA and AQAA

If re-authentication has taken place, the quality of the second authentication limits the QAA to be assigned to these attributes: that can never be superior to the quality of the used credential. So in the typical case of username / password, if complemented with normal measures for serious use, these credentials are QAA 2.

This new QAA would be a maximum for the attribute AQAA, on which other criteria for attribute issuing, as defined by the STORK 2.0 Legal and Trust Analysis, should be applied.

4.3.2 Double identities – two persons?

The eIdentifier is guaranteed, by the government which issued it, to be traceable to one and only one person. If the Attribute provider did not use the citizen's eIdentifier to retrieve his data, the other mechanism may include the possibility that 2 different persons collaborate to assign one person's attributes to the other person.

The AP has no way to observe if one or two persons are present at the terminal and how they interact with their systems and the central STORK nodes.

4.3.3 STORK "solution"

In the paper world the receptor of the data (the university degree or whatsoever) normally verifies that the person to whom the document was issued is the same as the one he has in front, at least that the data included in the document are the same as the data of the person's identification. Such document would only be accepted if name and surname are the same, or at least *sufficiently similar*.

4.3.3.1 Conceptual solution

What is done in the paper world may also be done in the electronic world; in some points better, in other worse. The basic idea of the solution is that, in case re-authentication has taken place, the AP returns, together with the requested domain-specific attributes, some personal data of the person to whom they were issued. The receptor may compare this set of personal data with the ones received from his national authority.

This set includes givenName and surname, very common in the paper world, and improves the possibility of fraud by adding the dateOfBirth.

4.3.3.2 Names comparison

Where date comparison is simple, names comparison is not. Comparing John Smith with John Smith seems straightforward, but there are several problems.

4.3.3.2.1 National cultures

In the first place, an official givenName is often substituted by the commonly used other name. Well known in the English speaking world is that Robert is often called Bob, less known is the same in the Dutch speaking world with Johannes being Jan, Hans or John, or in the Spanish world José vs. Pepe, and Greek Stylianos vs. Stelios.

For surnames also limitations exist, especially (though not limited to) people getting married in many countries may assume the surname of their partner.

Such national cultural substitutions cannot be qualified as similar by any machine.

4.3.3.2.2 Accents and diacritics

All keyboards in Europe allow typing the basic letters A-Z. On many of those keyboards there are facilities to type special characters, accented vowels (á, è, ô, etc.). But few of them support stranger special characters like ñ, ß, ð.

When a citizen has relations with foreign institutions, i.e. goes to some foreign office and is registered there, the employees at this foreign institution will do their best to copy this citizen's data, sometimes applying evident transposition rules, eliminating accents and diacritics; sometimes asking the citizen what to write instead of such weird letters. In this last case the citizen would indicate that in DE and AT "ß" is equivalent to "ss".

4.3.3.2.3 Transposition tables

In STORK 2.0 a pragmatic approach is presented, which is not universal, but gives an indication of similarity of names; based on the similarity the receiver of the data may decide to accept or to reject the associated business data.

The solution would use transposition tables in order to establish the most probable substitution of the special letter to the standard 26 letter set, thus "stripping" the accents and diacritics from those letters, and substituting ð by d.

Applying this mechanism to both data sets, "comparable" results can be found.

This module would indicate the grade of similarity between the sets.

4.3.3.2.4 Examples

If the examples in the first column would be compared with the second column, the similarity would be indicated in the third column.

Name in one set	*Name in another set*	*Similarity*	*Comment*
Gómez	Gomez	100%	
Piñuela	Pinuela	100%	
Müller	Mueller	85%	In German speaking world, ü and ue are equal. For this example it has been assumed that the transposition table indicates that ü should be translated with u. The idea behind this is that these tables can't take into account that Spanish equivalent is different from German equivalent
Žužek	Zuzek	100%	
Smith	Smit	80%	This is probably a "false positive"; Smith is English, Smit is Dutch
Alenka	Alicia	33%	
Sigurður	Siggi	38%	This is probably a "false negative"
Stylianos	Stelios	50-66%	

Table 19 –Similarity examples

4.3.4 Alternative solutions

In the first place the best way to solve this issue is avoiding it by storing user's data under his eIdentifier. However, although this solution is being used since STORK got live, in 2010, we cannot expect all "legacy data", which is some 99,9% of the domain-specific attributes in several countries, and 99,9% of domain-specific attributes of foreigners in all countries.

Any other solution would not solve the issues mentioned in section 4.3.3.2.1, and even then, this would be limited to countries with a persistent eIdentifier (not GR).

4.3.4.1 Language Sensitive Transposition tables

An alternative for the simple transposition table would be to include the language to use with transposition. This way any transposition would have 2 results: the simple one and the language-sensitive one. When comparing one name with another name, the best result of 4 comparisons should be taken into account.

4.3.4.2 Multiple Transposition tables

A transposition table with multiple translatable values (ü to u as well as ue), but without limitation of the number of possible transpositions, could increase the probability of reaching 100% similarity.

This sounds very good, but it also means that the number of *false positives* will increase.

4.3.5 Comparison of the chosen solution with other solutions

4.3.5.1 False positives

A false positive is in this context the acceptance of data as belonging to one person when in reality the data belong to two persons. Of course this does not depend on the module for determining the similarity itself, but on the requirements of the one who accepts them or not.

But it is this module which will implement the guidelines for similarity, and report just a number, on which the receptor of the data will base his decision, mostly in an automated way, to accept the domain-specific attributes.

Thus false positives may increase the possibility of fraud.

4.3.5.2 False negatives

A false negative is in this context the rejection of data as belonging to one person when in reality they do. Of course this does not depend on the module for determining the similarity itself, it also depends on the requirements of the one who accepts them or not.

But this module will implement the guidelines for similarity, and report just a number, on which the receptor of the data will base his decision, mostly in an automated way, to accept the domain-specific attributes.

False negatives will usually be rejected by automated processes in order to be treated by human beings, therefore false negatives just cause more work.

4.3.5.3 Complexity

The proposed solution is, within the scope of STORK, the simplest solution.

The “Multiple transposition tables” solution seems not much more difficult, but has a disadvantage of major amount of false positives.

The “Language Sensitive Transposition” is definitely more complex than the proposed solution.

4.3.6 Software design and package usage examples

The software has been implemented as Java package. The basic aim of the package is to estimate how similar are two names, for example when assessing whether the two names belong to the same person. The similarity is a value between 0 and 1, calculated based on the Jaro–Winkler distance[8]. The names are transformed by transliteration rules from UTF-8 to latin, according to the Machine readable travel documents specification published by

[8] See Wiki page http://en.wikipedia.org/wiki/Jaro-Winkler_distance for details.

International Civil Aviation Organization[9], before similarity comparision. Greek transliteration is based on ELOT 743 standard[10].

Identical names have a similarity value of 1.0.

The package supports latin, cyrilic and greek alphabet. Serbian, Macedonian, Ukrainian and Bulgarian cyrilic exceptions are supported.

4.3.6.1 Usage examples

Below are presented simple usage examples. Class NameTransform can be used to transform the names to the transliterated strings. The transform could be set to a specific country variation, as in example to Belorussia's.

The JaroWinklerSimilarity calculates the similarity between two strings according to their distance. Country specific similarity can be instantiated by creating a coresponding country specific JaroWinklerSimilarity object. The examples below are provided for various names in different languages together with note on transliteration or comaprission results in the code comments.

```
import eu.stork.namesimilarity.JaroWinklerSimilarity;
import eu.stork.namesimilarity.NameTransform;

public class Example {
 public static void main(String[] args) throws Exception{

 // Use default transliteration
 NameTransform nt = new NameTransform();
 // prints "Alenka Zuzek"
 System.out.println(nt.transform("Alenka Žužek"));

 // Invoke Belorussia transliteration variation
 NameTransform ntBY = new NameTransform("BY");
 // prints "Siarhei Aliakseevich Rutenka"
 System.out.println(ntBY.transform("Сяргей Аляксеевіч Рутэнка"));

 // Check default similarity
 JaroWinklerSimilarity jwsDefaults = new JaroWinklerSimilarity();
 // prints true
 System.out.println(jwsDefaults.isSimilar("Alenka Žužek","Alenka Zuzek"));
 // prints true
 System.out.println(jwsDefaults.isSimilar("Владимир Јовановиќ","Vladimir Jovanovik"));
 // MK has a specif fallbacks, invoke specific country transliteration
 JaroWinklerSimilarity jwsMK = new JaroWinklerSimilarity("MK");
 System.out.println(jwsMK.isSimilar("Владимир Јовановиќ","Vladimir Jovanovik"));
 // prints true
 System.out.println(jwsMK.isSimilar("Владимир Јовановиќ","Vladimir Jovanovikj"));

 // Belorussian example
```

[9] See Civilian Aviation Organization transliteration specification document for details, http://www.icao.int/publications/Documents/9303_p3_v1_cons_en.pdf

[10] See ELOT 743 transliteration standard specification for details, http://sete.gr/files/Media/Egkyklioi/040707Latin-Greek.pdf

```
    JaroWinklerSimilarity jwsBY = new JaroWinklerSimilarity("BY");
    // prints false, country code can be specified per call (r in name
    // Сяргей is usually transliterated to g but in Belorussia's case
    // to h)
    System.out.println(jwsBY.isSimilar("Сяргей Аляксеевіч Рутэнка","Siargei Aliakseevich
Rutenka"));

    // Greek examples
    JaroWinklerSimilarity jwsGR = new JaroWinklerSimilarity("GR");
    // All prints are true
    System.out.println(jwsGR.isSimilar("Αριστοτέλης Ωνάσης","Aristotelis Onasis"));
    System.out.println(jwsGR.isSimilar("Σταύρος Σπύρος Νιάρχος","Stavros Spyros Niarchos"));
    System.out.println(jwsGR.isSimilar("Ευγενία Λιβανού","Evgenia Livanou"));
    System.out.println(jwsGR.isSimilar("Γεώργιος Παπαδόπουλος","Georgios Papadopoulos"));
    System.out.println(jwsGR.isSimilar("Δημήτριος Ιωαννίδης","Dimitrios Ioannidis"));
    System.out.println(jwsGR.isSimilar("Σπυρίδων Μαρκεζίνης","Spyridon Markezinis"));
    System.out.println(jwsGR.isSimilar("Νικόλαος Γεωργαλής","Nikolaos Georgalis"));
    System.out.println(jwsGR.isSimilar("Παναγιώτης Γιαννάκης","Panagiotis Giannakis"));
    System.out.println(jwsGR.isSimilar("Γιώργος Βασιλακόπουλος","Giorgos Vasilakopoulos"));
    System.out.println(jwsGR.isSimilar("Πλάτων","Platon"));
    System.out.println(jwsGR.isSimilar("Άριστοτέλης","Aristotelis"));
    System.out.println(jwsGR.isSimilar("Sōkrátis","Σωκράτης"));
    }
  }
```

4.3.7 Conclusion

In case citizen's data are not stored under his foreign eIdentifier, probably a re-authentication should take place. Such re-authentications opens the door to authentication of two different persons, allowing the second person the first one to use his domain-specific attributes in his own benefit. For such cases, the receptor of the data should verify that the domain-specific attributes issued for one person correspond to the principal.

In the majority of givenNames and surnames the correspondence is 100%, except for typing errors. For a minority of European citizens (estimated between 10 and 20%) a tool will try to map special characters to "normal" characters, using a simple transposition table like already mentioned Civilian Aviation Organization transliteration specification in Section 4.3.6.

The proposed tool offers a transposition of special letters to normal letters, to be used by receptors of domain-specific attributes.

4.4 Browser Temporary Storage Management

The improvement of user-friendliness ("Attribute Aggregation") is achieved using the browser's temporary storage, often also referred to as "cookies".

4.4.1 Introduction to the problem

The scope of using the "cookies" is to improve the user friendliness when exploiting the STORK 2.0 infrastructure for authentication and attribute retrieval. The number of interactions between the STORK 2.0 infrastructure and the user in which the user is asked to select the country and/or the authority for a given attribute could occur very often and thus would require a significant time spent by the user in selecting and re-selecting several times the same country/authority.

To avoid this inconvenience it is necessary that the location information (e.g. of a country or of an authority) obtained in an authentication session completed successfully are memorized in the temporary storage of the user's browser so that it can be re-used in one or more successive authentication sessions.

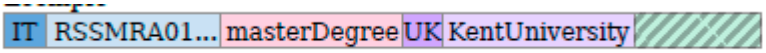
The location information is contained in a structure called Attribute Object Identifier (AOI) – further referred also as token - which will be saved in the user's browser. The Attribute Object Identifier contains several data items:

- C-PEPS identifier (CC): is a code uniquely identifying the citizen's PEPS in the STORK 2.0 infrastructure
- Citizen's electronic identifier (CEID): is a local identifier which allows C-PEPS to uniquely identify a citizen. The subject identifier might be different from the eIdentifier attribute defined in STORK 2.0.
- Attribute identifier (Adata): is a field which allows identifying an attribute. It could a simple attribute name or a more complex string identifying an attribute.
- A-PEPS identifier (AC): is a code uniquely identifying the A-PEPS in the STORK 2.0
- Attribute Provider Identifier (AP): contains a string uniquely identifying on A-PEPS the Attribute Authority that can return the value for an attribute identifier through the above "Attribute Identifier"
- Additional information (AVALUEPTR): is a field allowing the Attribute Authority to reference the attribute, so that it can be retrieved faster in successive interactions. The value is not fixed, each Attribute Authority is free to choose/implement the value of this field, which could be for example a pointer towards a central storage of public data.

CC	CEID	Adata	AC	AP	AVALUEPTR

Table 20: Basic AOI format

An example AOI is given below, where "IT" is the C-PEPS identifier, "RSSMRA01" is the subject identifier, "masterDegree" is an attribute identifier, "UK" is the A-PEPS identifier, "Kent University" is the Attribute Provider Identifer and the Additional Information is left empty :



Two phases involving AOI management are distinguished:

Phase 1: AOI creation. In this phase the citizen selects the C-PEPS where he will be authenticated, and selects also the A-PEPSes from where the (foreign) attributes will be retrieved. The C-PEPS creates the AOI, the AOI is stored in the user's browser. This phase will be further detailed in Section 4.4.3 (Generation of the token).

Phase 2: AOI usage. In this phase, the AOI previously created and stored in the user's browser in phase 1 is processed by the C-PEPS, by A-PEPSes and (depending on AP) possibly also by APes to retrieve attribute faster and in a transparent manner. This phase will be further detailed in Section 4.4.5 (Interpretation of the token).

The exploitation of the AOIs in STORK 2.0 is shown (at a high level view) in *Figure 26***.**

Note: In STORK 2.0, AOIs are created/processed by C-PEPS, A-PEPS and (possibly) by AP.

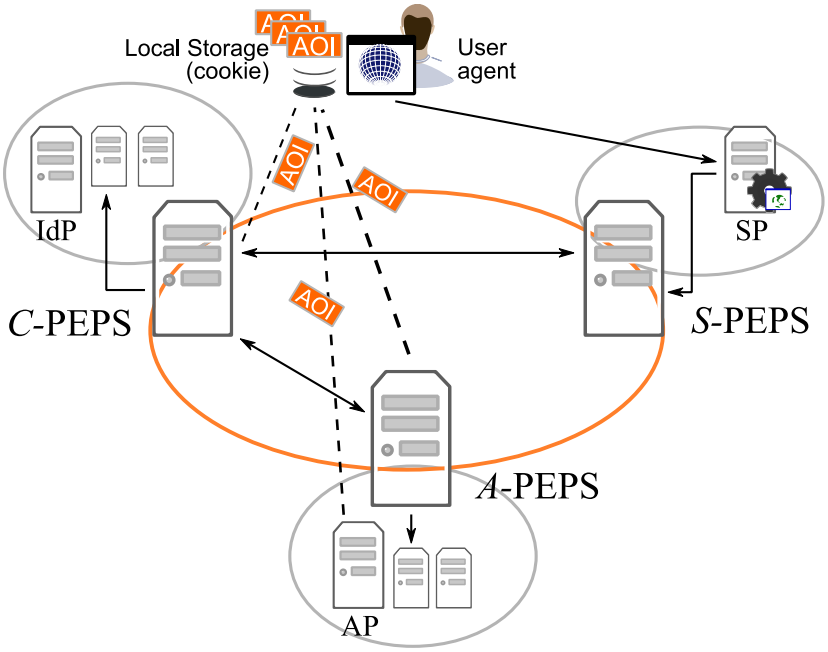


Figure 26. AOI exploitation in STORK 2.0

4.4.2 Integrity protection of the token

To guarantee the integrity and authentication of the AOI stored in the cookies, STORK 2.0 proposes to associate to each of them a Message Authentication Code (MAC). In the implementation it is proposed to use HMAC SHA 256 for the calculation of the MAC. Moreover, to protect from attacks aimed to extend the original message by appending new data, a field containing the length of the entire AOI (including the MAC) is added to the AOI format.

To guarantee confidentiality when transferring the AOI, it is possible to associate the attribute "Secure" to the AOI cookies so that they are transported only over HTTPS connections.

Thus, the format of the secured AOI is shown below:

CC	CEID	Adata	L	AC	AP	AVALUEPTR	MAC(AOI, $K_{C\text{-}PEPS}$)

Name (key) Value

Figure 27. Secure AOI format

Where L is the length in bytes of the AOI (name + value) and the value of the MAC(AOI, $K_{C\text{-}PEPS}$) (calculated with a symmetric key $K_{C\text{-}PEPS}$) is encoded in Base 64.

4.4.3 Generation of the token

In STORK 2.0 the AOI will be implemented in form of cookies to be stored in the user's browser.

The cookies can be read and processed only by the application that generated them. Thus, in the first place it should be decided which component in STORK 2.0 infrastructure is in charge with generating the AOI (or the token).

In brief, the main approach for AOI generation/reading in STORK 2.0 is described below:

1. C-PEPS stores/retrieves in its cookie the AP identification of national attributes;

2. C-PEPS stores/retrieves in its cookie the country-code of foreign attributes;
3. A-PEPS stores/ retrieves in its cookie the AP-identification of attributes to be retrieved in its country;
4. A-PEPS may store/retrieve in its cookie the country codes for foreign attributes, if the user had nested countries;
5. AP may store/retrieve user's identification from the cookie, if different from the (foreign) eIdentifier;

The SAML token in all requests from C-PEPS and A-PEPS will include the citizen's eIdentifier.

The generation of the AOI is shown in ***Figure 28***.

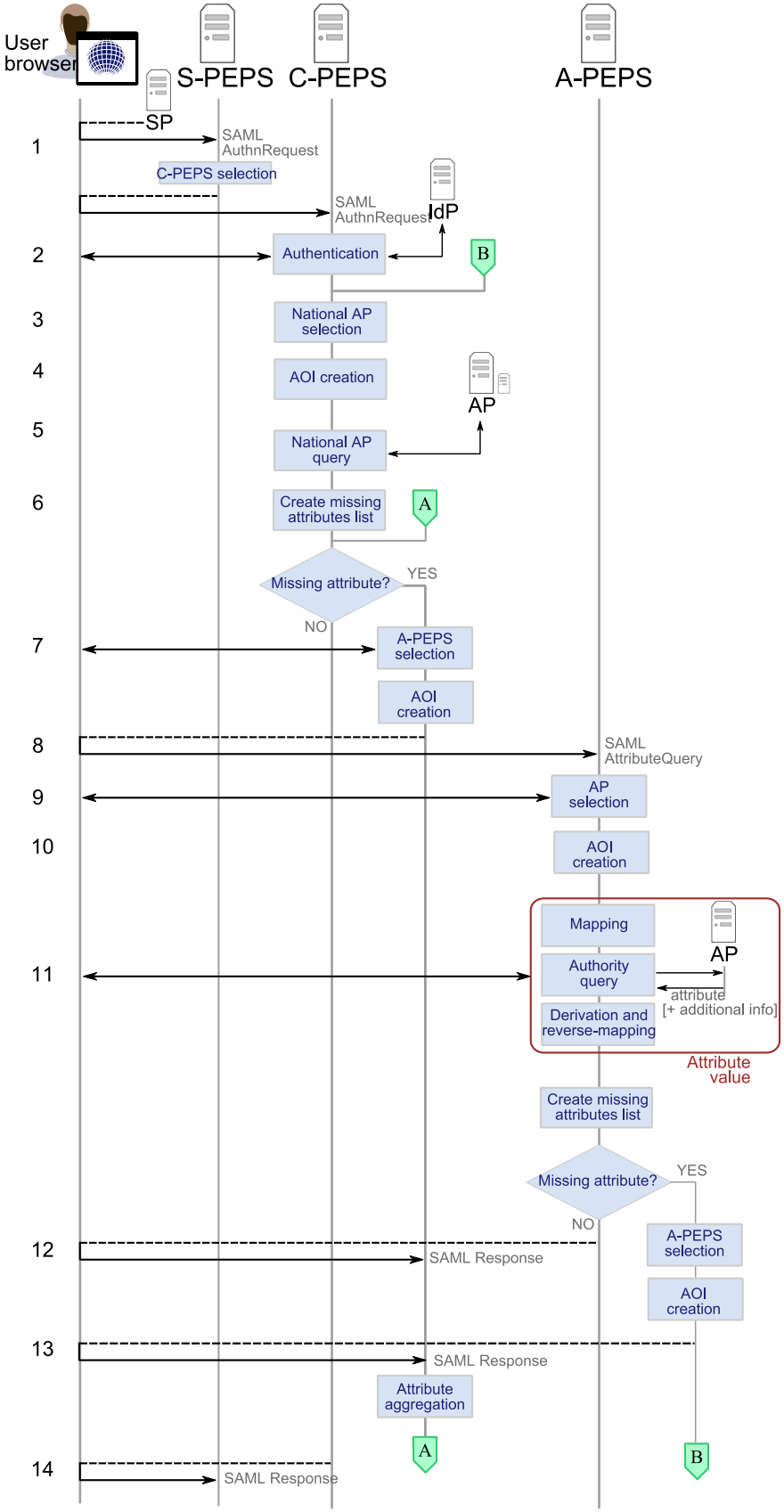


Figure 28. AOI creation

After authentication (Step 2), the C-PEPS creates a list of national attribute providers for the attributes requested (Step 3 in***Figure 29***). On selection, the C-PEPS constructs the AOI (Step 4)

which is sent to the user's browser upon redirect to national APs and retrieval of national attributes (Step 5).

