COMPETITIVENESS AND INNOVATION FRAMEWORK PROGRAMME
ICT Policy Support Programme (ICT PSP)

Towards pan-European recognition of electronic IDs (eIDs)

ICT PSP call identifier: ICT-PSP/2007/1
ICT PSP Theme/objective identifier: 1.2

Project acronym: STORK

Project full title: Secure Identity Across Borders Linked
Grant agreement no.: 224993

D5.8.3c Software Design for PEPS architecture

Deliverable Id :	**D5.8**
Deliverable Name :	**D5.8 Technical Design**
Status :	**Final**
Dissemination Level :	**Public**
Due date of deliverable :	**December 31st 2011**
Actual submission date :	**November 11st 2011**
Work Package :	**5.1**
Organisation name of lead contractor for this deliverable :	**ES-MAP**
Author(s):	**Diana Berbecaru, Joaquín Alcalde-Moraño, Jorge López Hernández-Ardieta, Renato Portela, Ricardo Ferreira**
Partner(s) contributing :	**IT, PT, ES**

Abstract: This document presents the Software design of the PEPS architecture. It pretends to specify the behaviour of the components, in such a way that programmers can work with it.
The view which was offered by D5.8.3a, by business process, is now complemented with views by components and classes. This document pretends to specify the behaviour of these components, in such a way that programmers can work with it.

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

History

Version	*Date*	*Modification reason*	*Modified by*
0.1	04/10/2011	Initial version, the same as D5.8.2c	Diana Berbecaru, Joaquin Alcalde-Moraño, Jorge López Hernández-Ardieta, Renato Portela, Ricardo Ferreira
0.2	14/10/2011	Inclusion of the standard format of the logfile	Ricardo Ferreira, John Heppe
Final 1.0	11/11/2011	Finalization	R. Wannee, A. v. Overeem (Capgemini)

Intermediate internal versions, e.g. for quality reviews, have been omitted

Table of contents

List of figures

List of tables

Executive summary

This document presents the Software design of the PEPS architecture. It pretends to specify the behaviour of its components, in such a way that programmers can work with it.

The view which was offered by D5.8.3a, by business process, is now complemented with views by components and classes.

As this is one document of the deliverable D5.8.3 Technical Design, please refer to D5.8.3 for the Executive Summary.

1 Introduction

1.1 Objective

This document presents the Software design of the STORK PEPS architecture. It pretends to specify the behaviour of its components, in such a way that programmers can work with it.

The view which was offered by D5.8.3a, by business process, is now complemented with views by components and classes.

Thus please note that this document is to be understood by programmers.

As this is one document of the deliverable D5.8.3 Technical Design, please refer to D5.8.3 for the other parts of the introduction.

1.2 Changes in this document

As explained in D5.8.3 Technical design, mostly changes were due to the comments by the EC reviewers. Nevertheless, in this document a major change has been applied: the inclusion of the log file format. These changes has lead to a new chapter.

2 PEPS

2.1 Introduction

A Pan European Proxy Service or Server (PEPS), as defined by IDABC, is a system that

1. hides national problems for other countries
2. elevates the national circle of trust to European level.

These general objectives have been translated to functional specifications in D5.7.3 Functional Specifications and were object of the description D5.8.3a Software Architecture Design. The actual document describes them for the people who'll be in charge of its construction and maintenance.

2.2 Package definition

In order to sort out the source code to be generated during the implementation phase, next package classification approach is proposed.

The motivation is the modularity principle, allowing specific functionalities to be included in a future without changing the functional view of the packages.

These packages and sub-packages contain both S-PEPS (Service Provider PEP) and C-PEPS (Client PEPS) functionalities. Therefore, these packages implement the whole STORK PEPS functionality according to STORK Functional Requirements and High-Level Design.