Next, the C-PEPS creates a list containing attribute identifiers whose values are to be retrieved from foreign countries (Step 6 in *Figure 28*). For each missing attribute, the user can specify the (foreign) countries where the attributes can be obtained (Step 7) and consequently towards which A-PEPS the attribute query will be sent by the C-PEPS (Step 8).

After A-PEPS selection, the user is redirected on A-PEPS where he will select the national APs (Step 9). For each national attribute, the corresponding AOI is created and sent to user's browser on redirect to the AP (Step 11). The national attributes are retrieved by the A-PEPS using Member State specific procedure. If other additional attributes need to be retrieved from foreign countries, the A-PEPS creates a list containing attribute identifiers whose values are to be retrieved from foreign countries. For each missing attribute, the user can specify the (foreign) countries where the attributes can be obtained, as it was performed on C-PEPS. After the selection of A-PEPS, the AOI containing the foreign country identifier is created by A-PEPS as above and is sent to user's browser upon redirecting to the foreign A-PEPS.

Note: To allow selection of foreign attributes provided by foreign APs on C-PEPS, a variation of the previous protocol might be done: the A-PEPS transfers the names of the foreign APs towards C-PEPS inside a SAML message. In particular, the attribute values together with the fields of the AOI containing the AP and (if available) the AVALUEPTR (named also "partial AOI") might be returned by the C-PEPS to A-PEPS. The C-PEPS constructs the complete AOI and it sends it to the user agent (on redirect or when sending out the response).

This case is particularly useful in those cases in which we have one attribute that can be retrieved from more than one AP (in the same country) or one attribute that can be retrieved from several countries (as the user might skip some A-PEPS interactions).

4.4.4 Format of the AOI stored in cookies

The cookies are identified uniquely by a name, and for this reason it is necessary to split the AOI in two parts: one is the name (or the key) and the other one is the value.

In our approach the name of the cookie (or the key) is composed of a triple (C-PEPS identifier, Subject Identifier, Attribute identifier), while the rest of the other fields (i.e. A-PEPS identifier, Attribute Provider Identifier, Additional information) are part of the value of the cookie.

When implementing the AOI in the cookie, the format of the actual AOI format is completed with a separator, named "S" in *Figure 29* :

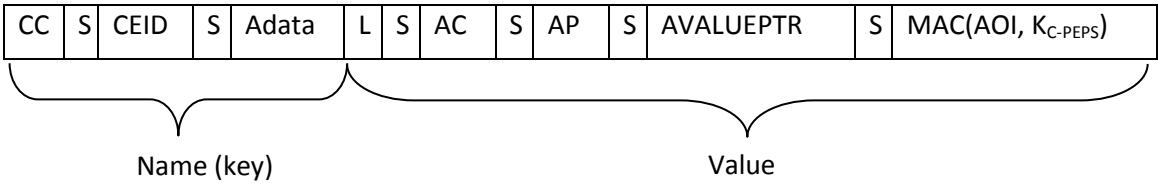


Figure 29. Structure of the AOI stored in the cookie

An indication of the values and lengths of the fields in the AOI are given below:

Name	***Meaning***	***Name Format***	***Length (bytes)***	***Example***
CC	Citizen's country (C-PEPS)	alpha-2 string ISO3166-1 (uppercase)	2	IT

S	Separator	Note: it must be a character that does not appear in any citizen's national identifier of any country, in Adata, or in AVALUEPTR.	1	"-"
CEID	Citizen's national identifier	Member State specific	Member State specific. Max length: 100	LVRMRA89S19C722L
Adata	Attribute identifier (name)	Attribute names defined in STORK 2.0	30 Max length: maximum length defined for STORK attributes	title
L	Length of the AOI	4 hexadecimal digits	2	005f
AC	Foreign attribute country (A-PEPS)	alpha-2 string ISO3166-1 (uppercase)	2	UK
AP	Attribute provider identifier (name)	Member State specific	Max length:100	University of Kent
AVALUEPTR	Other information	AP specific	Max length: 200	345hfjhg6999dds
MAC	HMAC-SHA256 hash calculated on (name+value), encoded in Base64 on 44 ASCII characters	44 ASCII characters	44	7RbJC3PhylFQn62ZfklaLC1tjn+pRKSfCOMndnkIaoE=

Table 21: Data format of the AOI components

Note:

Since the values of “CC”, “AC”, “L” and “MAC” are of fixed length, four separators could be actually removed from the AOI structure, thus in case of stringent length constraints (see the next section on “Cookie length”) the cookie containing the AOI may be reduced by removing the separators mentioned.

Cookie length

According to the indication on number and size limits in Internet Explorer (http://support.microsoft.com/kb/306070) , for one domain name, each cookie is limited to 4,096 bytes. This total can exist as one name-value cookie pair of 4 kilobytes (KB) or as up to 20 name-value cookie pairs that total 4 KB.

In STORK 2.0, besides the cookie storing the AOI, there is another cookie used by Java for session management, which is about 30-40 bytes in length. Thus, the cookie storing the AOI can be at most 4096 – 40 = 4056 bytes in length.

4.4.5 Interpretation of the token

After authentication, the user should select the attributes to be retrieved from national providers. Since AOI have been created in previous interactions with STORK and stored in cookies in the user’s browser, the C-PEPS will read the cookie created in the previous interactions (Step 3 in *Figure 30*). Thus, the national APs will appear preselected in the user’s browser.

If additional attributes are to be retrieved from foreign countries, the user agent contains AOIs (token) associating the attributes identified by attribute names/identifier and the country from which those attributes can be retrieved. Such AOIs are processed by the C-PEPS (Step 4 in *Figure 30*). The countries read from the AOI will be thus appeared preselected in the user’s browser. By using the triple (CC, CEID, Adata) the C-PEPS selects among the cookies stored in the browser the one containing the correct AOI. It extracts the AC containing the A-PEPS identifier.

On A-PEPS, the national AP is also preselected if they have been stored in the AOI in previous interactions with STORK (Step 6 in *Figure 30*).

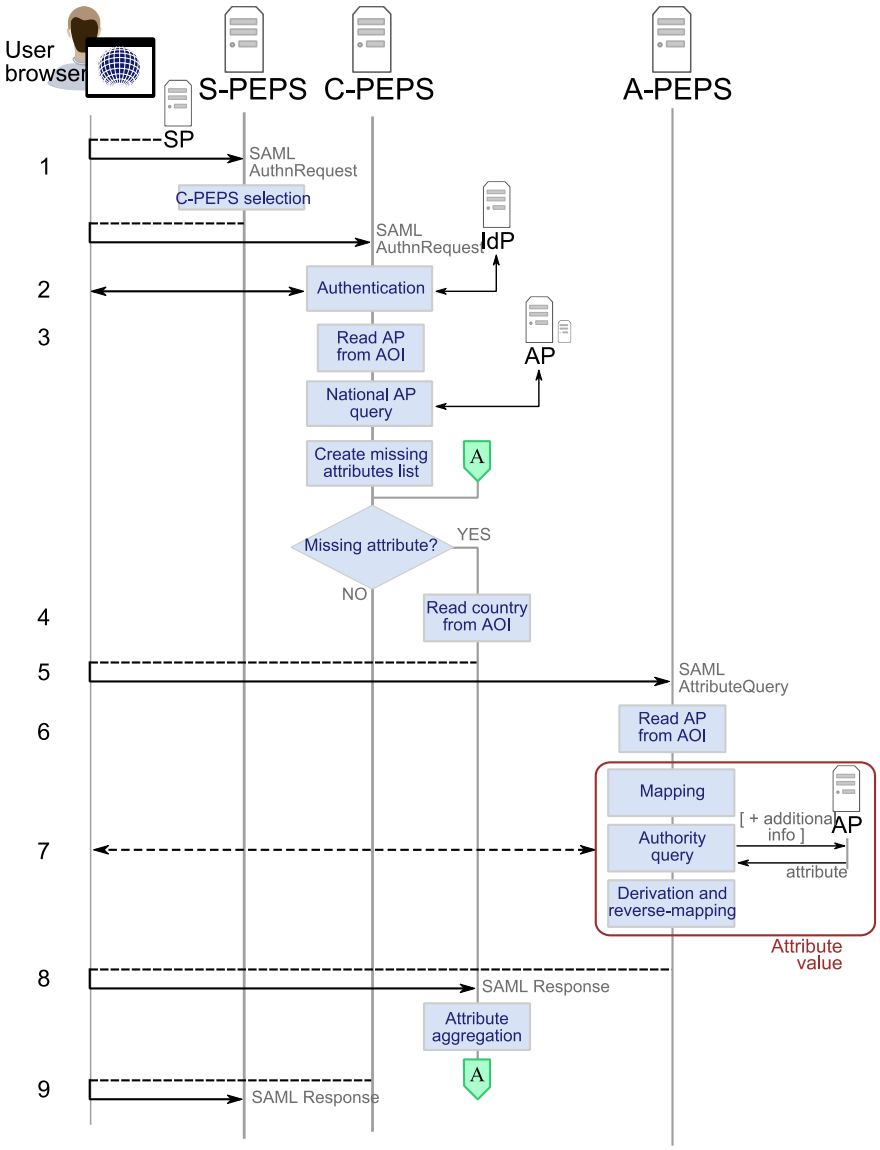


Figure 30. AOI processing

4.4.6 Maintenance of the token

If the cookies are lost, a user is forced to reselect countries and the Attribute providers providing the attributes required by services he wants to access. It might be helpful to investigate a "temporary browser storage backup service" aimed to save or backup the cookies at one time in the cloud or on an external support (NOT in the STORK platform, to avoid data protection issues) so that they can be easily restored if necessary.

4.5 SAML Unpackager

4.5.1 Introduction

The objective of the SAML unpackager is to read part of the SAML assertion to extract attributes types and values in user language to ask his consent. The SAML unpackager module will be included in the generate data type consent page to dynamically present the list of requested attributes and domain-specific attributes. For each attribute, the SAML unpackager module should extract from the SAML assertion:

- the name,
- the value,

- if the attribute is mandatory or not.

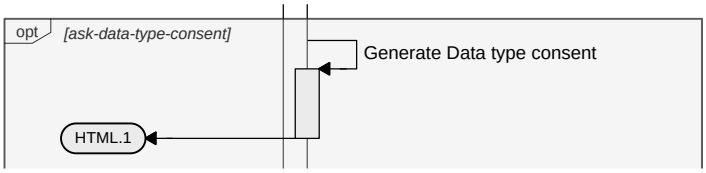


Figure 31. Sequence diagram for consent

4.5.2 Presentation of the module

The SAML unpackager module will use JavaScript to allow citizen browser side SAML decryption.

The SAML unpackager module will be called after data type consent page loading if the user clicks the corresponding button, and will present the citizen a waiting message during the SAML assertion treatment.

Actions:

- Validate SAML assertion
- Extract attributes types, values and required level (mandatory or not)

Data:

- Input :
    - SAML request
- Output :
    - In case of success : list of attributes, with, for each element :
        - name (String)
        - value (String)
        - Mandatory (Boolean)
    - In case of failure : error code corresponding to :
        - Invalid SAML assertion
        - Invalid signature
        - Error on attribute

Please note that common attribute names are in English language, and attribute values (mostly strings) may be in any European language. Thus the Spanish PEPS may present an unpackaged (partial) token saying that a citizen has done a “study” of “Natuurkunde” at the “Technische Universiteit Delft”.

5 Software design

The software design describes for each of tha Java packages the class diagram and the specification of the interface. Only the PEPS module is described in two separate sections, as they were developed by two separate teams.

Most processes in terms of D4.9 are reflected in only one package, with some exceptions:

- The AUB/PO/BA process is reflected in several packages: Commons, PEPS, VIDP, Specific, SAMLEngine, AttributeAggregation; due to its complexity and differences in implementation (centralised vs distributed)
- The PV is integrated in previous process
- The signature creation and validation is included in 2 packages: signatures and Document Transport Layer
- The Version Control is made common for PEPS/VIDP together with the VC for SPs.

For each of these packages a class diagram is included which explains the relationship between all classes. Also for each package an interface specification is included which specifies the available methods, their parameters and function.

5.1 PEPS

5.1.1 Description

The PEPS module is core of STORK 2.0. It is the project where the business logic and web layer is implemented.

This module includes two libraries:

- Commons library → implements utility classes to be used by the main project (PEPS).
- Specific library → implements reference code to be used and modified by each Member States regarding its own national requirements.

PEPS requires configurations which includes:

- An own keystore, to handle all the network SAML Requests (sign and validation).
- Main configuration file (peps.xml) and one for each of the two librarys: Commons (pepsUtils.properties) and Specific (specific.properties).
- Two log files, one for each PEPS' component (S-PEPS and C-PEPS) and one for Commons library.

5.1.2 Package specification

The following diagram shows the main classes involved in the AUB, BA and PV processes and the respective description.

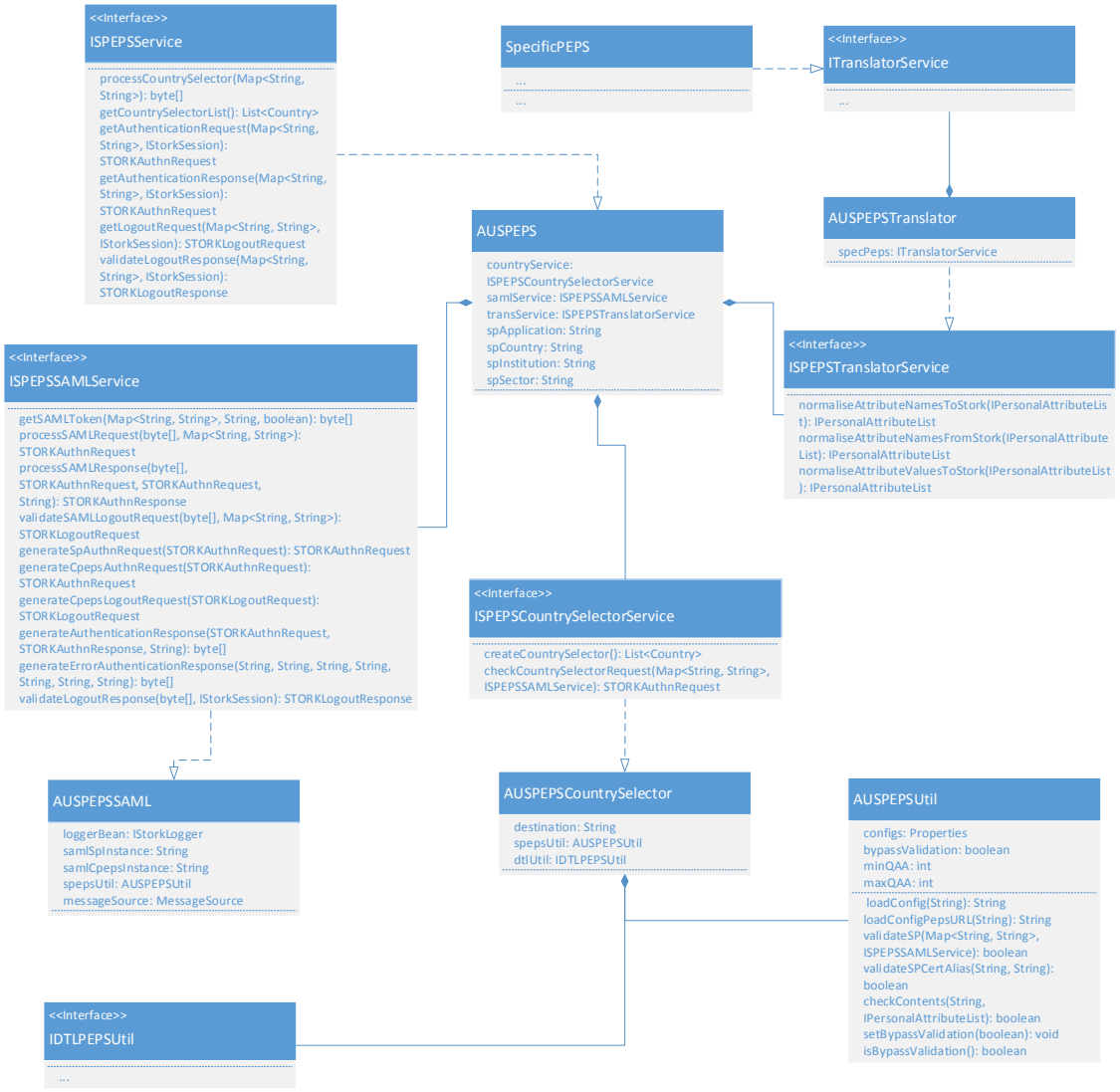


Figure 32. Class diagram for S-PEPS

Next, the classes depicted above (S-PEPS) are briefly explained:

Interface Class	***ISPEPSService***
Description	Interface for managing incoming requests coming from the Service Provider and forward them to the CPEPS, and Vice-versa.
Methods	• **processCountrySelector** (Map<String, String> parameters): byte[] Description: Generates a SAML token for the Country Selector. Output: The SAML token in the format of byte array. • **getCountrySelectorList** (): List<Country> Description: Generates the Country Selector List. Output: The List of known countries. • **getAuthenticationRequest** (Map<String, String> parameters,

Interface Class	***ISPEPSService***
	IStorkSession session): STORKAuthnRequest Description: Validates the origin of the request and of the Country Selected, and creates a SAML token to send to the C-PEPS. Output: An authentication request. • **getAuthenticationResponse** (Map<String, String> parameters, IStorkSession session): STORKAuthnRequest Description: Receives an Authentication Response, validates the origin of the response, and generates a SAML token to be sent to the SP. Output: An Authentication response. • **getLogoutRequest** (Map<String, String> parameters, IStorkSession session): STORKLogoutRequest Description: Receives a Logout Request and validates the origin, the destination and generates a SAML token to be sent to CPEPS. Output: A logout request. • **validateLogoutResponse** (Map<String, String> httpRequestParameters, IStorkSession session): STORKLogoutResponse Description: Receives a Logout Response, validates its fields and generates a SAML token to be sent to SP. Output: A logout response.

Table 22: Interface of ISPEPSService class of S-PEPS

Class	***AUSPEPS***
Description	This class serves as the middle-man in the communications between the Service Provider and the CPEPS. It is responsible for handling the requests coming from the Service Provider and forward them to the CPEPS, and Vice-versa.
Methods	See ***Table 40***

Table 23: Class AUSPEPS of S-PEPS

Interface Class	***ISPEPSCountrySelectorService***
Description	Interface to that holds the method to present the citizen the country selector form.
Methods	• **createCountrySelector** (): List<Country>

Interface Class	***ISPEPSCountrySelectorService***
	Description: Creates the CountrySelector form. Output: List of known countries and respective IDs. • **checkCountrySelectorRequest** (Map<String, String> parameters, ISPEPSSAMLService spepsSAMLService): STORKAuthnRequest Description: Creates authentication data and checks if a SP is allowed to access requested attributes. Output: An authentication request.

Table 24: Interface of ISPEPSCountrySelectorService class of S-PEPS

Class	***AUSPEPSCountrySelector***
Description	This class is used by AUSPEPS to create the Country Selector and to check the selected Country.
Methods	See *Table 24*

Table 25: Class AUSPEPS of S-PEPS

Interface Class	***ISPEPSSAMLService***
Description	Interface for working with SAMLObjects.
Methods	• **getSAMLToken** (Map<String, String> parameters, String errorCode, boolean isRequest): byte[] Description: Base64 decodes the incoming SAML Token. Output: The decoded SAML token in the format of byte array. • **processSAMLRequest** (yte[] samlToken, Map<String, String> parameters): STORKAuthnRequest Description: Validates the SAML Token request and checks if the SP is reliable. Output: An authentication request created from the SAML token. • **processSAMLResponse** (byte[] samlToken, STORKAuthnRequest authnData, STORKAuthnRequest spAuthnData, String remoteAddr): STORKAuthnResponse Description: Validates the response SAML Token. Output: The authentication response with a new PersonalAttributeList. • **validateSAMLLogoutRequest** (byte[] samlToken, Map<String, String> parameters): STORKLogoutRequest

Interface Class	***ISPEPSSAMLService***
	Description: Validates the SAML Token logout request and checks if the SP is reliable. Output: A logout request created from the SAML token. • **generateSpAuthnRequest** (STORKAuthnRequest authData): STORKAuthnRequest Description: Creates a SAML Authentication Request to send to SP. Output: A new authentication request with the SAML token embedded. • **generateCpepsAuthnRequest** (STORKAuthnRequest authData): STORKAuthnRequest Description: Creates a SAML Authentication Request to send to C-PEPS. Output: A new authentication request with the SAML token embedded. • **generateCpepsLogoutRequest** (STORKLogoutRequest reqData): STORKLogoutRequest Description: Creates a SAML Logout Request to send to C-PEPS. Output: A new logout request with the SAML token embedded. • **generateAuthenticationResponse** (STORKAuthnRequest authData, STORKAuthnResponse authResp, String ipUserAddress): byte[] Description: Generates a response's SAML token. Output: The response's SAML token in the format of byte array. • **generateErrorAuthenticationResponse** (String inResponseTo, String issuer, String destination, String ipUserAddress, String statusCode, String subCode, String message): byte[] Description: Generates a response's SAML token in case of error. Output: The response's SAML token in the format of byte array. • **validateLogoutResponse** (byte[] samlToken, IStorkSession session): STORKLogoutResponse Description: alidates STORK Logout response and generate a new one to be sent to SP. Output: A Logout Response

Table 26: Interface of ISPEPSSAMLService class of S-PEPS

Class	***AUSPEPSSAML***
Description	This class is used by AUSPEPS to get, process and generate SAML Tokens.

Class	***AUSPEPSSAML***
Methods	See *Table 26*

Table 27: Class AUSPEPSSAML of S-PEPS

Interface Class	***ISPEPSTranslatorService***
Description	Interface for normalizing IPersonalAttributeList.
Methods	• **normaliseAttributeNamesToStork** (IPersonalAttributeList pal): IPersonalAttributeList Description: Normalizes the attributes' name from a given IPersonalAttributeList to a common format. Output: The normalized personal attribute list. • **normaliseAttributeNamesFromStork** (IPersonalAttributeList pal): IPersonalAttributeList Description: Normalizes the attributes' name from a given IPersonalAttributeList to a specific format. Output: The normalized personal attribute list. • **normaliseAttributeValuesToStork** (IPersonalAttributeList pal): IPersonalAttributeList Description: The normalized personal attribute list. Output: The normalized personal attribute list.