Package	Description
eu.stork.peps	Package for PEPS functionality
eu.stork.peps.auth	Package that collects the Authentication Service functionality
eu.stork.peps.auth.commons	Common Authentication Service functionalities to be deployed in every PEPS is contained in this package In particular, it contains the SAML Engine that implements the SAML messages management
eu.stork.peps.auth.commons.core	OpenSAML start-up and configuration is restricted to this package
eu.stork.peps.auth.specifics	Specific PEPS functionality of the Authentication Service
eu.stork.peps.keystores	Package for the keystore management
eu.stork.peps.validation	Package that collects the Validation Service functionality
eu.stork.peps.validation.commons	Common Validation Service functionalities to be deployed in every PEPS is contained in this package In particular, it contains the OCSP Engine that implements the OCSP messages management
eu.stork.peps.validation.specifics	Specific PEPS functionality of the Validation Service

Table 1: Package descriptions

2.3 Diagram

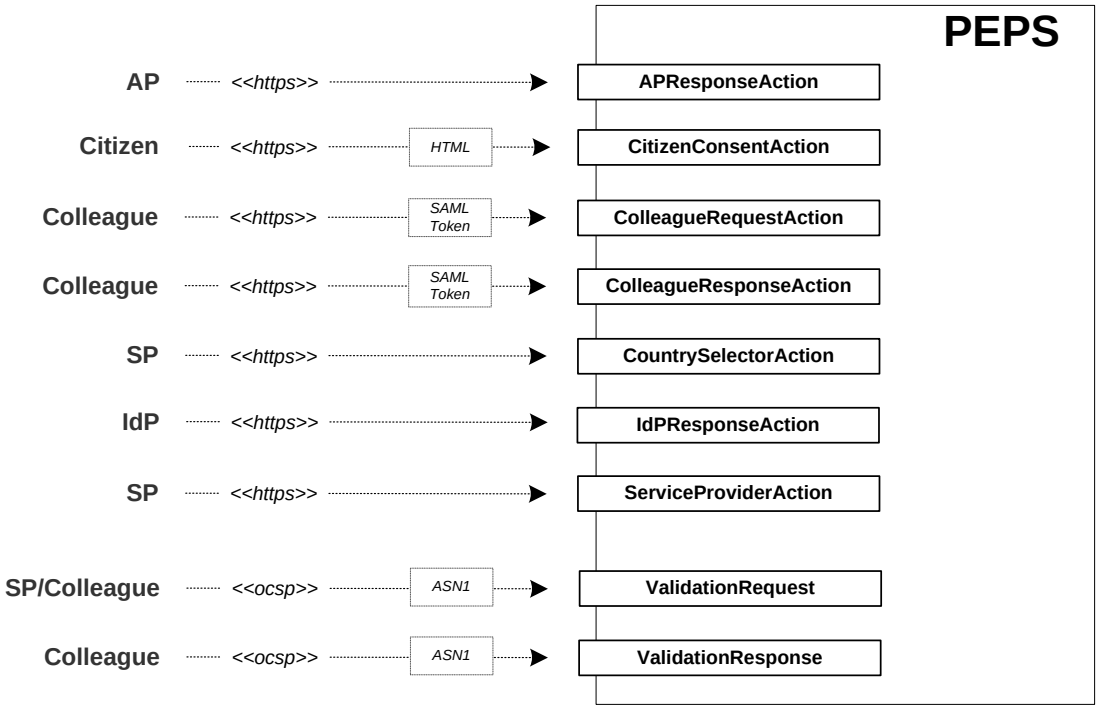
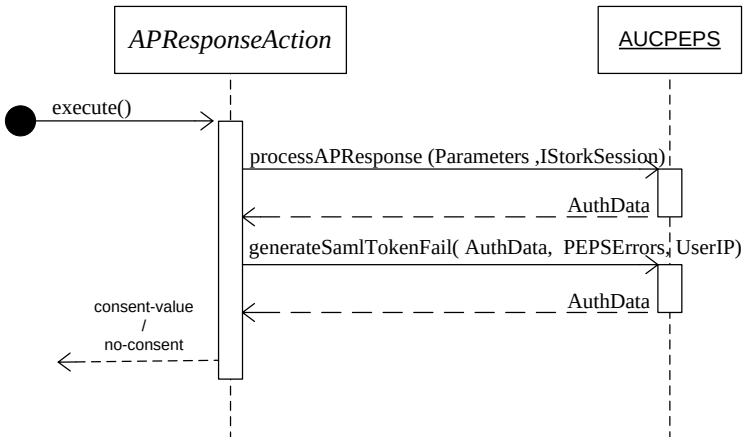
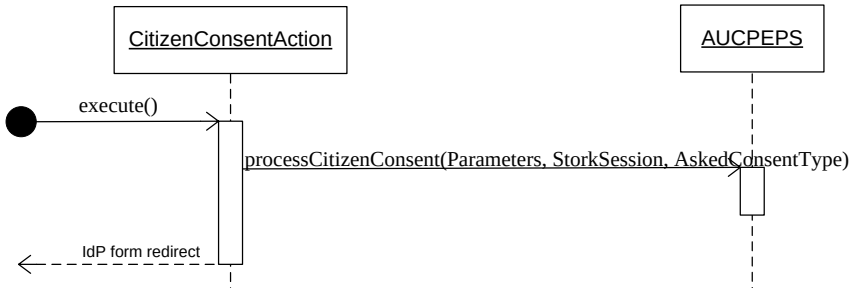
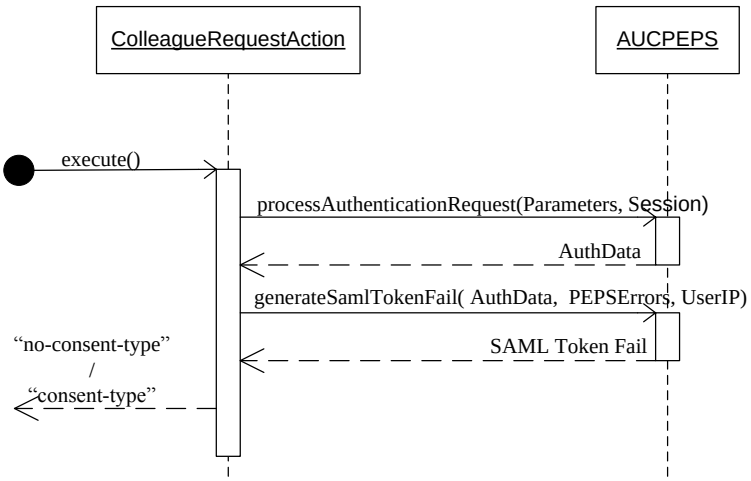