Table 28: Interface of ISPEPSTranslatorService class of S-PEPS

Class	***AUSPEPSTranslator***
Description	This class is a service used by AUSPEPS to normalise attribute names and values.
Methods	See *Table 27*

Table 29: Class AUSPEPSTranslator of S-PEPS

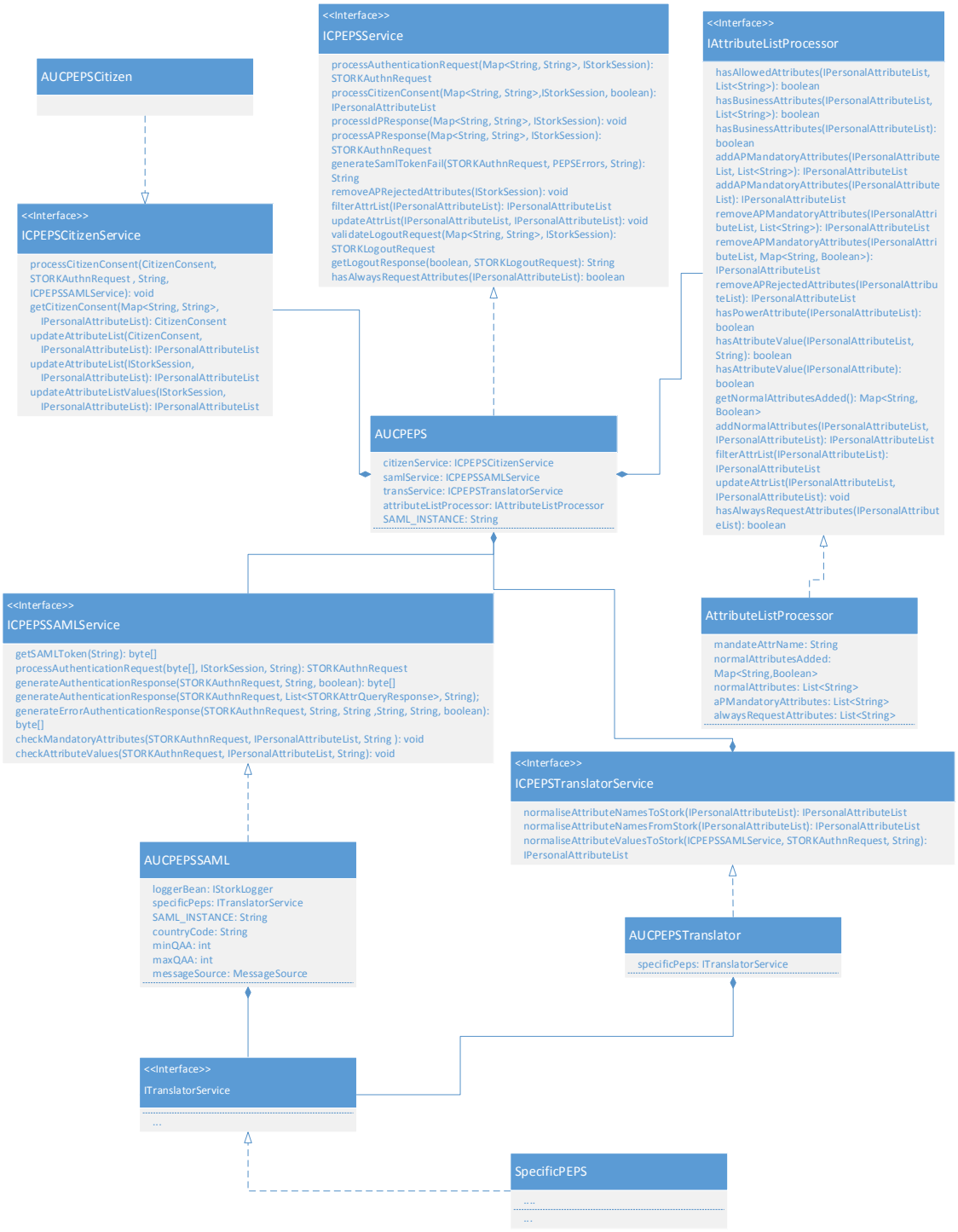


Figure 33. Class diagram for C-PEPS

Next, the classes depicted above (C-PEPS) are briefly explained:

Interface Class	***ICPEPSService***
Description	Interface for handling incoming requests coming from the S-PEPS and handling communication with the IdP and APs in order to authenticate the citizen.
Methods	• **processAuthenticationRequest** (Map<String, String> parameters,

Interface Class	*ICPEPSService*
	IStorkSession session): STORKAuthnRequest Description: Decodes the SAML Token, normalizes data from STORK format to specific format, and presents a consent-type form for the citizen to choose the optional attributes to be requested from the IdP/AP. Alternatively, the user can cancel the process. Output: The newly created authentication request. • **processCitizenConsent** (Map<String, String> parameters, IStorkSession session, boolean askConsentType): IPersonalAttributeList Description: Validates the consent sent by the citizen, then redirects the citizen to the IdP for the login process. Output: The Personal Attribute List updated with user consent. • **processIdPResponse** (Map<String, String> params, IStorkSession session): void Description: Processes the incoming response from the IdP and updates the personal attribute list, in session, if the IdP provided any attributes' value. Output: N.A. • **processAPResponse** (Map<String, String> parameters, IStorkSession session): STORKAuthnRequest Description: Normalizes the attributes to STORK format, generates the SAML Tokens to send to S-PEPS, and if required displays the consent-value form. Output: The new authentication request. • **generateSamlTokenFail** (STORKAuthnRequest authData, PEPSErrors errorId, String ipUserAddress): String Description: Generates an error SAML token. Output: A Base64 encoded SAML token. • **removeAPRejectedAttributes(**IStorkSession session): void Description: Remove all normal attributes that cannot be requested to AP. Output: N.A. • **filterAttrList** (IPersonalAttributeList attrList): IPersonalAttributeList Description: Updates list by filtering any attribute that must be requested instead of using a value obtained from cache (business and legal attrs). Output: the filtered list.

Interface Class	***ICPEPSService***
	• **updateAttrList** (IPersonalAttributeList cachedAttrList, IPersonalAttributeList requestedAttrsList): void Description: Updates the list of cached attrs by inserting the business and/or legal attrs requested by the user. Output: N.A. • **comparePersonalAttributeLists** (Map<String, String> parameters, IStorkSession session): STORKLogoutRequest Description: Validates the received Logout Request contained in the http parameters. Output: The Logout Request. • **getLogoutResponse** (boolean error, STORKLogoutRequest logoutReq): String Description: Generates the Logout Response. Output: A Logout Response. • **hasAlwaysRequestAttributes** (IPersonalAttributeList attributeList): boolean Description: Verifies if normal attribute list contains any attribute that we must always request (usually business attributes) Output: true is there is at least one attribute that must be requested or false otherwise.

Table 30: Interface of ICPEPSService class of C-PEPS

Class	***AUCPEPS***
Description	This class deals with the requests coming from the S-PEPS and communicates with the IdP and APs in order to authenticate the citizen, validate the attributes provided by him/her, and to request the values of the citizen's attributes.
Methods	See *Table 30*

Table 31: Class AUCPEPS of S-PEPS

Interface Class	***ICPEPSCitizenService***
Description	Interface that supplies methods for processing citizen-related matters.
Methods	• **processCitizenConsent** (CitizenConsent consent, STORKAuthnRequest authData, String ipUserAddress, ICPEPSSAMLService cpepsSAMLService): void Description: Checks if the citizen consent has all the required mandatory attributes.

Interface Class	***ICPEPSCitizenService***
	Output: N.A. • **getCitizenConsent** (Map<String, String> parameters, IPersonalAttributeList personalList): CitizenConsent Description: Constructs the Citizen Consent based on the checked boxes from consent-type form. Output: CitizenConsent containing the mandatory and optional attributes that PEPS has permission to request. • **updateAttributeList** (CitizenConsent citizenConsent, IPersonalAttributeList personalList): IPersonalAttributeList Description: Eliminates attributes without consent, and updates the Personal Attribute List. Output: The updated Personal Attribute List. • **updateAttributeList** (IStorkSession session, IPersonalAttributeList attributeList): IPersonalAttributeList Description: Replaces the attribute list in session with the one provided. Output: The updated Personal Attribute List. • **updateAttributeListValues** (IStorkSession session, IPersonalAttributeList attributeList): IPersonalAttributeList Description: Updates the values and the status of the attributeList in session. Output: The updated Personal Attribute List.

Table 32: Interface of ICPEPSCitizenService class of C-PEPS

Class	***AUCPEPSCitizen***
Description	This class is a service used by AUCPEPS to get, process citizen consent and to update attribute the attribute list on session or citizen consent based.
Methods	See *Table 32*

Table 33: Class AUCPEPSCitizen of C-PEPS

Interface Class	***ICPEPSSAMLService***
Description	Interface for communicating with the SAMLEngine.
Methods	• **getSAMLToken** (String samlToken): byte[] Description: Decodes the incoming SAML Token from Base64. Output: A byte array containing the decoded SAML Token. • **processAuthenticationRequest** (byte[] samlObj, IStorkSession

Interface Class	*ICPEPSSAMLService*
	session, String ipUserAddress): STORKAuthnRequest Description: Validates the SAML Token request. Output: The processed authentication request. • **generateAuthenticationResponse** (STORKAuthnRequest authData, String ipUserAddress, boolean isConsent): byte[] Description: Generates a SAML response Token. Output: A byte array containing the SAML Response Token. • **generateAuthenticationResponse** (final STORKAuthnRequest authData, • final List<STORKAttrQueryResponse> attrQueryResponse, final String ipUserAddress): byte[] Description: Generates a SAML response Token. Output: A byte array containing the SAML Response Token. • **generateErrorAuthenticationResponse** (STORKAuthnRequest authData, String errorCode, String subCode, String errorMessage, String ipUserAddress, boolean isAuditable): byte[] Description: Constructs a SAML response token in case of error. Output: A byte array containing the SAML Response. • **checkMandatoryAttributes** (STORKAuthnRequest authData, IPersonalAttributeList allAttrList, String ipUserAddr): void Description: Checks if all mandatory attributes have the status to Available. Output: N.A. • **checkAttributeValues** (STORKAuthnRequest authData, IPersonalAttributeList allAttrList, String ipUserAddress): void Description: Validates the values of the attributes. Output: N.A. • **validateLogoutRequest** (byte[] samlToken, IStorkSession session): STORKLogoutRequest Description: Validates the SAML Logout Request. Output: N.A.

Table 34: Interface of ICPEPSSAMLService class of C-PEPS

Class	*AUCPEPSSAML*
Description	This class is used by AUCPEPS to get, process and generate SAML Tokens. Also, it checks attribute values and mandatory attributes.

Class	***AUCPEPSSAML***
Methods	See *Table 34*

Table 35: Class AUCPEPSCitizen of C-PEPS

Interface Class	***ICPEPSTranslatorService***
Description	Interface for normalizing the IPersonalAttributeList.
Methods	• **normaliseAttributeNamesToStork** (IPersonalAttributeList pal): IPersonalAttributeList Description: Normalizes the attributes' name from a given IPersonalAttributeList to a common format. Output: The normalized personal attribute list. • **normaliseAttributeNamesFromStork** (IPersonalAttributeList pal): IPersonalAttributeList Description: Normalizes the attributes' name from a given IPersonalAttributeList to a specific format. Output: The normalized personal attribute list. • **normaliseAttributeValuesToStork** (ICPEPSSAMLService samlService, STORKAuthnRequest authData, String ipUserAddress): byte[] Description: Normalizes the attributes' values from a given IPersonalAttributeList to the common format. Output: The normalized personal attribute list's values. • **deriveAttributesToStork** (ICPEPSSAMLService samlService, IStorkSession session, STORKAuthnRequest authData, String ipUserAddress): IPersonalAttributeList Description: Derives the attributes' name to a common format. Updates the original Personal Attribute List, stored in the session, based on the values of attrList. Output: The new personal attribute list with the derived attributes. • **deriveAttributesFromStork** (IPersonalAttributeList pal): IPersonalAttributeList Description: Derives the attributes' name to a specific format. Output: The new personal attribute list with the derived attributes.

Table 36: Interface of ICPEPSTranslatorService class of C-PEPS

Class	***AUCPEPSTranslator***
Description	This class is a service used by AUCPEPS to normalise attribute names and values.

Class	***AUCPEPSTranslator***
Methods	See *Table 36*

Table 37: Class AUCPEPSCitizen of C-PEPS

Interface Class	***IAttributeListProcessor***
Description	Interface for AttributeListProcessor.
Methods	• **hasAllowedAttributes** (final IPersonalAttributeList attrList, final List<String> attributes): boolean Description: Checks if attribute list only contains allowed attributes. Output: true is all the attributes are allowed. • **hasBusinessAttributes** (final IPersonalAttributeList attrList, final List<String> normalAttributes): boolean Description: Lookup for business attribute. Output: true is at least one business attribute was requested. • **hasBusinessAttributes** (final IPersonalAttributeList attrList): boolean Description: Lookup for business attribute in normal attribute list (loaded by implementation). Output: true is at least one business attribute was requested. • **addAPMandatoryAttributes** (final IPersonalAttributeList attrList, final List<String> attributes): IPersonalAttributeList Description: Add eIdentifier, name, surname, and DateOfBirth attributes to get business attributes from some AP. Output: the requested attribute list and the new attributes added (eIdentifier, name, surname, and DateOfBirth). • **addAPMandatoryAttributes** (final IPersonalAttributeList attrList): IPersonalAttributeList Description: Adds eIdentifier, name, surname, and DateOfBirth attributes, loaded by implementation, to get business attributes from some AP. Output: the requested attribute list and the new attributes added (eIdentifier, name, surname, and DateOfBirth). • **removeAPMandatoryAttributes** (final IPersonalAttributeList attrList, final List<String> attributes): IPersonalAttributeList Description: Removes from attribute list the given list of attributes. Output: the requested attribute list and the attributes removed. • **removeAPMandatoryAttributes** (IPersonalAttributeList attrList, Map<String, Boolean> attributes): IPersonalAttributeList Description: Removes from attribute list the given list of attributes

Interface Class	*IAttributeListProcessor*
	and change attributes status if attribute was optional in the request. Output: the requested attribute list and the attributes removed. • **removeAPRejectedAttributes** (IPersonalAttributeList attrList): IPersonalAttributeList Description: Removes from attribute list the STORK list of attributes. Output: the attribute list without rejected attributes. • **hasPowerAttribute** (IPersonalAttributeList attrList): boolean Description: Checks if mandate attribute exist in the requested Attribute List. Power attribute name to lookup is loaded by implementation. Output: true if mandate attribute exists or false otherwise. • **hasAttributeValue** (final IPersonalAttributeList attrList, final String attrName): boolean Description: Checks if attribute name was requested and has value. Output: true if attribute was requested and has value or false otherwise. • **hasAttributeValue** (final PersonalAttribute attr): boolean Description: Checks if attribute has value. Output: true if has value. • **getNormalAttributesAdded** (): Map<String, Boolean> Description: Gets a map (attribute name, attribute isRequired) of attributes added to attribute list. Output: the Map of attributes added and if is required to attribute list. • **addNormalAttributes** (IPersonalAttributeList attrList, IPersonalAttributeList allAttrList): IPersonalAttributeList Description: Add normal attributes to personal attribute list if exist in original list (allAttrList). Output: the attributes list updated. • **filterAttrList** (IPersonalAttributeList attrList): IPersonalAttributeList Description: Updates list by filtering any attribute that must be requested instead of using a value obtained from cache (business and legal attrs). Output: the filtered list • **updateAttrList** (IPersonalAttributeList cachedAttrList, IPersonalAttributeList requestedAttrsList): void

Interface Class	***IAttributeListProcessor***
	Description: Updates the list of cached attrs by inserting the business and/or legal attrs requested by the user. Output: N.A. • **hasAlwaysRequestAttributes** (IPersonalAttributeList attributeList): boolean Description: Verifies if normal attribute list contains any attribute that we must always request (usually business attributes). Output: true if there is at least one attribute to be request or false otherwise.

Table 38: Interface of IAttributeListProcessorclass of C-PEPS

Class	***IAttributeListProcessor***
Description	This class is utility to process IPersonalAttributeList.
Methods	See ***Table 38***

Table 39: Class AUCPEPSCitizen of C-PEPS

Figure 34. Class diagram for Specific Module

Next, the classes depicted above are briefly explained:

Interface Class	***IAUService***
Description	Represents Specific Authentication methods to be implemented by each Member State.
Methods	• **prepareCitizenAuthentication**(IPersonalAttributeList personalList, Map<String, Object> parameters, Map<String, Object> requestAttributes, IStorkSession session): byte[] Description: Prepares the citizen to be redirected to the IdP. Output: the SAML Request. • **preparePVRequest**(IPersonalAttributeList personalList, Map<String, Object> parameters, Map<String, Object> requestAttributes, IStorkSession session): byte[] Description: Prepares the citizen to be redirected to the Power Validation module. Output: the SAML Request. • **authenticateCitizen**(IPersonalAttributeList personalList, Map<String, Object> parameters, Map<String, Object> requestAttributes): IPersonalAttributeList Description: Authenticates a citizen. Output: the Personal Attribute requested and obtained in the Authentication process. • **prepareAPRedirect**(IPersonalAttributeList personalList, Map<String, Object> parameters, Map<String, Object> requestAttributes, IStorkSession session): boolean Description: Prepares the Citizen browser to be redirected to the AP. Output: true in case of no error. • **getAttributesFromAttributeProviders**(IPersonalAttributeList personalList, Map<String, Object> parameters, Map<String, Object> requestAttributes): IPersonalAttributeList Description: Prepares the Citizen browser to be redirected to the AP. Output: Returns the Personal attribute list updated by AP. • **getAttributesWithVerification** IPersonalAttributeList personalList, Map<String, Object> parameters, Map<String, Object> requestAttributes, IStorkSession session, String auProcessId): boolean Description: Get the attributes from the AP with verification. Output: true if the attributes were correctly verified. • **processAuthenticationResponse** (byte[] samlToken, session):

Interface Class	***IAUService***
	STORKAuthnResponse Description: Validates a SAML Response. Output: the STORKAuthnResponse associated with the validated response. • **generateErrorAuthenticationResponse** (String inResponseTo, String issuer, String assertionURL, String code, String subcode, String message, String ipUserAddress): byte[] Description: Compares two given personal attribute lists. Output: true if the original list contains the modified one. False otherwise. • **comparePersonalAttributeLists** (IPersonalAttributeList original, IPersonalAttributeList modified): boolean Description: Validates a SAML Response. Output: the SAML Response. • **prepareAttributeRequest** (IPersonalAttributeList personalList, Map<String, Object> parameters, IStorkSession session): byte[] Description: Prepares the citizen to be redirected to the AtP. Output: the SAML Request. • **processAttributeResponse** (byte[] samlToken, IStorkSession session): STORKAttrQueryResponse Description: Validates a SAML Response. Output: the STORKAttrQueryResponse associated with the validated response.

Table 40: Interface of IAUService class of Specific Module

Interface Class	***ITranslatorService***
Description	Represents attribute normalization methods to be implemented by each member state.
Methods	• **normaliseAttributeNamesToStork** (IPersonalAttributeList personalList): IPersonalAttributeList Description: Translates the attributes from local format to STORK format. Output: The Personal Attribute List with normalised attributes. • **normaliseAttributeValuesToStork** (IPersonalAttributeList personalList): IPersonalAttributeList

Interface Class	*ITranslatorService*
	Description: The PersonalAttributeList with normalised values. Output: the SAML Request. • **normaliseAttributeNamesFromStork** (IPersonalAttributeList personalList): IPersonalAttributeList Description: Translates the attributes from STORK format to local format. Output: The PersonalAttributeList with normalised attributes. • **deriveAttributeFromStork** (IPersonalAttributeList personalList): IPersonalAttributeList Description: Derive Attribute Names To STORK format. Output: The PersonalAttributeList with derived attributes. • **deriveAttributeToStork** (IPersonalAttributeList personalList): IPersonalAttributeList Description: Derive Attribute Names from STORK format. Output: The PersonalAttributeList with derived attributes. • **checkAttributeValues** IPersonalAttributeList personalList): boolean Description: Validate the values of the attributes. Output: True, if all the attributes have values. False, otherwise.

Table 41: Interface of ITranslatorService class of Specific Module

Interface Class	*SpecificPEPS*
Description	This class implements ITranslatorService and IAUService interfaces and is also a reference code to be used by each member state.
Methods	See *Table 40* and *Table 41*

Table 42: Class SpecificPEPS of Specific Module

5.2 PEPS/V-IDP Attribute Aggregation

5.2.1 Description

The PEPS/V-IDP Attribute Aggregation is in charge of retrieving and aggregating domain specific requested attributes.

AUB Part 2 is part of the PEPS module.

5.2.2 Package specification

The following table describes the main classes involved in AUB process part 2.