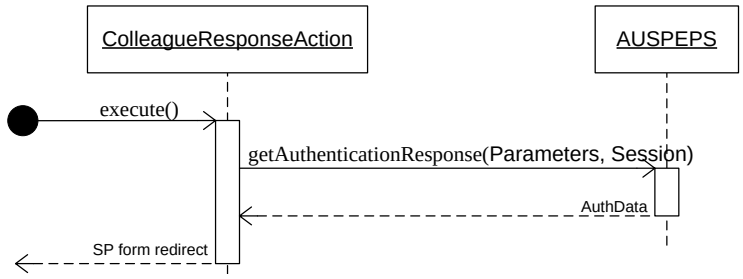
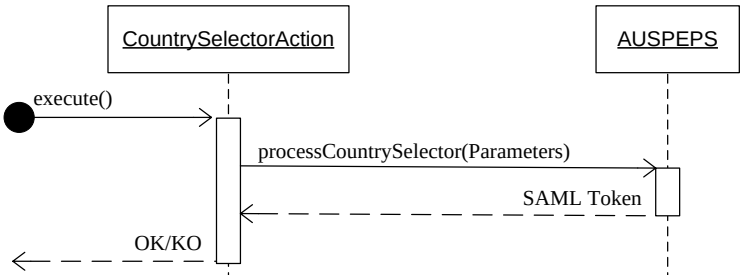
Figure 1: PEPS global overview