Interface Class	***IAPEPService***
Description	Interface for handling attribute requests.
Methods	• **getAttributeProviderSelectorList(final boolean remoteApeps)** : List<AttributeProvider> Description: Generates the Attribute Provider Selector List. Input parameters: remoteApeps: True if this is remote A-PEPS. Output: The List of known Attribute Providers. • **getAllowedProviderSelectorList(final List<AttributeProvider> providers, final IPersonalAttributeList pal)** : Hashtable<String, List<AttributeProvider>> Description: Generates an Attribute Provider Selector List for each Attribute Name. Inpute parameters: providers: The List of known Attribute Providers. pal: The List of Attribute Names. Output: The Hashtable with a list of Attribute Providers for each Attribute Name. • **getAttributeNameSelectorList(final IPersonalAttributeList exclude)**: List<AttributeName> Description: Generates the Attribute Name Selector List. Output: The List of known Attribute Names. • **getAttributeNameSessionSelectorList(final IPersonalAttributeList sessionPal)**: List<AttributeName> Description: Generates the Attribute Name Selector List. Output: The List of known Attribute Names. • **getCountrySelectorList()** : List<Country> Description: Generates the Country Selector List. Output: The List of known Countries. • **processCitizenAPSelection final Map<String, String> parameters, final IPersonalAttributeList personalList, boolean remoteApeps)**: IAttributeProvidersMap Descrption: Validates citizen's selection of APs and groups the attributes to be requested per AP. Input Parameters: parameters: A map of the selected attribute providers. personalList: The personal attribute list. remoteApeps: True if we are running in REMOTE A-PEPS mode. Output: Map of Attribute Providers with the respective Attributes or null if something is wrong (i.e. invalid user selection). • **processCitizenMoreAPSelection(final Map<String, String> parameters, final IPersonalAttributeList personalList, final IPersonalAttributeList gatheredList)**: IAttributeProvidersMap Description: Validates citizen's selection for More attributes of APs

Interface Class	***IAPEPService***
	and groups the attributes to be requested per AP. Input Paramters: parameters: A map of the selected attribute providers. personalList The personal attribute list. gatheredList The personal attribute list already gathered. Output: map of Attribute Providers with the respective Attributes or null if something is wrong (i.e. invalid user selection). • **filterCitizenAttributeList(final IPersonalAttributeList personalList)** : IPersonalAttributeList Description: Filters the citizen's attribute list in order to return only the attributes that will be sent to APs. Input parameters: personalList The personal attribute list. Output: The filtered personal attribute list. • **compareAttributeLists(final IPersonalAttributeList sessionList, final IPersonalAttributeList providerList, final IStorkSession session)** : boolean Description: Compare the two Attribute Lists in order to verify the response from the Attribute Source (either provider or country). Input paramters: sessionList The list that was requested. providerList The list inside the response. session The current session. Output: true if the list is accurate (contains all attributes and all required elements are filled) • **prepareAPEPSRequest(final IPersonalAttributeList attrList, final Map<String, String> parameters, final IStorkSession session)** : STORKAttrQueryRequest Description: Creates a STORKAttrQueryRequest to send to APEPS (remote CPEPS) on attribute collection from remote countries. Input Paramerers: attrList: List of personal attributes to gather parameters: HTTP Request parameters session: Current HTTP session Output: The APEPS request • **processAPEPSRequest(final Map<String, String> parameters, final IStorkSession session)** : STORKAttrQueryRequest Description: Decodes the SAML Token, normalizes data from STORK format to specific format, and stores the request to session. Input parameters: Parameters: HTTP Request parameters Session: Current HTTP session Output: STORKAuthnRequest created from the SAML Token • **prepareAPEPSResponse(final Map<String, String> parameters, final IStorkSession session)** : byte[]

Interface Class	***IAPEPService***
	Description: Decodes and validates the SAML Token, normalizes data from STORK format to specific format. Input parameters: Parameters: HTTP Request parameters param session: Current HTTP session • **processAPEPSResponse(final Map<String, String> parameters, final IStorkSession session)** : STORKAttrQueryResponse Description: Decodes and validates the SAML Token, normalizes data from STORK format to specific format. Input parameters: Parameters: HTTP Request parameters param session: Current HTTP session **Output:** The STORKAttributeQueryRequest

Table 43: Class SpecificPEPS of Specific Module

Interface Class	***IAPEPSSAMLService***
Description	Interface for communicating with the SAMLEngine for APEPS operations.
Method	• **checkMandatoryAttributes(STORKAttrQueryRequest attrData, String ipUserAddress)**: Description: Checks if all mandatory attributes have the status to Available. Input parameters: authData The authentication request. ipUserAddr The citizen's IP address. • **generateAttrQueryResponse(STORKAttrQueryRequest attrData, List<STORKAttrQueryResponse> responses, String ipUserAddress)** : byte[] Description:Generates the attribute query response object with the collected attribute values to send to caller CPEPS Input parameters: attrData: The attribute query request ipUserAddress: The citizen's IP address. Output: The Attribute Query Response • **generateErrorAttrQueryResponse(STORKAttrQueryRequest attrData, String code, String subCode, String errorMessage, String ipUserAddress, boolean isAuditable)** : byte[] Description: Constructs a SAML response token in case of error. Input parameters: authData: The authentication request. errorCode: The status code. subCode: The sub status code. errorMessage: The error message.

Interface Class	*IAPEPSSAMLService*
	ipUserAddress: The citizen's IP address. isAuditable: Is a auditable saml error? Output: A byte array containing the SAML Response. • **getSAMLToken(String samlToken)** : byte[] Description: Decodes the incoming SAML Token from {@link Base64}. Input parameters: samlToken: The Token to be decoded. Output: A byte array containing the decoded SAML Token. • **processAttrQueryRequest(byte[] samlObj, IStorkSession session, String ipUserAddress)** : STORKAttrQueryRequest Description: Validates the SAML Token request. Input parameters: samlObj: the SAML Token to be validated. session: The current session. ipUserAddress The citizen's IP address. Output: The processed authentication request. • **processAttrQueryResponse(byte[] samlObj, IStorkSession session, String ipUserAddress)** : STORKAttrQueryResponse Description: Validates the attribute query response and generates a STORKAttrQueryResponse from the saml token Input parameters: samlObj: Response saml token session: Current session ipUserAddress: The citizen's IP address Output: A STORKAttrQueryResponse object generated from response saml token • **generateAttrQueryRequest(STORKAttrQueryRequest authData)** : STORKAttrQueryRequest Description: Creates the saml token for the given STORKAttrQueryRequest object Input parameters: authData: STORKAttrQueryRequest to generate saml token Output: A STORKAttrQueryRequest object with the saml token set • **checkAttributeValues(STORKAttrQueryRequest attrData, String ipUserAddress)** : Description: Validates the values of the attributes. Input parameters: authData: The authentication request. ipUserAddress: The citizens' IP address.

Table 44: Interface of communication with SAML Engine

Interface Class	***IAASPEPSIDDiscoveryService***
Description	Interface that holds the methods to discover the APs the citizen has selected.
Methods	• **discoverAttributeProviderURL(Map<String, String> parameters, IPersonalAttributeList personalList)** : IAttributeProvidersMap Description: Extracts and groups the attributes to be requested by each AP. Input parameters: parameters A map of the selected attribute providers. personalList The personal attribute list. Ouptut: Map of Attribute Providers with the respective Attributes. • **discoverMoreAttributeProviderURL(Map<String, String> parameters, IPersonalAttributeList personalList)** : IAttributeProvidersMap Description: Extracts and groups the attributes to be requested by each AP. This method is used when collecting more attributes, in order to copy the friendly name of a Personal Attribute to the name. Input parameters: parameters: A map of the selected attribute providers. personalList: The personal attribute list. Output: Map of Attribute Providers with the respective Attributes. • **isAttributeProviderValid(String providerId, String attrName, boolean remoteApeps)** : Boolean Description: Check if the provider ID is valid or not. Input parameters: providerId: The providerId to verify attrName: The Attribute Name requested from this AP remoteApeps: True if we are running in REMOTE A-PEPS mode Output: true if the provider ID is valid • **isAttributeNameValid(String attributeId)** : boolean Description: Check if the attribute ID is valid or not. Input parameters: attributeId: The attributeId to verify Output: true if the attribute ID is valid • **isCountryValid(String countryId)** : Boolean Description: Check if the country ID is valid or not. Input parameters: countryId: The countryId to verify Output: true if the country ID is valid

Table 45: Interface AASPEPSIDDiscovery class

Interface Class	***IAASPEPSAttributeProcessorService***
Description	Interface that holds the methods to filter Attributes and process the responses from Attribute Providers.
Methods	• **filterPersonalAttributes(IPersonalAttributeList personalList)** : IPersonalAttributeList Description: Filters the personal attribute list in order to return only the list that will be requested to the Attribute Providers. Input parameters: personalList The personal attribute list. Output: The filtered personal attribute list. • **compareAttributeLists(IPersonalAttributeList sessionList, IPersonalAttributeList providerList)** : boolean Description: Compare the two Attribute Lists in order to verify the response from the Attribute Source (either provider or country). Input parameters: sessionList: The list that was requested. providerList: The list inside the response. Output: true if the list is accurate (contains all attributes and all required elements are filled)

Table 46: Interface AASPEPSAttributeProcessor class

Interface Class	***IAASPEPSAttributeProviderSelectorService***
Description	Interface that holds the methods to present the citizen the attribute provider selector form.
Methods	• **createAttributeProviderSelector(boolean remoteApeps)** : List<AttributeProvider> Description: Creates the AttributeProviderSelector form. Input parameters: remoteApeps: True if this is remote A-PEPS. Output: List of known Attribute Providers and respective IDs. • **createAttributeNameSelector(IPersonalAttributeList exclude)** : List<AttributeName> Description: Creates the AttributeNameSelector form. Input parameters: exclude: The exclude list Attribute Names Output: List of known Attribute Names. • **createAttributeNameSessionSelector(IPersonalAttributeList sessionPal)** : List<AttributeName> Description: Creates the AttributeNameSelector form. Input parameters: sessionPal: The PAL from the session to retrieve the list of Attribute Names Output: List of known Attribute Names.

Interface Class	***IAASPEPSAttributeProviderSelectorService***
	• **allowedAttributeProviderSelector(List<AttributeProvider> providers, IPersonalAttributeList pal)** : Hashtable<String, List<AttributeProvider>> Description: Creates the AllowedAttributeProviderSelector form. Input parameters: providers: The List of known Attribute Providers. pal: The List of Attribute Names. Output: The Hashtable with a list of Attribute Providers for each Attribute Name.

Table 47: Interface AASPEPSAttributeProviderSelector class

Interface Class	***IAASPEPSCountrySelectorService***
Description	Interface that holds the method to present the citizen the country selector form.
Methods	• **createCountrySelector()** : Description: Creates the CountrySelector form. Output: List of known countries and respective IDs.

Table 48: Interface Country Selector class of Anonymity

Interface Class	***IAPEPSTranslatorService***
Description	Interface to normalise attribute names and values.
Methods	• **normaliseAttributeNamesToStork(IPersonalAttributeList pal)** : IPersonalAttributeList Description: Normalizes the attributes' name from a given {@link IPersonalAttributeList} to a common format. Input parameters: pal: The personal attribute list to normalize. Output: The normalized personal attribute list. • **normaliseAttributeNamesFromStork(IPersonalAttributeList pal)** : IPersonalAttributeList Describe: Normalizes the attributes' name from a given {@link IPersonalAttributeList} to a specific format. Input parameters: pal: The personal attribute list to normalize. Output: The normalized personal attribute list. • **normaliseAttributeValuesToStork(IAPEPSSAMLService samlService, STORKAttrQueryRequest attrData, String ipUserAddress)** : IPersonalAttributeList Description: Normalizes the attributes' values from a given {@link IPersonalAttributeList} to the common format. Input parameters: samlService: The SAML Service.

Interface Class	*IAPEPSTranslatorService*
	authData: The authentication request. ipUserAddress: The citizen's IP address. Output: The normalized personal attribute list's values. • **deriveAttributesToStork(IAPEPSSAMLService samlService, IStorkSession session, STORKAttrQueryRequest attrData, String ipUserAddress)** : IPersonalAttributeList Description: Derives the attributes' name to a common format. Updates the original Personal Attribute List, stored in the session, based on the values of attrList. Input parameters: samlService: The SAML Service. session: The session containing the original attribute list to update. authData: The authentication request. ipUserAddress: The citizen's IP address. Output: The new personal attribute list with the derived attributes. • **deriveAttributesFromStork(IPersonalAttributeList pal)** : IPersonalAttributeList Description: Derives the attributes' name to a specific format. Input parameters: pal: Personal attribute list with the attributes to derive. Output: The new personal attribute list with the derived attributes.

Table 49: Interface AASPEPSTranslator class

5.3 V-IDP

5.3.1 Description

V-IDP acts as a gateway between SP, S-PEPS and C-PEPS entities. In the course of these deployments, the same V-IDP software supports both deployment modes and roles in a country following the decentralized deployment model formerly referred to as “middleware” (in this case, Austria) and abroad, integrating the broad range of country-specific and STORK-specific functionalities and interfaces in one coherent package. This approach enables easier deployment and maintenance of VIDP instances and lowers the integration and support costs.

As it was the case in STORK, Austria integrated the STORK 2.0 functionality into its production component referred to a “MOA”. This allows Austrian SPs beyond the actual STORK 2.0 pilots to activate STORK 2.0 simply by adjusting their configurations, as the STORK 2.0 functionality is already given in the components the SP operates in the decentralized mode. As Austria is the only country in STORK 2.0 that follows the decentralized approach, the common V-IDP functions are integrated in its production component (as used in STORK1 with AT and DE in a so-called “MARS” architecture). Therefore, the terms V-IDP (as the decentralized component operating the STORK 2.0 protocol) and MOA (that complements it by country-specific functions similar to country-specific parts in a PEPS) might interchange. This also highlights that STORK 2.0 support is tightly integrated in the AT eGov enabler software components “MOA”. The two terms therefore refer to the roles the single instance or system takes. While V-IDP refers to the instance deployed in other MS (interfacing with an S-PEPS), MOA

describes the instance in the role that provides STORK integration for the service providers deployed in country applying a decentralized deployment model (interfacing with a C-PEPS).

This section provides the overview of the V-IDP applications, deployment and packages provided in the V-IDP solution.

5.3.2 Applications

There are two applications that provide the functionality of V-IDP package:***Name***	***Description***
moa-id-auth	Provides all services and functionality of V-IDP and of the services it depends on
moa-id-configuration	Exposes web interface for configuration of V-IDP services and functionality

Table 50: Applications included in the V-IDP package

Additionally, in the course of V-IDP deployments, the installation of the third application is recommended as well. BKUOnline[11] (MOCCA suite) is a part of the MS–specific components. Acting as an optional module that facilitates easier deployment, it is not considered as a part of V-IDP software itself. It represents a separate project that provides one of implementations that exposes functionality of Austrian citizen card environment.

5.3.3 Modules

This section provides the description of the most relevant software modules of V-IDP software.

5.3.3.1 MOA-ID

Name	***Description***
moa-id-auth	Contains an implementation of Web service used for authentication and communication with service providers and other STORK nodes.
moa-id-configuration	Contains the implementation of the web interface used to configure MOA-ID and VIDP functionality.
moa-id-lib	Contains API that provides the functionality of MOA-ID and STORK related processes and functions
moa-id-modules	This module contains additional functionality that enables the dynamic extensions of MOA-ID and VIDP. It enables the definition of tasks that are then executed by process engine, in the different phases of the process flows. This module also contains two submodules that implement the modular support for monitoring functionality and STORK related processes.

[11] https://joinup.ec.europa.eu/software/mocca/home

Name	*Description*
Commons	The common STORK library that provides beans, Java Interfaces and utility classes to integrate PEPS/VIDP with SAML Engine.
SamlEngine	This common STORK library provides tools to support developers working with the Security Assertion Markup Language (SAML).

Table 51: The main software modules included in the V-IDP package

5.3.3.2 MOA-Common

This module provides logging and utility classes that are used by various components of VIDP solution.

5.3.3.3 MOA-SPSS

MOA-SPSS represents a set of modules that enable the applications to create and verify electronic signatures. It consists of the module for server signature (SS) and the module for signature verification (SP).

Module SS provides functionality for creation of XML-Signatures according to the interface specification of SecurityLayer[12]. It supports both the creation of software based signatures, and the ones using external Hardware Security Module (HSM). The applications can use this module through the Java-API or as a WebService.

Module SP enables the applications to verify XML and CMS signatures, as well as XAdES and CAdES based signatures produces according to the SecurityLayer specification.

The following table provides the overview and descriptions of the relevant modules included in the MOA-SPSS package.

Name	*Description*
moa-spss-lib	API that provides the functionality of modules for server signature (SS) and signature verification (SP)
moa-spss-ws	Web service that provides the functionality of modules for server signature (SS) and signature verification (SP)
moa-spss-tools	This module integrates helper processes used both by SP and SS portions of moa-spss

Table 52: Modules of MOA-SPSS package

5.3.4 Package descriptions

In this section provided are package descriptions separated by modules and components that provide particular functionality. This description includes the most relevant packages for VIDP functionality. Due to the complex structure and for the purpose of readability, the package names in the following descriptions are provided without prefixes common for each module. These prefixes are available at the beginning of each subsection.

[12] https://www.buergerkarte.at/konzept/securitylayer/spezifikation/20140114/core/core.html

5.3.4.1 Packages in moa-id-auth module

The package names provided in this section include the prefix *at.gv.egovernment.moa*. In order to provide readable descriptions and maintain proper formatting of the tables, this prefix has been omitted from the following tables. Therefore, all the package names contained in this section are assumed to begin with this this prefix, suffixed with respective package name.

5.3.4.1.1 Packages providing general functionality

Package name	***Description***
id.advancedlogging	Class used to perform the logging for statistical purposes
id.client	Contains the client and helper classes used to access supplementary register gateway (SZRGW)
id.config	Contains interfaces, implementations, utility classes, exceptions, factories and providers for configuration parameters used for authentication components and STORK, legacy and proxy configurations.
id.data	Contains interfaces and classes used to provide and hold authentication data used by various protocols and processes.
id.entrypoints	Contains the classes used for the processing and further routing of incoming web service requests.
id.iaik.config	Provides an implementation of the interfaces needed to initialize an IAIK JSSE TrustManager
id.moduls	This package contains the definitions of interfaces of actions, requests and responses provided in various processing steps. It furthermore contains the classes that enable the management of authentication and single-sign-on in various processing steps, including the supporting utility classes and storage handlers.
id.opensaml	Contains the helper functions for using OpenSAML in MOA components.
id.process	This package contains the interface definition for process engine used to manage the execution of processes. It contains the implementation of this functionality as well, including the related exceptions, enumerations and helper parsers. These are provided in the respective subpackages of this package, including model, task, event and execution context management and representation packages, and the package to support the integration with SpringWeb.
id.protocols	This package contains subpackages related to the four main protocols used for authentication: OAuth2, PVP2, STORK 2.0 and legacy Saml1.

Package name	*Description*
id.protocols.stork2	This package contains the classes that implement processing for STORK2 protocol. These include the two-step based process support for MOA process engine, the classes providing support for consent evaluation, attribute management and gathering, as well as mandate gathering, storage, and general data container.
id.storage	This package provides classes used for the storage of assertions and authentication sessions.
id.util	This package contains utility classes used to establish secure connections, store cryptographic constants and paramameters, verify QAA levels, XML processing and HTTP connection, forms and session management.

Table 53: Packages providing general functionality

5.3.4.1.2 Packages enabling authentication and STORK 2.0 specific flows

Package name	*Description*
id.auth	Contains API for Authentication Service, session and mandate management. Additionally this package contains the class supporting the initialization and management of web application components.
id.auth.builder	Contains the classes used to generate various objects and descriptions used in the various authentication and request processing steps. These include the support for assertions, authentication blocks and data, generation of BPK and DataURL references, as well as support for XMLSignatureRequests, InfoBoxRequests and login forms.
id.auth.data	This package provides the interfaces and classes defining and implementing the data containers for authentication and single-sign on.
id.auth.exception	Contains the exceptions used in various subpackages in the scope of moa.id.auth package.
id.auth.invoke	This package provides the class supporting the integration of signature verification using MOA-SPSS web service and API.
id.auth.modules	This package contains the interfaces and classes used for the definition, description, registration and integration of separate authentication modules and supported tasks. These objects are defined in respective subpackages, providing additional definitions of internal tasks available in MOA.
id.auth.parser	Provides classes used to support parsing of InfoBoxReadResponse and further verification integration with MOA-SPSS.

Package name	*Description*
id.auth.servlet	Provides classes that implement servlets and support functionalities for redirection, SSO and SLO, generation of iFrames, authentication and logout.
id.auth.stork	Defines interface and classes for verification and processing of STORK responses.
id.auth.validator	Defines an interface and provides an implementation for validation of responses, including ones of InfoboxReadResponse, CreateXMLSignatureResponse, IdentityLink and VerifyXMLSignatureResponse. The subpackages of this package include additionally the implementation of SZRGW client and utility classes to process SZRGW responses and manipulate their parts.

Table 54: Packages enabling the integrationof STORK 2.0 flows

5.3.4.1.3 Packages supporting the integration of MOA-ID modules

The following table contains the packages defined separately in the module integration package (moa-id-modules) that provides the description and integration of additional modules into the main application. The packages listed there are assumed to begin with *at.gv.egovernment.moa.id prefix*.

Package name	*Description*
auth.servlet	Servlet implementation of monitoring module used to periodicaly check the functionality and state of the applicaiton.
monitoring	Interface and implementation of test manager component. This package also contains the implementations of specific monitoring tasks, such as database test task and IdentityLink test.
auth.modules.stork	Contains module descriptor for the integration of STORK 2.0 authentication processes in the application flow.
auth.modules.stork.tasks	This package contains the implementations of STORK 2.0 specific actions, enabling their integration into MOA ID processing flow.

Table 55: Packages supporting the integration of MOA-ID modules

5.3.4.1.4 Packages integrating common STORK 2.0 functionality

VIDP integrates the modules that provide common STORK 2.0 core functionality. These include Commons and SamlEngine. Based on the V-IDP architecture, these modules are integrated without any significant changes imposed upon their structure. For the descriptions of the packages included in these modules please refer to the respective sections for Commons and SamlEngine.

5.3.4.2 Packages in MOA-ID-Configuration

The package names provided in this section begin with the prefix *at.gv.egovernment.moa.id*. In order to provide readable descriptions and maintain proper formatting of the tables, this prefix has been omitted from the following tables. Therefore, all the package names contained in this section are assumed to begin with this prefix.

Package name	***Description***
configuration.auth	This package contains the classes used to manage authenticated session's data, including the users's data and credentials.
configuration.auth.pvp2	This package includes utility classes used to build, filter and process metadata and attributes that support the authentication using PVP, AT specific protocol. Its subpackage also includes the servlets used to manage SLO in this context.
configuration.config	This package contains the configuration provider for the application as well as supporting and utility classes.
configuration.data	This package contains the classes used as data entities, DAOs and building elements used for the storage, description and processing of particular configuration entries and options. These additionally include the
configuration.data.pvp2	This package contains classes related to data descritions of users using PVP2 authentication protocol
configuration.data.oa	This package contains interface defining the description of online applications as used in persistent storage for configuration elements. It also contains the implementations specific for each protocol and configuration section, as supported by configuration interface.
configuration.exception	This package contains exception classes used by the configuration interface.
configuration.filter	This package contains utility classes used for processing, encoding and filtering of configuration elements.
configuration.helper	This package contains helper classes used for user interface specific adjustments, such as date/time presentation and processing, language support, and parsing of the data elements.
configuration.struts	This package and its subpackages contain implementations of specific classes used in the Struts framework. These include action handlers and integration with Hibernate.