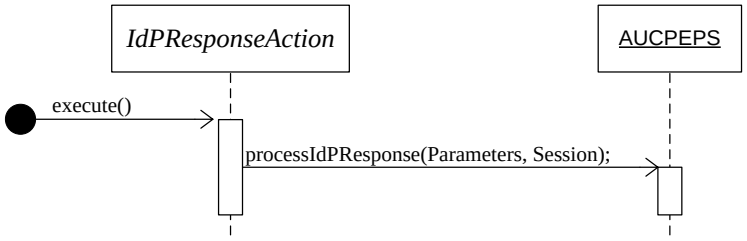
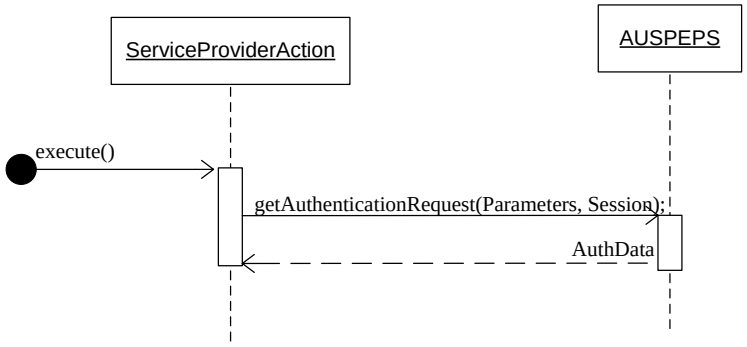
2.4 Actions

Action Class	*APResponseAction*	
Methods	**execute () :** *String*	
	Description	Receives the request from AP, requests the AUCEPS to validate AP Response and Present Citizen Value Consent Form to Citizen (specific) or sends a SAML Response to S-PEPS.
	Action	*APResponseAction*
	Input Parameters	
	Output Returns	String
	Sequence Diagram	



Action Class	*CitizenConsentAction*	
Methods	**execute () :** *void*	
	Description	Handles the Citizen's Consent, then redirects the citizen to the IdP for the login process.
	Servlet	*CitizenConsentAction*
	Input Parameters	
	Output Returns	String
	Sequence Diagram	
Action Class	*ColleagueRequestAction*	
Methods	**execute () :** *String*	
	Description	Receives, validates SAML Authentication request from Colleague PEPS and Presents a Citizen Consent Form to the Citizen chooses the Attributes that consents to be requested from AP (if configured to do so, otherwise it will authenticate citizen on IdP).
	Servlet	*ColleagueRequestAction*
	Input Parameters	
	Output Returns	String("no-consent-type": don't display citizen consent form/"success":display citizen consent form)
	Sequence Diagram	
Action Class	*ColleagueResponseAction*	

Methods	**execute () :** *String*	
	Description	Handles and validates SAML Authentication response (with an Attribute Response or an Error Response) from Colleague PEPS, creates and send an Authentication response.
	Servlet	*ColleagueResponseAction*
	Input Parameters	
	Output Returns	String
	Sequence Diagram	
Action Class	*CountrySelectorAction*	
Methods	**execute () :** *String*	
	Description	Generates and Present Country Selector Form to Citizen.
	Servlet	*CountrySelectorAction*
	Input Parameters	
	Output Returns	String
	Sequence Diagram	
Action Class	*IdPResponseAction*	
Methods	**execute () :** *String*	
	Description	Receives the request from IdP, updates the citizen session.
	Action	*IdPResponseAction*
	Input Parameters	
	Output Returns	String

	Sequence Diagram  	
Action Class	*ServiceProviderAction*	
Methods	**execute () :** *String*	
	Description	Validates SP Request, normalises data and sends SAML Authentication Request to Colleague PEPS.
	Servlet	*ServiceProviderAction*
	Input Parameters	
	Output Returns	String
	Sequence Diagram  	
Servlet Class	*ValidationRequest*	
Methods	**doPost (***HttpRequest, HttpResponse***) :** *void*	
	Description	
	Input Parameters	- HttpRequest - HttpResponse
	Output Returns	void