Table 56: Packages in web configuration interface of V-IDP

Package name	*Description*
configuration.utils	This package contains utility classes used or management of request storage, processing and conversion of SAML elements and encryption of persistent storage elements.
configuration.validation	This package contains the constants used across the project. Its subpackages contain component specific validation classes.
configuration.validation.moaconfig	This package contains validation classes used for general configuration, as well as ones applied for the validation of PVP2 and STORK2 entries
configuration.validation.oa	This ackage contains validation classes used for the validation of online applications (service provider) configuration entries

Table 57: Other packages of the web configuration interface

5.3.4.3 Packages in MOA-Common

The package names provided in this section begin with the prefix *at.gv.egovernment*. In order to provide readable descriptions and maintain proper formatting of the tables, this prefix has been omitted from the following tables. Therefore, all the package names contained in this section are assumed to begin with this prefix.

Package name	*Description*
moa.logging	This package implements and wraps logging functionality used in MOA and V-IDP context
moa.util	This package contains common utility classes that support the generation processing of entitites, URLs, XPath expressions, messages, DOM elements and other entitites used across the project
moa.util.ex	This package contains exception handling classes

Table 58: Common VIDP packages

5.3.4.4 Packages in MOA-SPSS

The package names provided in this section begin with the prefix *at.gv.egovernment.moa*. In order to provide readable descriptions and maintain proper formatting of the tables, this prefix has been omitted from the following tables. Therefore, all the package names contained in this section are assumed to begin with this prefix.

5.3.4.4.1 Packages providing API and general functionality

Package name	*Description*
spss.api	This package contains the classes used for configuration and initialization of SPSS API and interfaces that provide functions for signature creation and verification.
spss.api.cmssign	Contains the interfaces that enable the definition and integration of CMS signature requests and responses.
spss.api.cmsverify	This package contains the interfaces that manage the verification of CMS signature requests and responses, including the definitions of data object and content references.
spss.api.common	In this package included are the classes used across the SPSS API, supporting various functions and transformations including the Base64 transformations, XSLT and XPath operations, filters and transformations. It furthermore includes the encapsulation classes that support the XML content, meta data, location references and binary content.
spss.api.impl	This package contains the implementation classes of the interfaces defined across the package.
spss.api.xmlbind	This package contains various parsers supporting the profiles, requests, transformations and responses.
spss.api.xmlsign	This package contains the interfaces used to support the integration and processing of CreateXMLSignatureRequest and CreateXMLSignatureResponse elements. It also included the interfaces that enable the integration of transformation profiles and encapsulate signature objects during various phases of signature creation.
spss.api.xmlverify	This package contains the interfaces used to support the integration and processing of VerifyXMLSignatureRequest and VerifyXMLSignatureResponse. It also includes the encapsulation and transformation classes.
spss.api.tsl.config	This package and its subpackages include the classes that support the integration and management of TSL used in the application. It also includes the configuration and connector for TSLs.
spss.api.tsl.utils	This package contains utility classes support the integration of TSL functionality in application.
spss.util	This package contains general utility classes used in the application.

Table 59: SPSS API packages

5.3.4.4.2 Packages providing support for SPSS server functions

Package name	*Description*
spss.server.config	In this package contained are the classes related to configuration of SPSS module. It also contains the classes that suppor the parsing of configuration files and suppor the integration of hardware crypto and key modules in the application. Furthemore the package contains configuration support for CRL and OCSP distribution points.
spss.server.iaik	This package and its subpackages contain the classes that provide the implementation profiles and auxiliary information for creation and verification of XML and CMS signatures. It furthermore includes the classes for XML manipulation, XPath and XSLT.
spss.server.iaik.pki	This subpackage contains the implementations of interfaces supporting certificate path validation and revocation checks.
spss.server.init	This package contains supporting classes for initialization and configuration implementation of SPSS web service
spss.server.invoke	In this package contained are various implementation and utility classes used by invocation of primary functionalitites related to generation and verification of XML and CMS signatures.
spss.server.logging	This package wraps the logging support for SPSS service.
spss.server.service	This package contains the web service endpoints for signature creation and verification, request handler and configuration servlet.
spss.server.transaction	This package contains transaction handling manager and context access.
spss.server.util	This package contains utility classes used by SPSS server module.

Table 60: SPSS server packages

5.4 SAMLEngine

5.4.1 Description

Next figure shows a functional view of the Authentication Engine implemented. From here on, the engine is called SAML Engine. The section follows a bottom-up approach to explain each component.

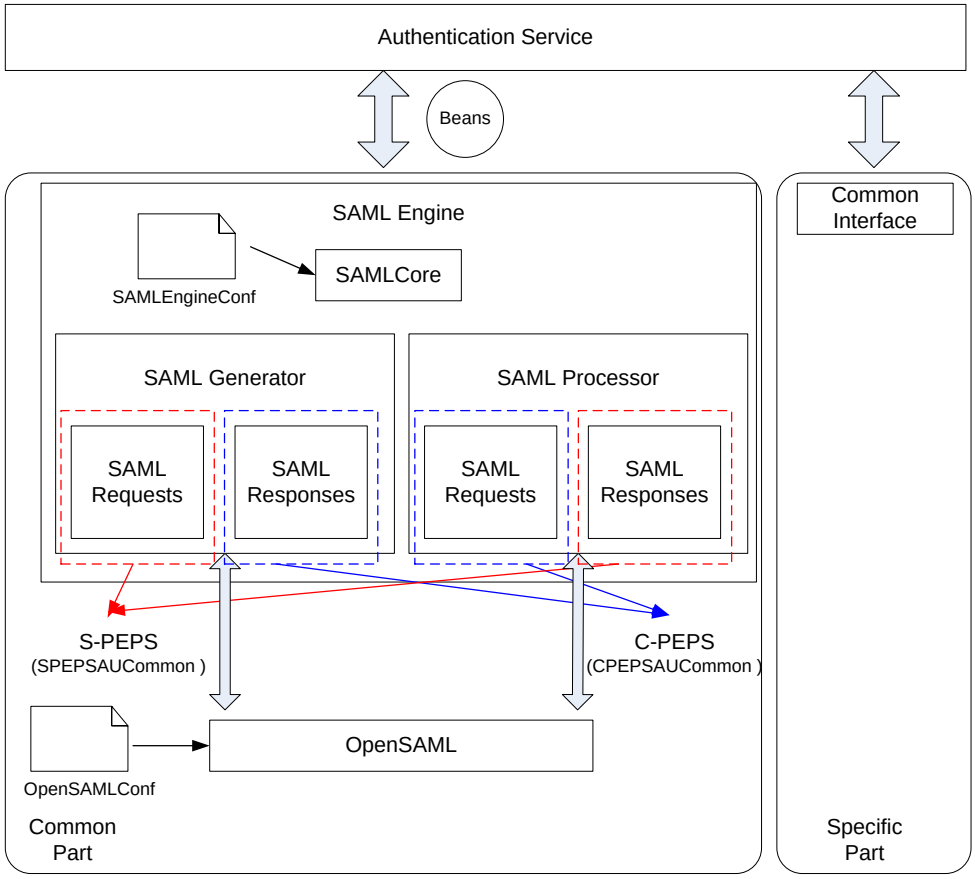


Figure 35 – Authentication/SAML engine: Model

The **Authentication Service Layer** is in charge of implementing the business logic of the PEPS authentication service itself, both for the C-PEPS and the S-PEPS. Below this layer, the functionality is split into the common and the specific parts. The figure above only details the common part, which is explained next.

The **SAML Engine** module is responsible for implementing the operations on SAML messages, both requests (S-PEPS) and responses (C-PEPS). This module is configured through the **SAML Core** submodule. Besides, and from a functional viewpoint, next submodules are differentiated:

- SAML Generator

  This part of the engine generates the SAML Tokens, which can be either SAML Authentication (with Attribute query as an extension) requests or SAML Assertions (Authentication and Attribute statements) responses.

- SAML Processor

  This part of the engine validates and processes the SAML Tokens above.

The S-PEPS functionality is covered by the SAML Generator → SAML Requests and SAML Processor → SAML Responses parts of the engine. That is, the S-PEPS will generate requests and process responses.

The C-PEPS functionality is covered by the SAML Processor → SAML Requests and SAML Generator →SAML Responses parts of the engine. That is, the C-PEPS will process requests and generate responses.

The **SAML Engine** manages SAML objects by means of the **OpenSAML** library.

The SAML-related information is transmitted from the SAML Engine layer to the Authentication Service layer through the identified **Beans**.

Furthermore, and as can be seen in the figure above, two **configurations files** are needed:

- OpenSAMLConf contains the configuration of the SAML library (OpenSAML).
- SAMLEngineConf contains the configuration needed for the operation of the PEPS SAML Engine.

Next subsections give more detail about each "box" identified in the figure above. In particular, next parts of the engine are described:

- OpenSAML
- SAML Engine
- Keystores management

5.4.2 OpenSAML

Package: OpenSAML specific

This subsection deals with XML signature processing (generation and validation) only. SAML token validation according the SAML 2.0 schema is not explained, but it is obviously necessary as a first step when parsing SAML tokens received from other PEPS. Other operations to be fulfilled while interacting with the OpenSAML, like library initialization and configuration, or SAML message generation and processing are explained further in SAML Engine description.

As shown in figure above, OpenSAML needs a configuration file, which corresponds to the file *OpenSAMLConf* and that establishes the configuration with which the library operates. It is supposed that no further or extra configuration will be needed except the default one. Please refer to OpenSAML for further information.

5.4.3 Basic Class Diagram (XML signature generation process)

Enveloped signatures are the only method formally prescribed in the XML Signature profile of the SAML specification. Next Figure depicts the most important classes from OpenSAML that must be used by the PEPS implementation, and in particular, by the PEPS Authentication Module, in order to generate the XML signature over a SAML Token.

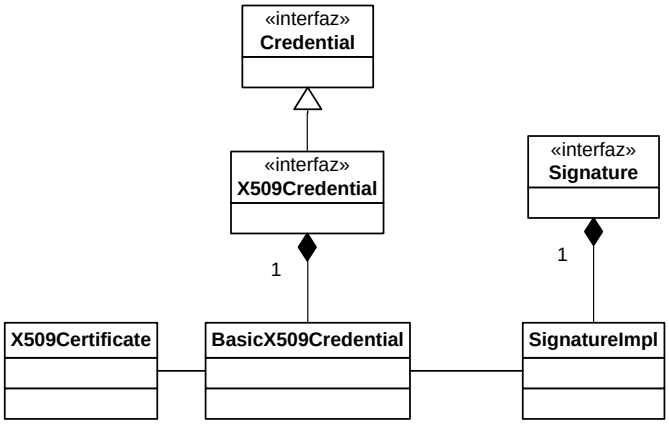


Figure 36 – OpenSAML Class Diagram for XML signature generation purposes

Next, the classes depicted in the figure above are briefly explained.

5.4.3.1.1 org.opensaml.xml.signature.Signature Interface

This Interface represents the XML signature to generate. Although it supports enveloped and detached signatures, only enveloped signatures must be generated.

5.4.3.1.2 org.opensaml.xml.signature.impl.SignatureImpl

This class (constructor protected) is instantiated by means of the *org.opensaml.xml.signature.impl.SignatureBuilder*.

5.4.3.1.3 org.opensaml.xml.security.credential.Credential Interface

This interface represents the credential material for an entity. In STORK, this credential will represent the asymmetric cryptographic information. Depending on the entity, the credential contains either the private and public keys (local entity) or just the public key (remote entity).

5.4.3.1.4 org.opensaml.xml.security.x509.X509Credential Interface

This interface is a particular view of the *Credential*. In STORK, it will represent an X.509 Certificate along with the private key.

5.4.3.1.5 org.opensaml.xml.security.x509.BasicX509Credential Class

This class is the implementation of the interface *org.opensaml.xml.security.x509.X509Credential*. This class manages an implementation of the JCE X.509 certificate.

5.4.3.1.6 java.security.cert.X509Certificate Class

This class is the JCE implementation of an X.509 certificate that wraps the public key for the verification of digital signatures.

5.4.4 Basic Class Diagram (XML Signature verification process)

OpenSAML provides several ways of performing an XML signature validation incorporated in a SAML token. The method based on trust engine offers both the cryptographic verification of the signature and the trust establishment of the verification credential. Therefore, this method has been chosen from the SAML core.

Next Figure depicts the most important classes from OpenSAML that must be used by the PEPS implementation, and in particular, by the PEPS Authentication Module, in order to verify the XML signature of a SAML Token according to the trust engine approach.

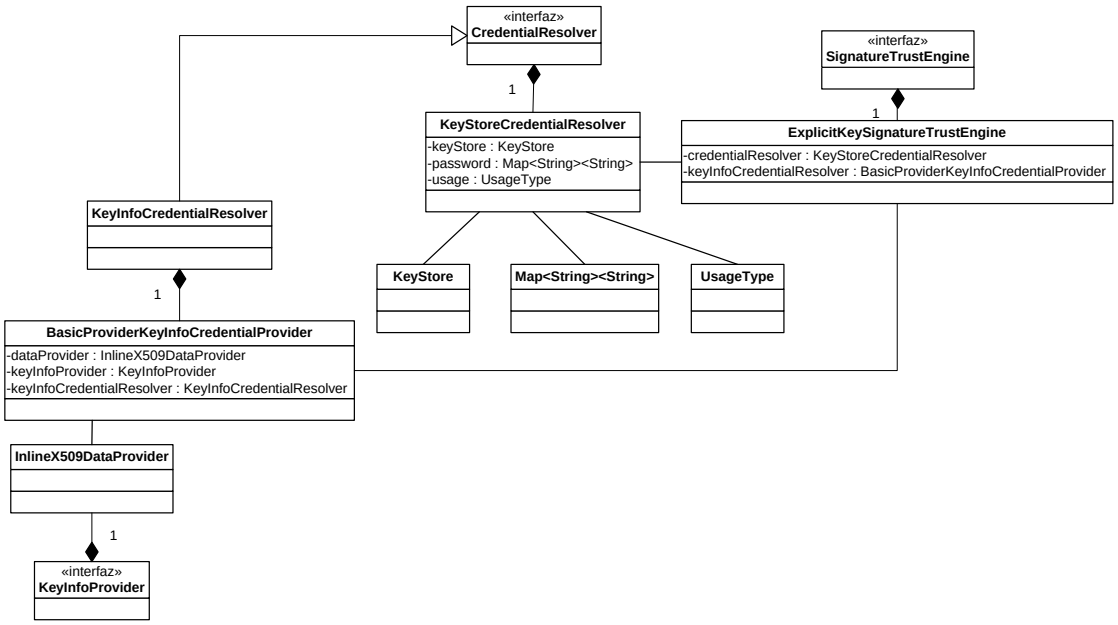


Figure 37 – OpenSAML Class Diagram for XML signature verification purposes

Next, the classes depicted in the figure above are briefly explained.

5.4.4.1.1 org.opensaml.xml.signature.SignatureTrustEngine Interface

This Interface represents the functionality to evaluate the trustworthiness and validity of XML or raw Signatures against implementation-specific requirements.

5.4.4.1.2 org.opensaml.xml.signature.impl.ExplicitKeySignatureTrustEngine Class

This class implements the interface *SignatureTrustEngine*. Two instances must be indicated when invoking the constructor of this class: *BasicProviderKeyInfoCredentialProvider* and *KeyStoreCredentialResolver*.

5.4.4.1.3 org.opensaml.xml.security.keyinfo.BasicProviderKeyInfoCredentialProvider Class

This class implements the interface *org.opensaml.xml.security.keyinfo.KeyInfoCredentialResolver*

A *KeyInfoCredentialResolver* allows the signature trust engine to retrieve the credential information from the KeyInfo material contained in the SAML signature. OpenSAML offers several implementations of key info credential resolver, among which this class has been selected.

BasicProviderKeyInfoCredentialProvider extracts the public key information from the <ds:KeyInfo> element contained in the XML signature to verify the digital signature. This resolver needs a list of *org.opensaml.xml.security.keyinfo.KeyInfoProvider* implementing providers in order to be able to search and retrieve the credential material from the XML signature.

In particular, STORK interfaces [Interfaces] define that the XML signature must contain the <ds:X509Certificate> embedded in a <ds:X509Data> element contained in <ds:KeyInfo>. As a result, this credential provider will need to obtain the public key from the X509Certificate. An instance of *InlineX509DataProvider* must be provided to the constructor of this class.

5.4.4.1.4 org.opensaml.xml.security.keyinfo.provider.InlineX509DataProvider Class

This class implements the *org.opensaml.xml.security.keyinfo.KeyInfoProvider* interface.

This provider is used by *BasicProviderKeyInfoCredentialProvider* to obtain the public key from the <ds:X509Certificate> information.

5.4.4.1.5 org.opensaml.xml.security.credential.KeyStoreCredentialResolver Class

Besides verifying the digital signature, the certificate that wraps the public key must be trusted by the verifier in order to give complete validity to the XML signature.

This class evaluates if the public key (certificate) is contained in the configured trusted key store (class *java.security.KeyStore*). The credentials to access the key store must be provided in a *java.util.Map* implementing class (e.g. *java.util.HashMap*). It must use the *STORKTrustedKeyStore* keystore to verify if the certificate is trusted or not.

Additionally, a key usage constraint can be indicated as well (*org.opensaml.xml.security.credential.UsageType*). The objective is to reject keys used to sign the SAML token that do not comply with the key usages defined for the PEPSs' certificates (see [11]).

OpenSAML only supports three types of key usages: ENCRYPTION, SIGNING and UNSPECIFIED. For that reason, *UsageType* SIGNING must be indicated during the instantiation of this class.

5.4.5 Methods

Methods	***SamlEngine***
Description	Interface for SPDocumentservie. **Methods**: • Generate stork attribute query request. o eu.stork.peps.auth.commons.STORKAttrQueryRequest **generateSTORKAttrQueryRequest**(eu.stork.peps.auth.commons.STORKAttrQueryRequest request) • Generate stork attribute query response. o eu.stork.peps.auth.commons.STORKAttrQueryResponse **generateSTORKAttrQueryResponse**(eu.stork.peps.auth.commons.STORKAttrQueryRequest request, eu.stork.peps.auth.commons.STORKAttrQueryResponse responseAttrQueryRes, String ipAddress, String destinationUrl, boolean isHashing) • Generate stork attribute query response fail. o eu.stork.peps.auth.commons.STORKAttrQueryResponse **generateSTORKAttrQueryResponseFail**(eu.stork.peps.auth.commons.STORKAttrQueryRequest request, eu.stork.peps.auth.commons.STORKAttrQueryResponse response, String ipAddress, String destinationUrl, boolean isHashing) • Generate stork attribute query response from multiple assertions o eu.stork.peps.auth.commons.STORKAttrQueryResponse **generateSTORKAttrQueryResponseWithAssertions**(eu.stork.peps.auth.commons.STORKAttrQueryRequest

Methods	***SamlEngine***
	request, eu.stork.peps.auth.commons.STORKAttrQueryResponse responseAttrQueryRes, List<eu.stork.peps.auth.commons.STORKAttrQueryResponse > responses, String ipAddress, String destinationUrl, boolean isHashing) • Generate stork authentication request. o eu.stork.peps.auth.commons.STORKAuthnRequest **generateSTORKAuthnRequest**(eu.stork.peps.auth.commons.STORKAuthnRequest request) • Generate stork authentication response. o eu.stork.peps.auth.commons.STORKAuthnResponse **generateSTORKAuthnResponse**(eu.stork.peps.auth.commons.STORKAuthnRequest request, eu.stork.peps.auth.commons.STORKAuthnResponse responseAuthReq, String ipAddress, boolean isHashing) • Generate stork authentication response. o eu.stork.peps.auth.commons.STORKAuthnResponse **generateSTORKAuthnResponseAfterQuery**(eu.stork.peps.auth.commons.STORKAuthnRequest request, eu.stork.peps.auth.commons.STORKAuthnResponse responseAuthReq, String ipAddress, boolean isHashing, List<eu.stork.peps.auth.commons.STORKAttrQueryResponse > res) • Generate stork authentication response fail. o eu.stork.peps.auth.commons.STORKAuthnResponse **generateSTORKAuthnResponseFail**(eu.stork.peps.auth.commons.STORKAuthnRequest request, eu.stork.peps.auth.commons.STORKAuthnResponse response, String ipAddress, boolean isHashing) • Generate stork logout request. o eu.stork.peps.auth.commons.STORKLogoutRequest **generateSTORKLogoutRequest**(eu.stork.peps.auth.commons.STORKLogoutRequest request) • Generate stork logout response. o eu.stork.peps.auth.commons.STORKLogoutResponse **generateSTORKLogoutResponse**(eu.stork.peps.auth.commons.STORKLogoutRequest request, eu.stork.peps.auth.commons.STORKLogoutResponse response) • Generate failed stork logout response. o eu.stork.peps.auth.commons.STORKLogoutResponse **generateSTORKLogoutResponseFail**(eu.stork.peps.auth.commons.STORKLogoutRequest request, eu.stork.peps.auth.commons.STORKLogoutResponse response) • Gets the single instance of STORKSAMLEngine. o static STORKSAMLEngine **getInstance**(String nameInstance) • Validate stork attribute query request.

Methods	***SamlEngine***
	○ eu.stork.peps.auth.commons.STORKAttrQueryRequest **validateSTORKAttrQueryRequest**(byte[] tokenSaml) • Validate stork attribute query response. ○ eu.stork.peps.auth.commons.STORKAttrQueryResponse **validateSTORKAttrQueryResponse**(byte[] tokenSaml, String userIP) • Validate stork authentication request. ○ eu.stork.peps.auth.commons.STORKAuthnRequest **validateSTORKAuthnRequest**(byte[] tokenSaml) • Validate stork authentication response. ○ eu.stork.peps.auth.commons.STORKAuthnResponse **validateSTORKAuthnResponse**(byte[] tokenSaml, String userIP) • Validate stork authentication response. ○ eu.stork.peps.auth.commons.STORKAuthnResponse **validateSTORKAuthnResponseWithQuery**(byte[] tokenSaml, String userIP) • Validate stork logout request. ○ eu.stork.peps.auth.commons.STORKLogoutRequest **validateSTORKLogoutRequest**(byte[] tokenSaml) • Validate stork logout response. ○ eu.stork.peps.auth.commons.STORKLogoutResponse **validateSTORKLogoutResponse**(byte[] tokenSaml)

Table 61: SAML Component interfaces

5.4.6 Keystore Management

Keystores have to be used by both the Authentication Engine and the Validation Engine in order to generate and validate the electronic signatures of SAML Tokens and OCSP tokens, respectively. This section offers the class design of the components that deal with keystore management.