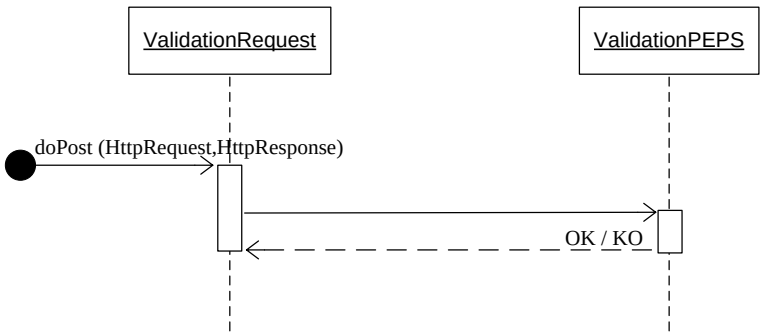
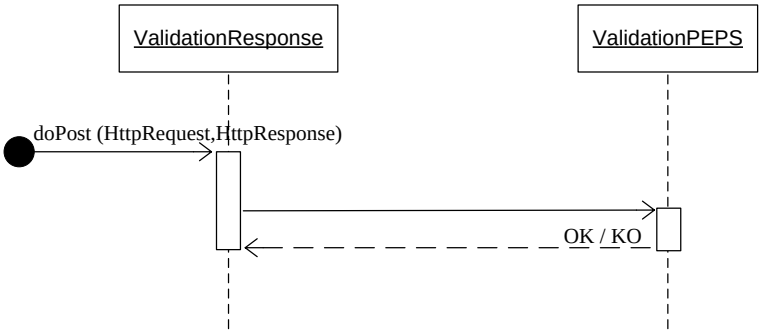
<table>
<tr><td></td><td>Sequence Diagram
 
</td></tr>
<tr><td>Servlet Class</td><td>ValidationResponse</td></tr>
<tr><td>Methods</td><td>doPost (HttpRequest, HttpResponse) : void</td></tr>
<tr><td></td><td>Description</td></tr>
<tr><td></td><td>Input Parameters
- HttpRequest
- HttpResponse</td></tr>
<tr><td></td><td>Output Returns
void</td></tr>
<tr><td></td><td>Sequence Diagram
 