Package: eu.stork.peps.keystores

This subsection gives an overview of the classes that support the PEPSP SAML Engine for the verification and generation SAML Tokens XML Signatures.

5.4.7 Basic Class Diagram

Next Figure outlines the classes that represent the static view of the KeyStore Management.

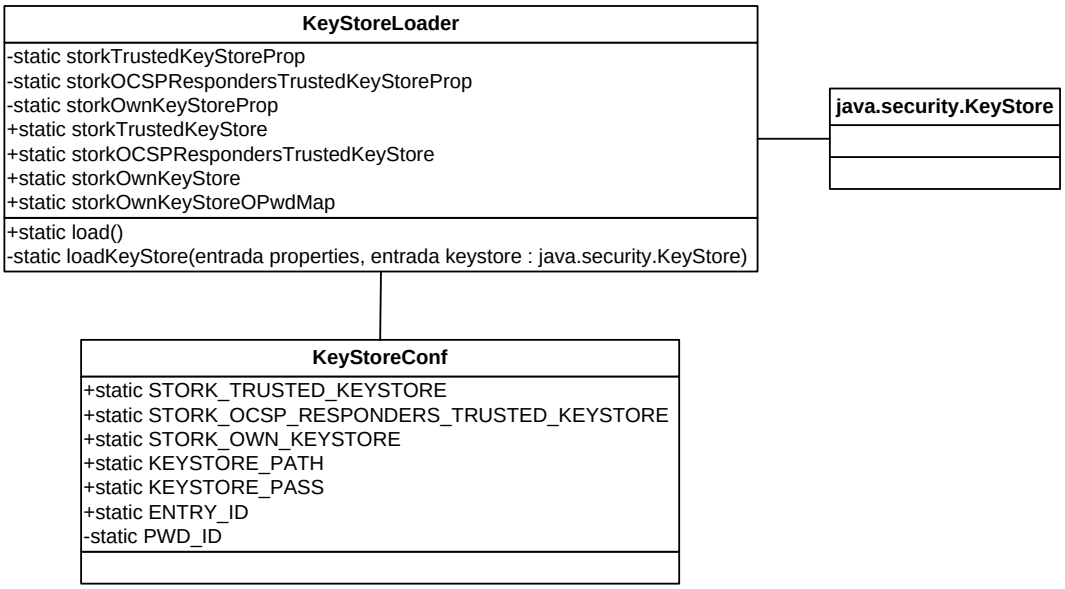


Figure 38 – KeyStore Management Classes

5.4.7.1.1 eu.stork.peps.keystoresKeyStoreLoader Class

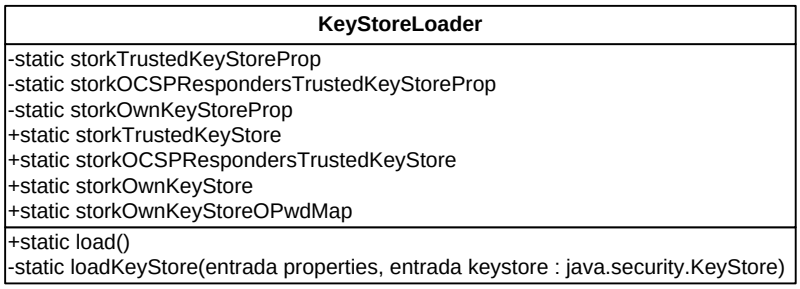


Figure 39 – KeyStoreLoader Class

This class loads in memory the information of the keystores. Thereby, cryptographic operations like XML signature/OCSP signature generation, XML signature/OCSP signature verification can be performed.

Attributes

1. private storkTrustedKeyStoreProp

   Properties of the STORKTrustedKeyStore keystore.

2. private storkOCSPRespondersTrustedKeyStoreProp

   Properties of the STORKOCSPRespondersTrustedKeyStore keystore.

3. private storkOwnKeyStoreProp

   Properties of the STORKOwnKeyStore keystore.

4. private storkTrustedKeyStore

   Keystore STORKTrustedKeyStore

5. private storkOCSPRespondersTrustedKeyStore

   Keystore STORKOCSPRespondersTrustedKeyStore

6. private storkOwnKeyStore

Keystore STORKOwnKeyStore

7. private storkOwnKeyStorePwdMap

   Hashmap with the password for every key entry in the keystore STORKOwnKeyStore

Methods

1. public static load

   Static method that loads all the information from the keystores and fills in the attributes described above.

2. private static loadKeyStore

   Auxiliary method to load each keystore information in memory.

5.4.7.1.2 eu.stork.peps.keystoresKeyStoreConf Class

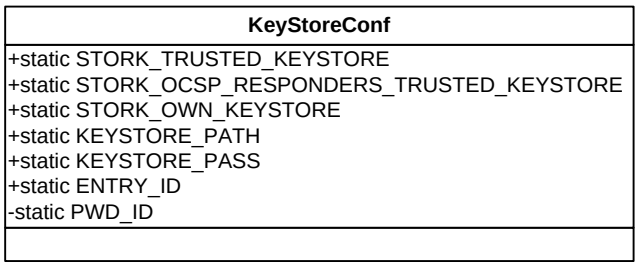


Figure 40 – KeyStoreConf Class

This class contains certain values used by *KeyStoreLoader* class for the keystores loading.

5.5 Digital Signatures

5.5.1 Description

The reference signature solution consists of an OASIS-DSS module that integrates the implementation of a signature service. A signature service implements the SPI interface.

In the course of STORK 2.0 developments for the purpose of facilitating signature integration among MSes, two different SPI implementations have been provided:

1) the reference SPI implementation that uses SD-DSS Signature applet, and

2) an Austrian SPI implementation using MS-specific services for creating digital signatures (Mocca and the Austrian Mobile Phone Signature).

The reference implementation consists of three parts, the SPI implementation added to the OASIS-DSS module, a web service for each SPI implementation and the SD-DSS signature applet. The reference implementation is based on SD-DSS version 4.1.0[13] .

The integration of the signature functionality with the PEPS is considered MS-specific. In order to provide a sample integration, additional signAP module and the accompanying code are provided. The code added to eu.stork.peps.auth.dtl.DTLPepsUtil.java class within the PEPS makes sure that the SignRequest and the document to be signed are transmitted from the PEPS to the OASIS-DSS module. The signAP then provides the component responsible for forwarding the user to the signing solutions (via the OASIS-DSS module) and returning the SignResponse as signedDoc attribute value back to the PEPS.

[13] https://joinup.ec.europa.eu/asset/sd-dss/description

The following subsections provide an overview of the packages integrated in the modules of reference signature solution. OASIS-DSS and OASIS-DSS-API modules provide the core of this solution. The common SOAP-client and STORK-database modules provide the supporting packages for the integration into practical MS-specific solution. However the MSes are encouraged to rely on their own implementations that suit their particular needs, infrastructure and environment. The two reference implementations delivered in this work also depend on them. Finally, the signAP module enables the integration of the digital signature functionality with the PEPS.

5.5.2 Packages in OASIS-DSS-API

Package name	***Description***
eu.stork.oasisdss.api	This package contains the classes that define signature and wrapping types and profiles. It furthermore contains the classes for handling, processing, marshalling and unmarshaling of oasis-dss data.
eu.stork.oasisdss.api.exceptions	This package consists of the classes for the handling of oasis-dss and processing specific exceptions.
eu.stork.oasisdss.api.utils	This package provides helper classes used for mapping and extracting data, as well as for the invocation of clients and services.
eu.stork.oasisdss.profile	This package contains the automatically generated classes supporting OASIS Stork profile, obtained by applying the schemas present in resources/schema/oasis-dss package.
eu.stork.signature.spi	This package includes the interface definition of SPI.

Table 62: DSS-API packages

5.5.3 Packages in OASIS-DSS module

Package name	***Description***
at.gv.egiz.bku.viewer	This package contains the definition of a validator interface applicable for various mime types, as well as its helper classes.
at.gv.egiz.bku.text	This package contains the implementation of text validator.
at.gv.egiz.bku.slxhtml	Contains the implementation of SLXHTML validator and supporting classes.
eu.stork.oasis.caching	This package includes the definition of caching provider for OASIS-DSS requests, as well as its in-memory based implementation and supporting classes.
eu.stork.oasis.exceptions	This package contains the exception classes from

Package name	*Description*
	the module.
eu.stork.oasisdss	In this package defined is the entry point for OASIS-DSS web service interface, integration servlet and OASIS DSS webform binding processing engine.
eu.stork.oasisdss.processing	This package provides various handlers used by OASIS-DSS processing engine.
eu.stork.oasisdss.redirectors	Contains definition interface and implementation for the targetLocator service for Oasis-DSS requests. It also contains SPITarget implementation that is responsible for forwarding the requests to other ISignature implementations.
eu.stork.oasisdss.utils	This package contains utility classes used in the package.
eu.stork.oasisdss.webform	This package contains the implementation of the registry of pending requests provided using webform binding.

Table 63: DSS module packages

5.5.4 Packages in reference SPI implementation using SD-DSS applet

The reference implementation based on SD-DSS signature applet consists of the following packages:

Package name	*Description*
eu.stork.signature.spi.impl.reference	This package contains the implementation of ISignature interface, as provided in OASIS-DSS API
eu.stork.signature.spi.impl.reference.config	Classes managing initialization of configuration parameters
Eu.stork.signature.referenceImplementation	This package contains servlet that handles signature requests.
Eu.stork.signature.spi.impl.reference.config	Contains configuration initialization classes.

Table 64: Reference implementation packages integrating SD-DSS

This module includes additional packages provided by third-party SD-DSS implementation. The details on these packages can be obtained from the developer's repository[14].

[14] https://joinup.ec.europa.eu/asset/sd-dss/description

5.5.5 Packages in reference SPI implementation using Austrian services

Package name	*Description*
eu.stork.signature.app.service	This package contains the implementation of ISignature interface, as provided in OASIS-DSS API
eu.stork.signature	Contains the interface definition for signature types and implementation helpers supporting PAdES and XAdES profiles.
eu.stork.signature.web	Contains the implementations of login, dataurl and signature servlets, as well as support for application configurations.
eu.stork.securitylayer	This package and its subpackages contain helper classes for the mapping of SignRequests to AT-specific SecurityLayer requests.
eu.stork.securitylayer.identitylink	Contains helper class for processing of InfoBoxReadRequests.
eu.stork.securitylayer.signature	Contains the classes for handling and processing of signature requests and responses.

Table 65: Reference implementation packages integrating MS services

5.5.6 Packages in the common SOAP-client module

Package name	*Description*
at.gv.e_government.reference.namespace.verificationservice._20120922	Axis-derived classes providing the support for AT eGov namespace
eu.stork.signature.verification.soap.client	Contains the implementation of sample SOAP client
org.w3._2000._09.xmldsig_	Contains auto-generate classes

Table 66: Common SOAP-client packages

5.5.7 Packages in the common STORK-database module

Package name	*Description*
eu.stork.signature.database	Provides entities that represent SignRequest including request metadata and document that should be cache, along with its particular metadata. It furthermore provides the transaction implementation using database.
eu.stork.signature.database.exceptions	Contains the exception clases used in the module

Table 67: Packages in common STORK-database module

5.5.8 Packages in the SignAP module

This module is an extended version of a the DemoAP module. Therefore, the most of the descriptions in DemoAP apply to this module and packages as well.

The most relevant additional functionalities provided by this module are the following:

- Extraction of SignRequest from the PersonalAttributeList
- Communication with the OASIS module via Web Form Binding
- Temporary storage of the incoming SAMLRequest, to enable later integration and correlation with the SignResponse
- Extraction of the SignResponse returned by the OASIS module and its integration into STORKAttrQueryResponse
- Delivery of STORKAttrQueryResponse to APResponseURL

Package name	***Description***
eu.stork.ap	This package contains the interfaces for managing of incoming requests and working with SAML objects and their implementations.
eu.stork.ap.actions	This package contains action handlers. The main actions that supports the integration with OASIS are provided in StartSignatureCreationAction and ObtainSignatureAction classes.
eu.stork.ap.exceptions	Contains the exceptions used in the module.
eu.stork.ap.idp	This package integrates the actions that support authentication.
eu.stork.ap.security	This package contains the interceptor class used to filter and validate incoming requests.
eu.stork.ap.storage	This package defines the interface and provides the implementation for the temporary storage of SAML requests, used to enable the maping after the signature has been returned by OASIS module.

Table 68: SignAP packages

5.6 Document Transfer Layer (DTL)

5.6.1 Description

The document transfer layer (DTL) handles the transfer of signature request (signDoc) between SP, S-PEPS, C-PEPS and member specific DSS (document signature service). The DTL is a JAX webservice with a mySQL database and WSDL interface for transfer of documents

The DTL is designed as a standalone web service to communicate with PEPS and other DTL for transfer of signature documents between Member States. The process has two phases, 1) The user and the document is transferred to the DSS of the user's home country, where the user signes the document. 2) The user and the signed document is signed data is returned to the SP.

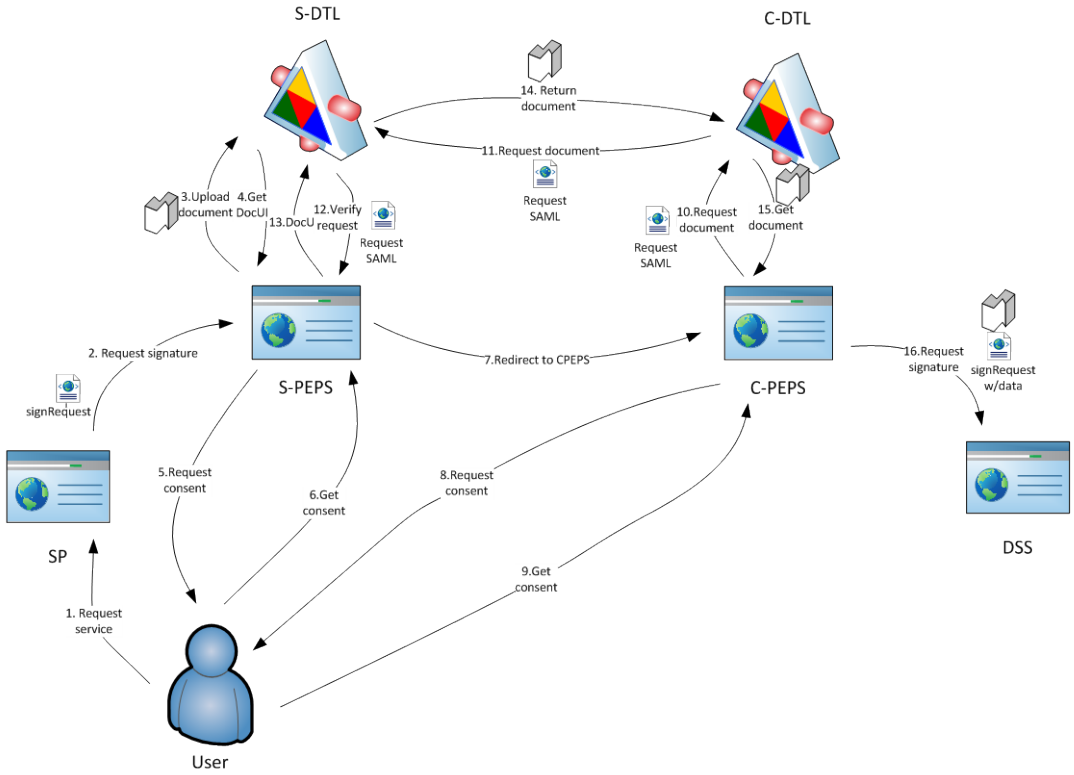


Figure 41 Signature request transferred from SP to country DSS

Phase one has the following steps.

1. User requests service from SP (i.e. bank)
2. SP redirects user to S-PEPS with an Oasis signRequest for the document to be signed.
3. S-PEPS process signRequest and uploads document to his S-DTL.
4. S-DTL stores document and return a document ID (DocUI)
5. S-PEPS requests users consent for action
6. User gives his consent
7. S-PEPS redirects user to C-PEPS
8. C-PEPS requests users consent for action
9. User gives his consent
10. C-PEPS creats a document transfer request SAML and sends to his C-DTL
11. C-DTL requests document from S-DTL with the document request SAML
12. S-DTL ask S-PEPS to validate request SAML
13. S-PEPS validates request and returns DocUI
14. S-DTL return document to C-DTL
15. C-DTL returns document and mime type to C-PEPS
16. C-PEPS forwards user to his DSS for signature creation

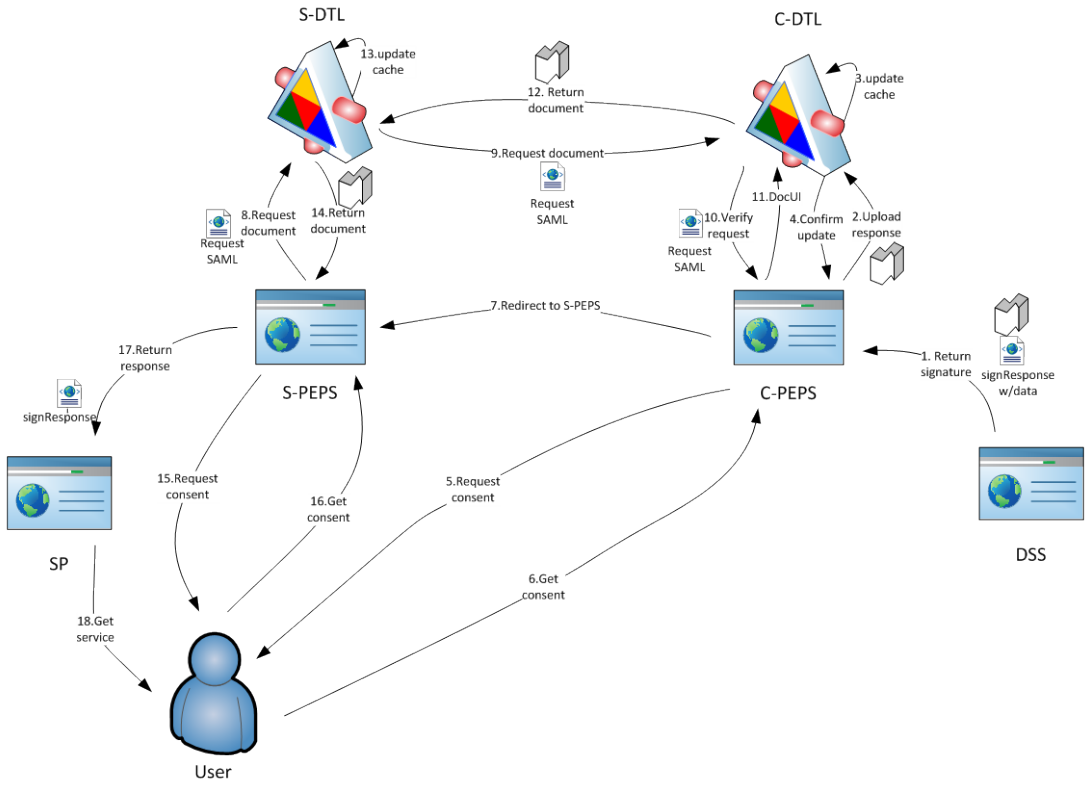


Figure 42 Signature response is returned to SP from DSS

Phase two has the following steps:

1. DSS returns signature response to C-PEPS
2. C-PEPS uploads response to C-DTL
3. C-DTL updates cache
4. C-DTL confirms update
5. C-PEPS requests consent from user
6. User gives his consent
7. C-PEPS redirect user to S-PEPS
8. S-PEPS creats a document transfer request SAML and sends to his S-DTL
9. S-DTL requests document from C-DTL with the document request SAML
10. C-DTL ask C-PEPS to validate request SAML
11. C-PEPS validates request and returns DocUI
12. C-DTL return document to S-DTL
13. S-DTL updates cache
14. S- DTL returns document and mime type to S-PEPS
15. S-PEPS requests users consent for action
16. User gives his consent
17. S-PEPS returns signature response to SP

5.6.2 Packages

Following are the main classes in the DTL and a short description of the functionality.

5.6.2.1.1 Documentservice

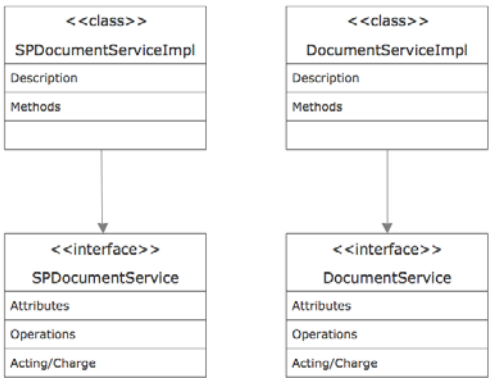


Figure 43 Class diagram for Documentservice

5.6.2.1.1.1 Class description

Here is a description of the main classes involved in the Documentservice package.