</td></tr>
</table>

Table 2: Actions

2.5 Components

2.5.1 Component Diagram

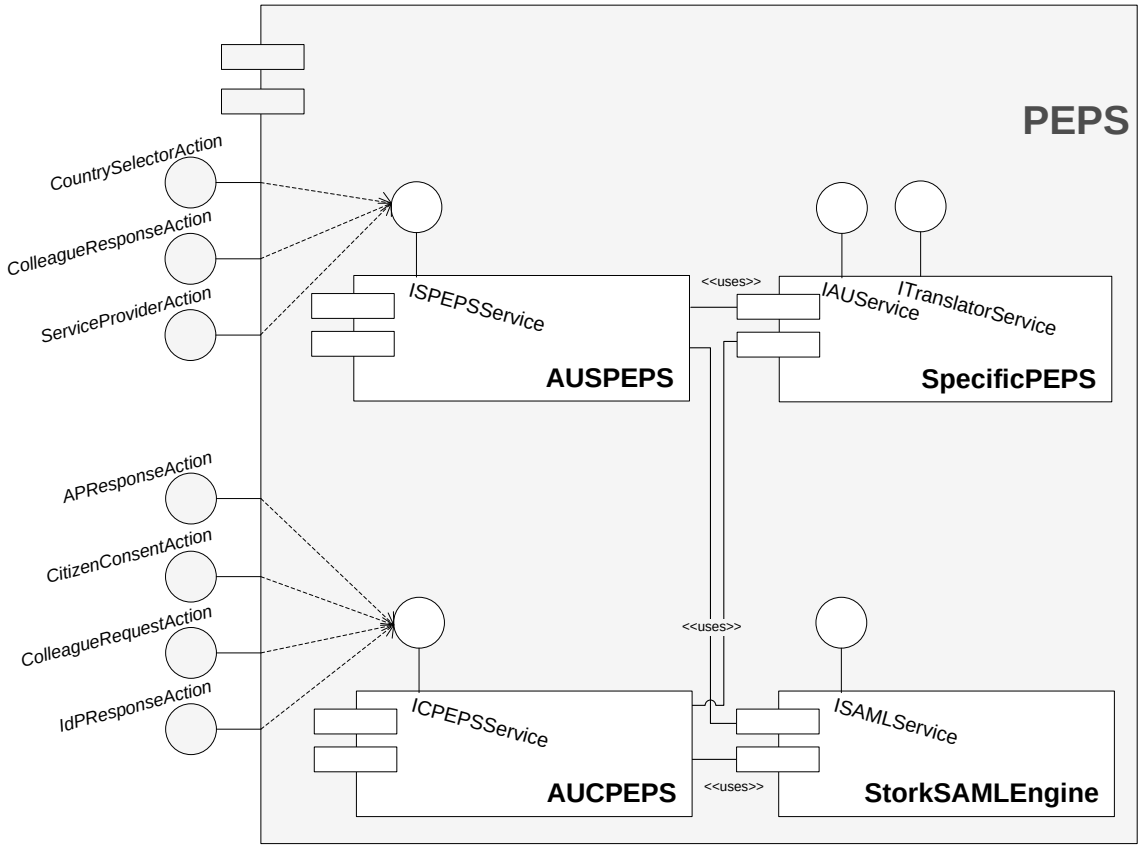


Figure 2: PEPS Component Diagram

2.5.2 AUSPEPS component

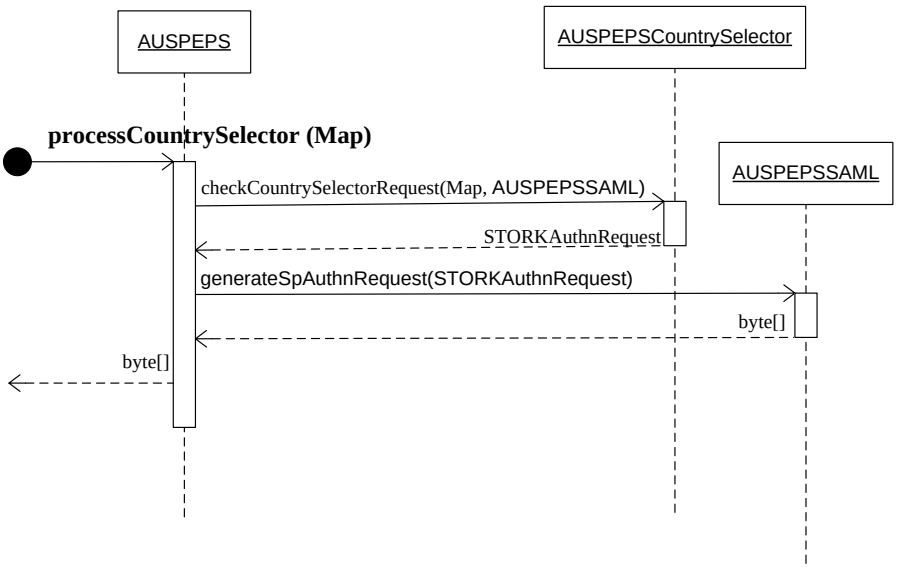
2.5.2.1 Description

The AUSPEPS component receives Authentication Requests and replies with an Authentication Response. It acts as a gateway: gets the Requests from Service Provider to be handled on the AUSPEPSSAML service, and forward the Authentication Responses from the AUSPEPSSAML serviceto the Service Provider.