Interface Class	***DocumentService***
Description	Implements the interface of DocumentService. **Methods**: • Add document to DTL layer o String **addDocument**(byte[] document, String xmlRequest, String country, String SpId, String mimeType, String receiverCert) • Add document to DTL layer which SP has uploaded o String **addSPDocument**(String docId, String xmlRequest, String country, String SpId, String receiverCert) • Get document from DTL o byte[] **getDocument**(String documentTransferRequest, String dtlUrl) • Get document mime type of document o String **getDocumentMime**(String docId, String dtlUrl) • Update document in dtl o Boolean **updateDocument**(String docId, String xmlResponse, byte[] document) • Update document in DTL and prepare for SP o boolean **updateSPDocument**(String documentTransferRequest, String dtlUrl, String xmlResponse)

Table 69: Main classes in the Document service package

Interface Class	***SPDocumentService***
Description	Interface for SPDocumentservice. **Methods**: • Add document to temp layer o String **addSPSignDocument**(byte[] document, String spId, String mimeType, String receiverCert) • Get document from Temp layer o byte[] **getSPDocument**(String docId, String spId)

Table 70: Methods in the SPDocumentService interface

Interface Class	***DocumentServiceImpl***
Description	Implements that functionality for the document service. **Methods**: • Add document to DTL layer o String **addDocument**(byte[] document, String xmlRequest, String destinationCountry, String SpId, String mimeType, String receiverCert) • Add document to DTL layer which SP has uploaded o String **addSPDocument**(String docId, String xmlRequest, String destinationCountry, String SpId, String receiverCert) • Get document from DTL o byte[] **getDocument**(String documentTransferRequest, String dtlUrl) • Get document mime type of document o String **getDocumentMime**(String docId, String dtlUrl) • Update document in dtl o boolean **updateDocument**(String docId, String xmlResponse, byte[] document) • Update document in DTL and prepare for SP o boolean **updateSPDocument**(String documentTransferRequest, String dtlUrl, String xmlResponse)

Table 71: Methods in the DocumentServiceImpl interface

Interface Class	***SPDocumentServiceImpl***
Description	Implements the actual interaction with documents in database **Methods**: • Add document to temp layer o String **addSPSignDocument**(byte[] document, String SpId,

Interface Class	***SPDocumentServiceImpl***
	String mimeType, String receiverCert) • Get document from Temp layer o byte[] **getSPDocument**(String docId, String spId)

Table 72: Methods in the SPDocumentServiceImpl interface

5.6.2.1.2 Documentservice.data

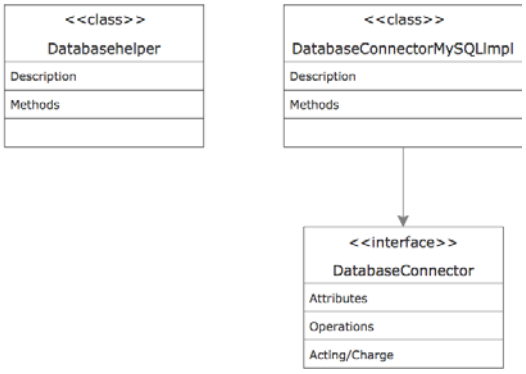


Figure 44 Class diagram for data

5.6.2.1.2.1 Class description

Here is a description of the main classes involved in the Documentservice data package.

Interface Class	***DatabaseConnector***
Description	Implements the interface for connection and communication with database. **Methods**: • Add document to database o boolean **addDocument**(DocumentModel document) • Add request to database o boolean **addRequest**(RequestModel request) • Add temp document to database o boolean **addTempDocument**(TempDocumentModel document) • Delete Document from database o boolean **deleteDocument**(String docId) • Delete temp document from database o boolean **deleteTempDocument**(String docId) • Get Document from database o DocumentModel **getDocument**(String docId) • Get request from database o RequestModel **getRequest**(String requestId) • Get request from database

Interface Class	***DatabaseConnector***
	○ RequestModel **getRequestByDocId**(String docId) • Get temp document from database ○ TempDocumentModel **getTempDocument**(String docId) • Update document in database ○ boolean **updateDocument**(DocumentModel document) • Update request in database ○ boolean **updateRequest**(RequestModel request) • Update temp document in database ○ boolean **updateTempDocument**(TempDocumentModel document)

Table 73: Methods in the DatabaseConnector interface

Interface Class	***DatabaseConnectorMySQLImpl***
Description	Implements the connection and communication with database. **Methods**: • Add document to database ○ boolean **addDocument**(DocumentModel document) • Add request to database ○ boolean **addRequest**(RequestModel request) • Add temp document to database ○ boolean **addTempDocument**(TempDocumentModel document) • Delete Document from database ○ boolean **deleteDocument**(String docId) • Delete temp document from database ○ boolean **deleteTempDocument**(String docId) • Get Document from database ○ DocumentModel **getDocument**(String docId) • Get request from database ○ RequestModel **getRequest**(String requestId) • Get request from database ○ RequestModel **getRequestByDocId**(String docId) • Get temp document from database ○ TempDocumentModel **getTempDocument**(String docId) • Update document in database ○ boolean **updateDocument**(DocumentModel document) • Update request in database

Interface Class	***DatabaseConnectorMySQLImpl***
	○ boolean **updateRequest**(RequestModel request) • Update temp document in database ○ boolean **updateTempDocument**(TempDocumentModel document)

Table 74: Methods in the DatabaseConnectorMySQLImpl interface

Interface Class	***DatabaseHelper***
Description	Helper class for database implementation. **Methods**: • Get the current timestamp in SQL timestamp ○ static Timestamp **getSqlCurrentDate**() • Convert Java Date to SQL timestamp ○ static Timestamp **getSqlDate**(Date date) • Convert SQL timestamp to Java Date ○ static Date **getUtilDate**(Timestamp time)

Table 75: Methods in the DatabaseHelper interface

5.6.2.1.3 Documentservice.model

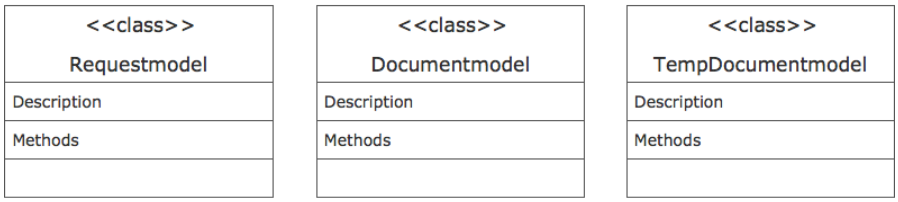


Figure 45 Class diagram for model

5.6.2.1.3.1 Class description

Here is a description of the main classes involved in the Documentservice model package.

Interface Class	***DocumentModel***
Description	Implements the gets and sets for the document model. **Methods**: Get the time of file creation Date getCreated() Get the file stream InputStream getDataStream()

Interface Class	***DocumentModel***
	Get the time when file was deleted Date **getDeleted**() Get the id of the document String **getDocid**() Get a byte array of the document byte[] **getDocument**() Get the encrypted String **getEnciv**() Get the encrypted key String getE**n**ckey() Get filename String **getFilename**() Get file mimetype String **getMimetype**() Get the reciver cert String **getReicevercert**() Get time of updated Date **getUpdated**() Validate before insert void **insertValidate**() Set the time of creation void **setCreated**(Date created) Set the file strea, void **setDataStream**(InputStream stream) Set time of deletion void **setDeleted**(Date deleted) Set document id void **setDocid**(String docid) set document void **setDocument**(byte[] document) Set encrypted void **setEnciv**(String enciv) Set encrypted key void **setEnckey**(String enckey)

Interface Class	***DocumentModel***
	Set filename void **setFilename**(String filename) Set mime type void **setMimetype**(String mimetype) Set reciever cert void **setReicevercert**(String reicevercert) Set time when file was last updated void **setUpdated**(Date updated) Validate data void **updateValidate**()

Table 76: Methods in the DocumentModel interface

Interface Class	***RequestModel***
Description	Implements get and sets for the request model **Methods**: Get the destination country String **getDestcountry**() Get the document id String **getDocid**() Get the full document id String **getFullDocID**() Get the request time Date **getReqtimestamp**() Get the request id String **getRequestid**() Get the time of request Date **getRestimestamp**() Get the service provider String **getSpcountry**() Get the service provider id String **getSpid**() Get the xml of the request String **getXmlrequest**()

Interface Class	***RequestModel***
	Get the xml of the response String **getXmlresponse**() Validate data void **insertValidate**() Set the destionation country void **setDestcountry**(String destcountry) Set the document id void **setDocid**(String docid) Set the request time void **setReqtimestamp**(Date reqtimestamp) Set the request id void **setRequestid**(String requestid) Set the time of request void **setRestimestamp**(Date restimestamp) Set service provider country void **setSpcountry**(String spcountry) Set service provider id void **setSpid**(String spid) Set the xml of the request void **setXmlrequest**(String xmlrequest) Set the xml of the response void **setXmlresponse**(String xmlresponse) Validate data void **updateValidate**()

Table 77: Methods in the RequestModel interface

Interface Class	***TempDocumentModel***
Description	Temp document model class **Methods**: Get the time of creation Date **getCreated**() Get the data stream InputStream **getDataStream**()

Interface Class	***TempDocumentModel***
	Get the time of deletion Date **getDeleted**() Return document id String **getDocid**() Return the file byte[] **getDocument**() Get the encrypted initialization vector String **getEnciv**() Get the encrypted key String **getEnckey**() Get the time of file access Date **getFetched**() Get the mime type of the file String **getMimetype**() Get the the reciever cert String **getReicevercert**() Return the service provider id String **getSpid**() Validate data void **insertValidate**() Set the time of creation void **setCreated**(Date created) Set the data stream void **setDataStream**(InputStream stream) Set the time of deletion void **setDeleted**(Date deleted) Set the document id void **setDocid**(String docid) Set the document void **setDocument**(byte[] document) Set the encrypted initialization vector void **setEnciv**(String enciv) Set the encrypted key void **setEnckey**(String enckey)

Interface Class	***TempDocumentModel***
	Set the time of last access void **setFetched**(Date fetched) Set the file mime type void **setMimetype**(String mimetype) Set the reciver cert void **setReicevercert**(String reicevercert) Set the service provider id void **setSpid**(String spid) Validate void **updateValidate**()

Table 78: Methods in the TempDocumentModel interface

5.6.2.1.4 Documentservice.utils

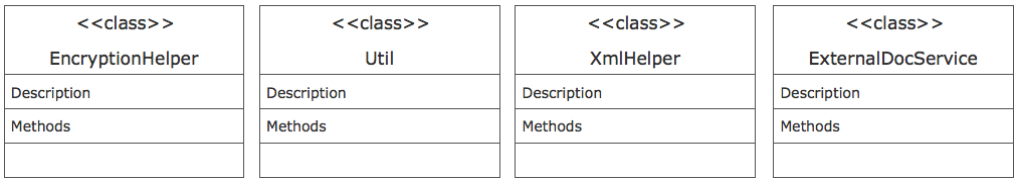


Figure 46 Class diagram for Utils

5.6.2.1.4.1 Class description

Here is a description of the main classes involved in the Documentservice utils package.

Interface Class	***EncryptionHelper***
Description	Utility class with encryption functionality: **Methods**: Decrypt data with keys byte[] **decrypt**(byte[] encData) Encrypt data with key byte[] **encrypt**(byte[] clearData) Encrypt string with certificate String **encryptWithCert**(String certString, String input) Generate new symmetric keys void **generateKeys**() Get the IV string

Interface Class	***EncryptionHelper***
	String **getIv**() Get the key string String **getKey**() Initialize keys with specified keys void **initKeys**(String inKey, String inIv)

Table 79: Methods in the EncryptionHelper interface

Interface Class	***ExternalDocservice***
Description	Utility class for file handling **Methods**: Get document from external DTL static byte[] **getDocument**(String documentTransferRequest, String dtlUrl) Get document mime from external DTL static String **getDocumentMime**(String docId, String dtlUrl)

Table 80: Methods in the ExternalDocservice interface

Interface Class	***Utils***
Description	General utility class for DTL **Methods**: Decode base64 string to bytes static byte[] **decodeBase64String**(String base64string, boolean urlSave) Base64 encode bytes static String **encodeBase64bytes**(byte[] bytes, boolean urlSafe) Get string stream static InputStream **getStream**(String string, String codePage) Get file data static byte[] **readData**(String fileName) Read the content of the file static String **readString**(String fileName) Send using GET static String **sendGet**(String url)

Interface Class	***Utils***
	Send using POST static String **sendPost**(String url, String urlParameters)

Table 81: Methods in the Utils interface

Interface Class	***XmlHelper***
Description	Utility class for xml handling **Methods**: Get request document static String **getRequestDocument**(String xmlRequest) Get request document data static String **getRequestDocumentData**(String xmlRequest) Get mime from request static String **getRequestDocumentMime**(String xmlRequest) Get request id from request static String **getRequestId**(String xmlRequest) String the document id static String **StripDocId**(String docId) Verify the transfer request static String **verifyRequest**(String transferRequest) Verify request bytes static String **verifyRequestByte**(byte[] transferRequest)

Table 82: Methods in the XmlHelper interface

5.6.2.1.5 Documentservice.exceptions

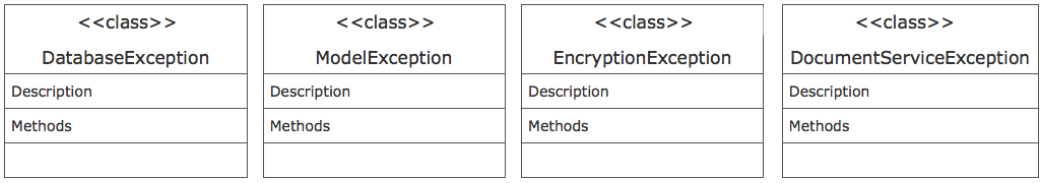


Figure 47 Class diagram for exceptions

5.6.2.1.5.1 Class description

Here is a short description of the main classes involved in the Documentservice exception package.

Interface	***DatabaseException***
Class	***DocumentServiceException***
	EncryptionException
	ModelException
Description	Implements exception handling for DTL classes.

Table 83: Classes in the Exceptions interfaces

5.6.3 Webservice

The DTL service has the following methods and is no dependency on other STORK components.

5.6.3.1.1 Interface

```
// Web method to upload a document to server
@WebMethod(operationName = "addDocument")
public String addDocument(byte[] document, String xmlRequest, String
country, String SpId, String mimeType, String receiverCert);

// web method to download a document from server
@WebMethod(operationName = "getDocument")
public byte[] getDocument(String documentTransferRequest, String dtlUrl);

// web method to download a document from server
@WebMethod(operationName = "getDocumentMime")
public String getDocumentMime(String docId, String dtlUrl);

// web method to update document in server
@WebMethod(operationName = "updateDocument")
public boolean updateDocument(String docId, String xmlResponse, byte[]
document);
```

5.6.4 Database

The DTL has been programmed to connect to a mySQL database but it should be fairly trivial to use another database vendor. The create script for the database and tables is in the folder DBSQL in the DocumentService project.

5.6.4.1.1 Tables

The database consists of two tables, document and request which can be seen onFigure 3. The document tables holds data pertaining to the document data (the data its self, mimetype, created time, encryption data etc). The request table holds information about the oasis sign request and is linked to the document data by the document id.

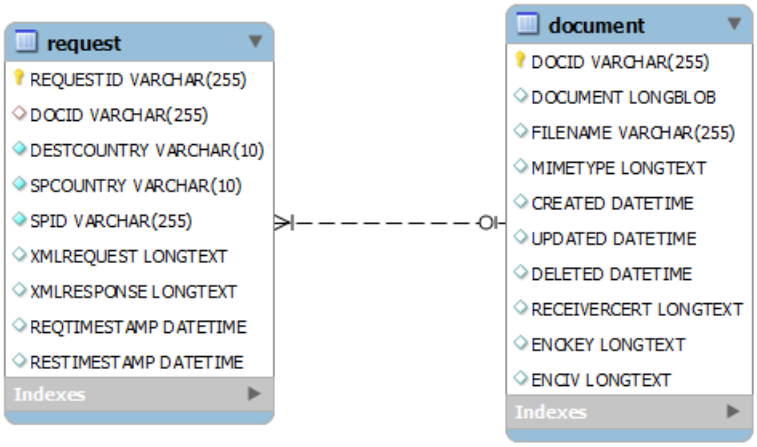


Figure 48 DTL database tables.

5.7 Version Control

5.7.1 Description

Version Control (VC) is the process of sharing the configuration information related to the STORK software and the environment.

This module includes two libraries:

- Version Control library → includes the version control functionalities.
- Updater library → schedules daily routine version control operations.

Three configuration files are existed specific to Version Control Module. First two of them are configurated for Version Control Library and the other is for Updater Library:

- versioninfo.properties configuration file isn't not specific to running environments (PROD, PRE_PROD, TEST, PEPS, SP).
- myinfo-*.xml configuration file is used when generating a version control file that is specific to running environment. * test|pre-prod|prod
- schedulerApplicationContex.xml is used for scheduling operation.

Other than these configurations, Version Control Module has configurations related with other projects (PEPS, SP and V-IDP) and store files(keystore, trust store).

The following diagram shows the main classes involved in the anonymity process.

5.7.2 Package specification

Next, the classes depicted above are briefly explained:

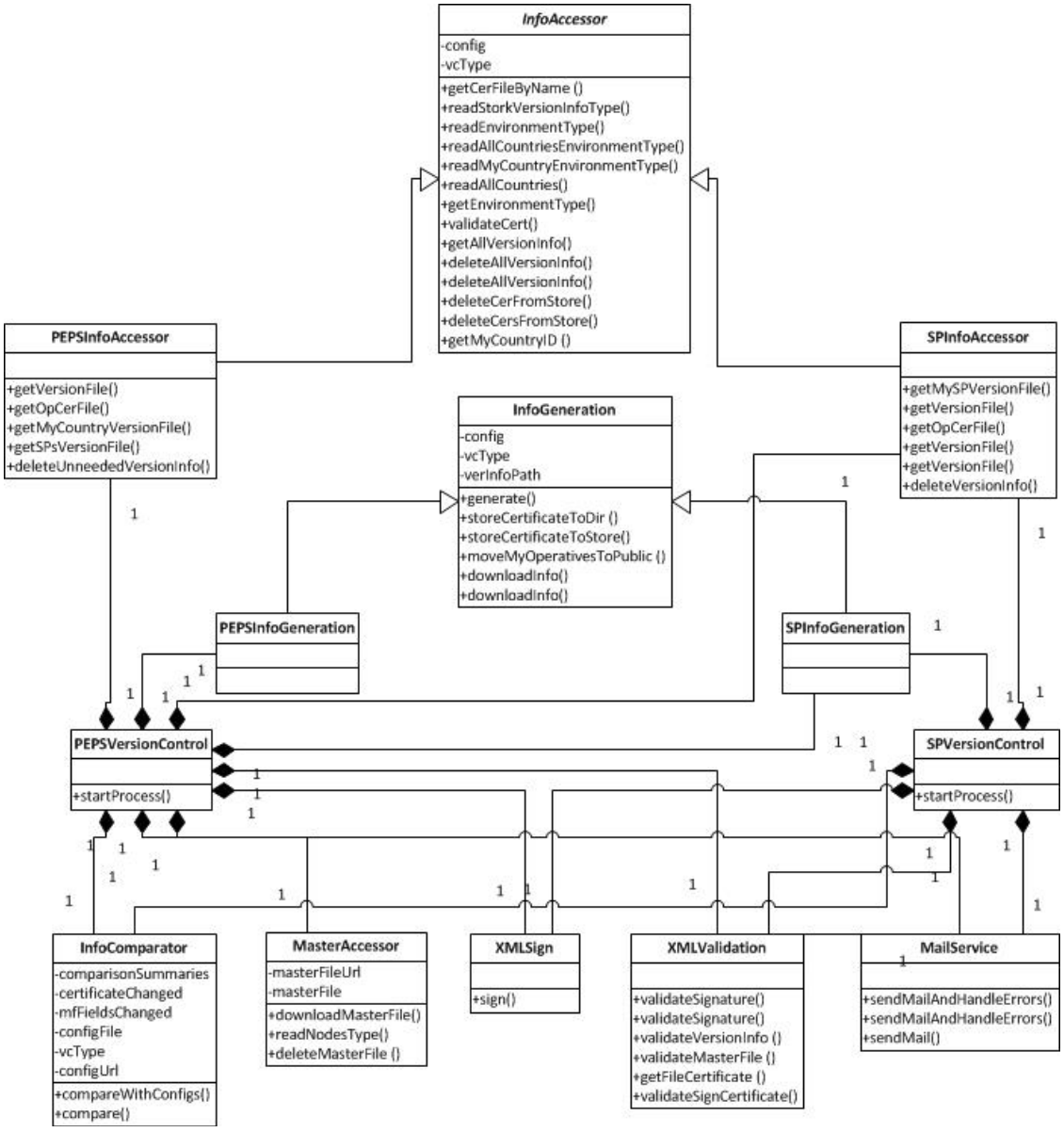


Figure 49 Class diagram for Version Control

Interface Class	***InfoGeneration***
Description	This is an abstract class for generating and downloading PEPS/SP version control files: • **config**: configuration file • **vcType**: version control file type • **verInfoPath**: version control file location
Methods	• **generate(InfoEnvironment environment)**: void Description: Abstract method for version control files generation. Input parameters: o **environment**: running environment

Interface Class	***InfoGeneration***
	• **storeCertificateToDir (byte[] cert, String cerFileName)**: void Description: Stores byte arrays as certificate file to local storage. o **cert**: certificate data as byte array o **cerFileName:** certificate file name • **storeCertificateToStore(byte[] cert, String cerFileName, VersionControlType vcSession)**: void Description: Store byte arrays as certificate file to trusted store o **cert**: certificate data as byte array o **cerFileName:** certificate file name o **vcSession:** which type certificate(PEPS |SP) • **moveMyOperativesToPublic ()**: void Description: Moves my file(s) to public directory. • **downloadInfo(Collection<String> urls)**: Collection<String> Description: This method downloads version control files to received directory according to given urls. o **urls**: given download urls Output returns: failed download urls • **downloadInfo(String urlStr)**: File Description: This method downloads version control file to received directory according to given url. o **url**: download url Output returns: downloaded file

Table 84: Interface of InfoGeneration class of Version Control

Interface Class	***PEPSInfoGeneration***
Description	Represents a class that generating and downloading PEPS version control file. This class contains PEPS specialized operations when generating version control files: • **infoAccessor:** PEPS version control files attributes accessor

Table 85: Interface of PEPSInfoGeneration class of Version Control

Interface Class	***SPInfoGeneration***
Description	Represents a class that generating and downloading SP version control file. This class contains SP specialized operations when generating version control files.

Table 86: Interface of SPInfoGeneration class of Version Control

Interface Class	***SamlXMLParser***
Description	Parses SamlEngine.xml and return trusted store path.
Methods	• **getKeystoreConfPath(final String samlEngineDir, final String instanceName)**: InputStream Description: Returns signer key store configuration path. Input parameters: o **samlEngineDir**: samlengine.xml file's directory path o **instanceName:** instance name in xml

Table 87: Interface of SamlXMLParser class of Version Control

Interface Class	***MailService***
Description	Class used for sending mails.
Methods	• **sendMailAndHandleErrors(String subject, String message, final URL configUrl)**: void Description: Sends a mail without any exception. • **sendMailAndHandleErrors(MailData mailData)**: void Description: Sends a mail with the given data contained mailData object. This method returns silently in all failure and success cases. It catches and logs all exceptions internally and does not rethrow them.This is because this method is intended to be used asynchronously either via jms or other asyncronous calls. • **sendMail(String subject, String message, URL configUrl)**: void Description: Sends a mail with the configuration data contained mailData object. Intended to be used in synchronous calls • **sendMail(MailData mailData)**: void Description: Sends a mail with the given data contained mailData object. Intended to be used synchronous calls

Table 88: Interface of MailService class of Version Control

Interface Class	***MasterAccessor***
Description	This class provides accessing interface to master file:

Interface Class	***MasterAccessor***
	• **masterFileUrl**: Master file url. • **masterFile**: Downloaded master file.
Methods	• **downloadMasterFile(final URL configUrl)**: void Description: Downloads master file from given url. Input parameters: o **configUrl**: master file configuration url. • **readNodesType():**void Description: Returns NodesType field of master file. • **deleteMasterFile ()**: boolean Description: Deletes master file. Output returns: deletion operation result

Table 89: Interface of MasterAccessor class of Version Control

Interface Class	***InfoAccessor***
Description	This abstract class provides accessing interface to version control files: • config: configuration • vcType: version control type
Methods	• **getCerFileByName (String fileName)**: File Description: Returns certificate that is given its name. Input parameters: o **fileName:** file name • **readStorkVersionInfoType (File xmlFile)**: StorkVersionInfoType Description: Returns StorkVersionInfoType field of version control file. Input parameters: o **xmlFile:** version control file xml • **readEnvironmentType(File xmlFile)**: EnvironmentType Description: Returns EnvironmentType field of version control file. Input parameters: o **xmlFile:** version control file xml • **readAllCountriesEnvironmentType(File xmlFile)**: Map<String,EnvironmentType> Description: Returns MyCountry's EnvironmentType field of version control file. This can be SPs file.

Interface Class	*InfoAccessor*
	Input parameters: o **xmlFile:** version control file xml • **readMyCountryEnvironmentType(File xmlFile)**: EnvironmentType Description: Returns MyCountry's EnvironmentType field of version control file. This can be SPs file. Input parameters: o **xmlName:** version control file xml • **readAllCountries(File xmlFile)**: File Description: Returns countries. This can be SPs file. Input parameters: o **xmlFile:** version control file xml • **getEnvironmentType(EnvironmentsType envsType)**: EnvironmentType Description: Reads country types. Input parameters: o **envsType:** environment type field of xml • **validateCert(File xmlFile, EnvironmentType envType)**: boolean Description: Compares inner and outer certificate's equality. Input parameters: o **xmlFile:** version control file xml o **envType:** environment type field of xml • **getAllVersionInfo(InfoDirType dirType):** File[] Description: Returns all version control files under given directory. Not exist SPs file. Input parameters: o **dirType:** operation directory type • **deleteAllVersionInfo(InfoDirType dirType):**void Description: Deletes all version control files under given directory. Input parameters: o **dirType:** operation directory type • **deleteAllVersionInfo(InfoDirType dirType, XMLGregorianCalendar dateBefore)**: void Description: Deletes all version control files under given directory that is generated older than given date. Input parameters: o **dateBefore:** date before o **dirType:** operation directory type

Interface Class	***InfoAccessor***
	• **deleteCerFromStore(String alias, VersionControlType vcSession)**: boolean Description: Deletes certificate from trust store with given alias Input parameters: o **alias:** trust store alias o **vcSession:** Version Control type • **deleteCersFromStore(Set<String> uniqueIDs, VersionControlType vcSession)**: boolean Description: Deletes certificates from trust store with given ids Input parameters: o **uniqueIDs:** version control file xml o **vcSession:** Version Control type • **getMyCountryID ()**: String Description: Returns own country id.

Table 90: Interface of InfoAccessor class of Version Control

Interface Class	***InfoComparator***
Description	This class provides to compare version control files: • **comparisonSummaries**: definitions of changes • **certificateChanged**: is certificate changed result • **mfFieldsChanged**: is master file changed result • **configFile**: configuration file • **vcType**: version control type • **configUrl**: configuration url
Methods	• **compareWithConfigs(final InfoEnvironment environment, final File xmlExist)**: boolean Description: Compares new version control file and configuration file values. If changes are existed, it will return true. Input parameters: o **environment**: running environment o **xmlExist:** version control file xml • **compare (File xmlOld, File xmlNew)**: int Description: Compares two version controls. Input parameters: o **xmlOld**: current version control file

Interface Class	***InfoComparator***
	o **xmlNew: upcoming version control file** Output Return: If equals return 0.

Table 91: Interface of InfoComparator class of Version Control

Interface Class	***PEPSInfoAccessor***
Description	This class provides accessing interface to PEPS version control files:
Methods	• **getVersionFile(String countryID, InfoDirType dirType)**: File Description: Returns any PEPS version control file. Input parameters: o **countryID**: country id o **dirType**: operation directory type • **getOpCerFile(String countryID)**: File Description: Return any PEPS operative certificate. Input parameters: o **countryID**: country id • **getMyCountryVersionFile(InfoDirType dirType)**: File Description: Returns own PEPS version control file. Input parameters: o **dirType:** operation directory type • **getSPsVersionFile(InfoDirType dirType)**: File Description: Returns own SPs version control file. Input parameters: o **dirType:** operation directory type • **deleteUnneededVersionInfo(Set<String> countryIDs, InfoDirType dirType)**: boolean Description: Deletes version control file that is removed from master file. Input parameters: o **countryIDs**: country ids o **dirType**: operation directory type

Table 92: Interface of PEPSInfoAccessor class of Version Control

Interface Class	***SPInfoAccessor***

Interface Class	***SPInfoAccessor***
Description	This class provides accessing interface to SP version control files:
Methods	• **getMySPVersionFile(InfoDirType dirType)**: File Description: Returns own SP version control file. Input parameters: o **dirType**: operation directory type • **getVersionFile(String countryID, String spID, InfoDirType dirType)**: File Description: Returns any SP version control file. Input parameters: o **countryID**: country id o **spID**: SP id o **dirType**: operation directory type • **getOpCerFile(String countryID)**: File Description: Return any SP operative certificate. Input parameters: o **countryID**: country id • **getVersionFile(String countryID, String spID, InfoDirType dirType, InfoEnvironment environment)**: StorkVersionInfoType Description: Returns version control file's info type. Input parameters: o **countryID**: country id o **spID**: SP id o **dirType**: operation directory type o **envType**: environment type • **getVersionFile(String countryID, String spID, InfoDirType dirType)**: EnvironmentType Description: Returns version control file's environment type. Input parameters: o **countryID**: country id o **spID**: SP id o **dirType**: operation directory type • **deleteVersionInfo(Set<String> countryIDs, InfoDirType dirType)**: boolean Description: Delete a SP version control file. Input parameters:

Interface Class	***SPInfoAccessor***
	o **countryIDs**: country ids o **dirType**: operation directory type

Table 93: Interface of SPInfoAccessor class of Version Control

Interface Class	***XMLSign***
Description	This class signs version control xml file:
Methods	• **sign(File xmlfile,String keystorePath,String keyStorePassword,String keyPassword)**: void Description: Signs a xml file by stored key. Input parameters: o **xmlfile**: version control file xml o **keystorePath**: keystore path o **keyStorePassword**: keystore password o **keyPassword**: key password • **sign(Document nodoOriginal,PrivateKey privateKey, X509Certificate cert)**: byte[] Description: Signs a document by private key. Input parameters: o **nodoOriginal**: document o **privateKey**: private key o **cert**: certificate

Table 94: Interface of XMLSign class of Version Control

Interface Class	***XMLValidation***
Description	This class validates version control xml file:
Methods	• **validateSignature(File xmlfile)**: boolean Description: Validates version control xml file signature. Input parameters: o **xmlfile**: xml file • **validateSignature(File xmlfile,String keystorepath,String keystorepass)**: boolean Description: Validates version control xml file signature. And compares certificate.

Interface Class	***XMLValidation***
	Input parameters: o **xmlfile**: xml file o **keystorepath**: keystore path o **keystorepass**: keystore password • **validateVersionInfo (File xmlFile)**: boolean Description: Validates version control xml file by xsd. Checks type mandatories over xml. Input parameters: o **xmlfile**: xml file • **validateMasterFile (File xmlfile)**: boolean Description: Validates master xml file by xsd. Checks type mandatories over xml. Input parameters: o **xmlfile**: xml file • **getFileCertificate (File xmlfile)**: byte[] Description: Returns xml file signer's certificate. Input parameters: o **xmlfile**: xml file • **validateSignCertificate(File xmlfile,String keystorepath,String keystorepass):**boolean Description: Checks signature certificate of xml is existed in keystore. Input parameters: o **xmlfile**: xml file o **keystorepath**: keystore path o **keystorepass**: keystore password

Table 95: Interface of XMLValidation class of Version Control

Interface Class	***UpdateJob***
Description	The UpdateJob class schedules version control daily routine process:
Methods	• **executeInternal (final JobExecutionContext context)**: void Description: Scheduled job start here.

Interface Class	***UpdateJob***
	Input parameters: o **context**: include configuration • **updateVersion(final VersionControlType vcType, final String confFilePath, final InfoEnvironment environment, final String masterFileUrl, URL configUrl)**: void Description: Loads VERSION_CTRL bean from the application context and calls a function to update the version. Input parameters: o **vcType**: version control type o **confFilePath**: configuration file path getting trusted urls from o **environment**: running environment o **masterFileUrl**: master file url o **configUrl:** configuration file url

Table 96: Interface of Updater class of Updater

Interface Class	***PEPSVersionControl***
Description	Wrapper to Stork Version Control For PEPS:
Methods	• **startProcess(final String confFilePath, final URL configUrl)**: void Description: Starts daily routine process for PEPS.. Input parameters: o **confFilePath**: peps.xml configuration file path o **configUrl:** versioninfo.properties file path

Table 97: Interface of PEPSVersionControl class of Updater

Interface Class	***SPVersionControl***
Description	Wrapper to Stork Version Control For SP:
Methods	• **startProcess(final String confFilePath)**: void Description: Starts daily routine process for SP. Input parameters: o **configUrl:** versioninfo.properties file path

Table 98: Interface of SPVersionControl class of Updater

5.8 Anonymity

5.8.1 Description

The Anonymity module is in charge of building an anonymity network between the PEPS, that is, an onion-routed, delayed delivery network design for the participation in electronic surveys.

This module includes two libraries:

- Anonymity library → implements the logic of the anonymity network.
- AnonymityVC library → version control of the anonymity nodes.

Apart from some configuration related with Version Control PEPS library, Anonymity module requires extra configuration, which includes:

- An own keystore, to handle all the network and own certificates and keys.
- A database schema, to store nodes info and manage the network packages.
- A configuration file, which contains the configuration of the anonymity module.
- Two log files, one for each library.

5.8.2 Package specification

The following diagram shows the main classes involved in the anonymity process.

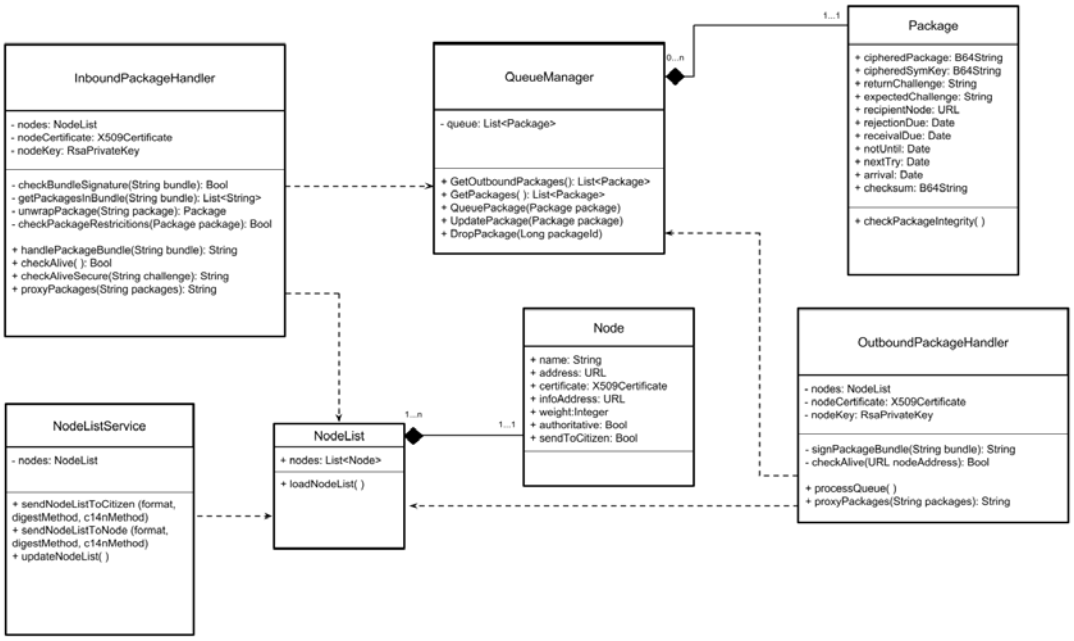


Figure 50. Class diagram for Anonymity

Next, the classes depicted above are briefly explained:

Interface Class	***Node***
Description	Represents a node in the anonymity network. Contains all the information required by the process: • **name**: a common name for the node

Interface Class	***Node***
	• **address**: node service URL • **certificate**: node certificate in X509 format • **infoAddress**: node info URL • **weight**: the weight of the node in the anonymity network • **authoritative**: indicates if the node is authoritative or dependent of another node • **sendToCitizen**: the node will not be sent if it is to be extinguished

Table 99: Interface of Node class of Anonymity

Interface Class	***NodeList***
Description	Represents the list of nodes in the anonymity network. • **nodes**: the list of nodes
Methods	• **loadNodeList()** Description: loads the list of nodes from the database

Table 100: Interface NodeList class of Anonymity

Interface Class	***NodeListService***
Description	Manages the list of nodes. • **nodes:** NodeList object
Methods	• **sendNodeListToCitizen**(String format, String digestMethod, String c14nMethod) : NodeList Description: Sends the node list excluding those to be extinguished. Input parameters:

Table 101: Interface NodeListService class of Anonymity

Interface Class	***Package***
Description	Represents a package in the anonymity network. Contains all the information required by the process: • **cipheredPackage:** the ciphered package to be sent to the next node • **cipheredSymKey:** the symmetrical key used to cypher the package, ciphered with the next node's public key • **returnChallenge:** the challenge code to be sent back to the sender • **expectedChallenge:** the challenge code we will expect from the

Interface Class	***Package***
	recipient • **recipientNode:** the URL of the next node in the anonymity network • **rejectionDue:** the date when the package will lapse • **receivalDue:** no more packages will be accepted by the SP after this date • **notUntil:** the package cannot be relayed to the next node prior to this date • **nextTry:** should the delivery fail, this will be the date when it will be tried again • **arrival:** the date when this package arrived to this node • **checksum:** checksum of the package in Base64 format
Methods	• **checkPackageIntegrity()** Description: calculates the package checksum and compares to the checksum included on the package

Table 102: Interface Package class of Anonymity

Interface Class	***InboundPackageHandler***
Description	Handles the packages before sending them to the anonymity network. It requires the next parameters: • **nodes:** the list of nodes (NodeList object) • **nodeCertificate:** the self node certificate in X509 format • **nodeKey:** the self node private key in RSA format
Methods	• **checkBundleSignature(String bundle)** : Boolean Description: receives a signed bundle and checks that the signature is valid and belongs to a trusted node, in the node list. Input parameters: o **bundle**: the signed xml bundle containing the packages Output: Boolean indicating if the signature is valid • **getPackagesInBundle**(String bundle) : List<String> Description: receives an xml bundle and extracts the xml string for each package contained in it. Input parameters: o **bundle**: the xml bundle Output: the list of packages in xml strings • **unwrapPackage**(String package): Package Description: receives a ciphered package, as extracted from the bundle, deciphers it with the node private key and returns the information contained in it.

Interface Class	*InboundPackageHandler*
	Input parameters: o **package**: a ciphered package Output returns: Package object with the deciphered information • **checkPackageRestricitions**(Package package): Boolean Description: receives a package and checks if there is any reason to drop it (receivalDue date expired, this is the last node and the package was received after participationDue date, etc.). Input parameters: o **package**: a Package object Output: Boolean indicating if the package passes all the restrictions • **handlePackageBundle**(String bundle) : String Description: receives a signed xml bundle of one or more packages, checks the signature, unwraps and checks the integrity of every package and, if restriction check is passed, queues them. Finally, the queue processing procedure is triggered. Input parameters: o **bundle**: the xml bundle Output: a string describing the result of the operation • **checkAlive**( ) : Boolean Description: sends back a simple answer, just to acknowledge that this node is currently running. Output: Boolean indicating if the node is alive • **checkAliveSecure**(String challenge) : String Description: receives a challenge ciphered with his public key and sends back the plain challenge, to securely acknowledge that this node is currently running (to a proxied client). Input parameters: o **challenge**: a string representing the ciphered challenge Output: a string with the result of deciphering the challenge • **proxyPackages**(String bundle): String Description: acting as an online proxy, a bundle of packages not addressed to this PEPS is received; the signature is verified and the proxyPackages method on the OutboundPackageHandler is invoked. Returned challenge codes are sent back to the client on a signed bundle. Input parameters: o **bundle**: the signed xml bundle containing the packages Output: a string containing the challenges of the deciphered packages

Table 103: Interface InboundPackageHandler class of Anonymity

Interface Class	***QueueManager***
Description	Handles the package queue. • **queue**: the queue of packages in a node
Methods	• **GetOutboundPackages**() : List<Package> Description: checks the package queue and returns all the packages that are allowed to be sent at this precise moment, grouped by recipient. Output: the list of Package objects in the queue • **getPackages**() : List<Package> Description: checks the package queue and returns all the packages, grouped by recipient. Output returns: the list of Package objects in the queue • **queuePackage**(Package package) Description: adds a new package to the queue. Input parameters: o **package**: the Package object to add to the queue • **updatePackage**(Package package) Description: updates the contents of an already queued package. Input parameters: o **package**: the Package object to be updated • **dropPackage**(Long packageId) Description: drops a package from the queue (this must be called either to drop expired packages or to delete properly delivered packages after the challenge code has been received and checked). Input parametes: o **packageId**: Long number that identificates the package in the database

Table 104: Interface QueueManager class of Anonymity

Interface Class	***OutboundPackageHandler***
Description	Handles the package queue. • **queue**: the queue of packages in a node
Methods	• **signPackageBundle**(String bundle) : String Description: gets an xml package bundle and uses the node certificate and key to sign it. Input parameters: o **bundle**: a string representing the xml package bundle Output returns: a string containing the xml bundle signed • **checkAlive**(URL nodeAddress): Boolean Description: checks if the specified node is up to receive packages

Interface Class	*OutboundPackageHandler*
	with a challenge package. Input parameters: o **nodeAddress**: URL address of the node Output returns: Boolean indicating if the node is alive or not • **processQueue**() Description: gets a list of packages to be sent. For each recipient, checks that he is up, builds and signs a bundle with all the packages addressed to him, sends it and waits for the challenge codes, which are checked and if correct, the packages are dropped from the queue. Expired packages are also dropped from the list. Failed deliveries are scheduled for a later attempt. This process must be run periodically, besides being triggered on every package arrival. • **proxyPackages**(String packages) : String Description: a set of packages not addressed to this PEPS is received; they are bundled (and signed) and sent to its destination. Returned challenge codes are sent back to the caller. Input parameters: o **packages**: a string representing the xml package bundle Output returns: a string with the challenge codes

Table 105: Interface OutboundPackageHanlder class of Anonymity

6 References

Please note that some of these references are deliverables of this project, pending on approval by the Commission.

[1] STORK1 specifications: https://www.eid-stork.eu/index.php?option=com_processes&Itemid=&act=streamDocument&did=1805

[2] STORK1 process flows: https://www.eid-stork.eu/index.php?option=com_processes&Itemid=&act=streamDocument&did=1465

[3] *How to Construct Pseudorandom Permutations from Pseudorandom Functions*, Luby, Michael, Rackoff, Charles (April 1988), SIAM Journal on Computing 17 (2): 373–386, doi:10.1137/0217022, ISSN 0097-5397

[4] *New European Schemes for Signatures, Integrity, and Encryption (NESSIE),* http://www.cryptonessie.org/

[5] *The Transitioning of Cryptographic Algorithms and Key Sizes, NIST – 02-07-2009,* http://csrc.nist.gov/groups/ST/key_mgmt/documents/Transitioning_CryptoAlgos_070209.pdf

[6] *Deterministic and efficiently searchable encryption*, Mihir Bellare, Alexandra Boldyreva, and Adam O'Neill, CRYPTO, volume 4622 of Lecture Notes in Computer Science, pages 535–552. Springer, 2007.

[7] *Anonymous Pseudonyms for Cross-Border Identification*, M. Barbosa and A. Pinto, Pre-print, CCTC/Departamento de Informática, Universidade do Minho, Portugal – 2010

[8] *Efficient Constructions of Deterministic Encryption from Hybrid Encryption and Code-Based PKE*, Yang Cui, Kirill Morozov, Kazukuni Kobara, and Hideki Imai, Research Center for Information Security (RCIS), National Institute of Advanced Industrial Science & Technology (AIST), Japan - http://staff.aist.go.jp/kirill.morozov/docs/cmki09efficient.pdf.

[9] *ISCED Fields of Education and Training in 2013*

http://www.uis.unesco.org/Education/Documents/isced-fos-consultation-draft-2013-en.pdf.

[10] Policy requirements for certification authorities issuing qualified certificates, ETSI TS 101 456.

[11] Digital Signature Service Core Protocols, Elements, and Bindings Version 1.0, OASIS Standard; 2007 http://docs.oasis-open.org/dss/v1.0/oasis-dss-core-spec-v1.0-os.html

[12] Advanced Electronic Signature Profiles of the OASIS Digital Signature Service Version 1.0, OASIS Standard, 2007 http://docs.oasis-open.org/dss/v1.0/oasis-dss-profiles-AdES-spec-v1.0-os.html

[13] STORK 2.0 Deliverable D4.9: Functional Design, 2013 https://www.eid-stork2.eu/index.php?option=com_processes&controller=document&view=document&task=streamFile&id=450&fid=1854

[14] PEPPOL Deliverable D1.3 Demonstrator and functional Specifications for Cross-Border Use of eSignatures in Public Procurement; Part 7: eID and eSignature Quality Classification; Revision: 2.1

[15] STORK 2.0 Deliverable D4.11 Interface Specification, 2013 https://www.eid-stork2.eu/index.php?option=com_processes&controller=document&view=document&

task=streamFile&id=450&fid=1912

[16] PEPPOL Pan-European Public Procurement Online http://www.peppol.eu/