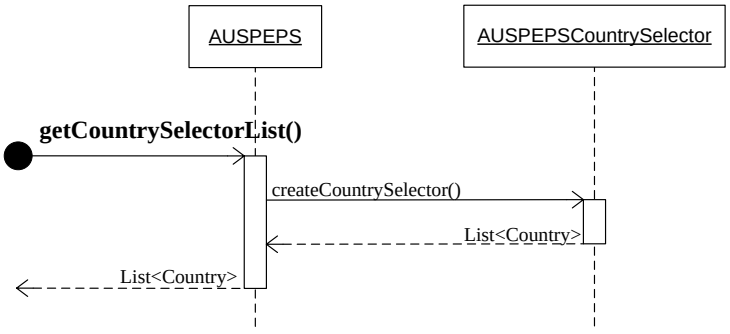
2.5.2.2 Interfaces

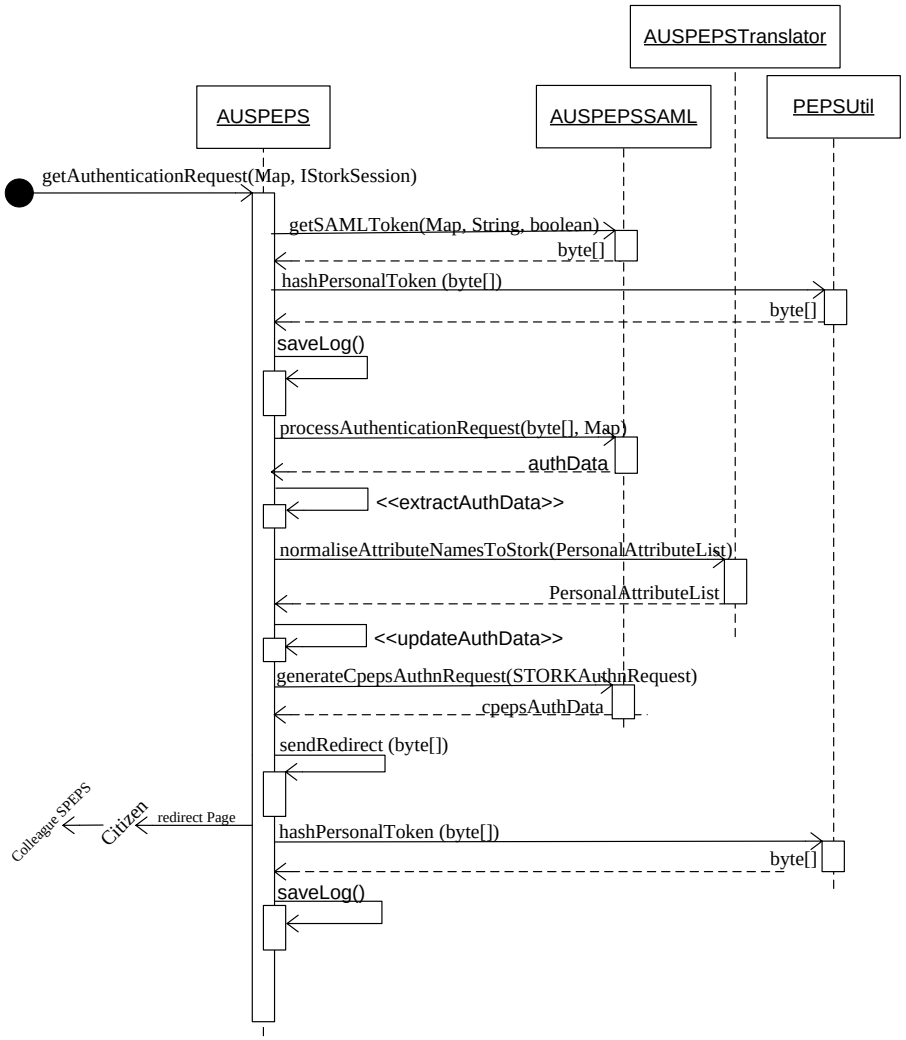
Interface Class	*ISPEPSService*
Methods	▪ **processCountrySelector(Map) : byte[]** ▪ **getCountrySelectorList() : List<Country>** ▪ **getAuthenticationRequest (***Map, IStorkSession***) :** *STORKAuthnRequest*

	▪ **getAuthenticationResponse(***Map, IStorkSession***) :** *STORKAuthnRequest*
	processCountrySelector (Map) : *byte[]*
	Description: Handles the Country Selector Request with a few parameters (QAA asked, mandatory and optional attributes needed and url for the response), validates the origin of the request.
	Interface: *ISPEPSService*
	Input Parameters: Map
	Output Returns: byte[]
	Sequence Diagram



	getCountrySelectorList() : List<Country>
	Description: Gets the list of Country Selector .
	Interface: *ISPEPSService*
	Input Parameters: Map
	Output Returns: List<Country>
	Sequence Diagram



	getAuthenticationRequest (*Map, IStorkSession***) :** *STORKAuthnRequest*	
	Description	Validates the origin of the request, normalise data to Stork Format, creates a SAML Authentication Query to send to Colleague PEPS.
	Interface	*ISPEPSService*
	Input Parameters	• Map • IStorkSession
	Output Returns	STORKAuthnRequest
	Sequence Diagram	
	getAuthenticationResponse (*Map, IStorkSession*) **:** *STORKAuthnRequest*	
	Description	Validates the origin of the request, normalise data to Stork Format, creates a SAML Authentication Response (with an Attribute Response or an Error Response) and sends to a Service Provider.
	Interface	*ISPEPSService*
	Input Parameters	• Map • IStorkSession
	Output	STORKAuthnRequest

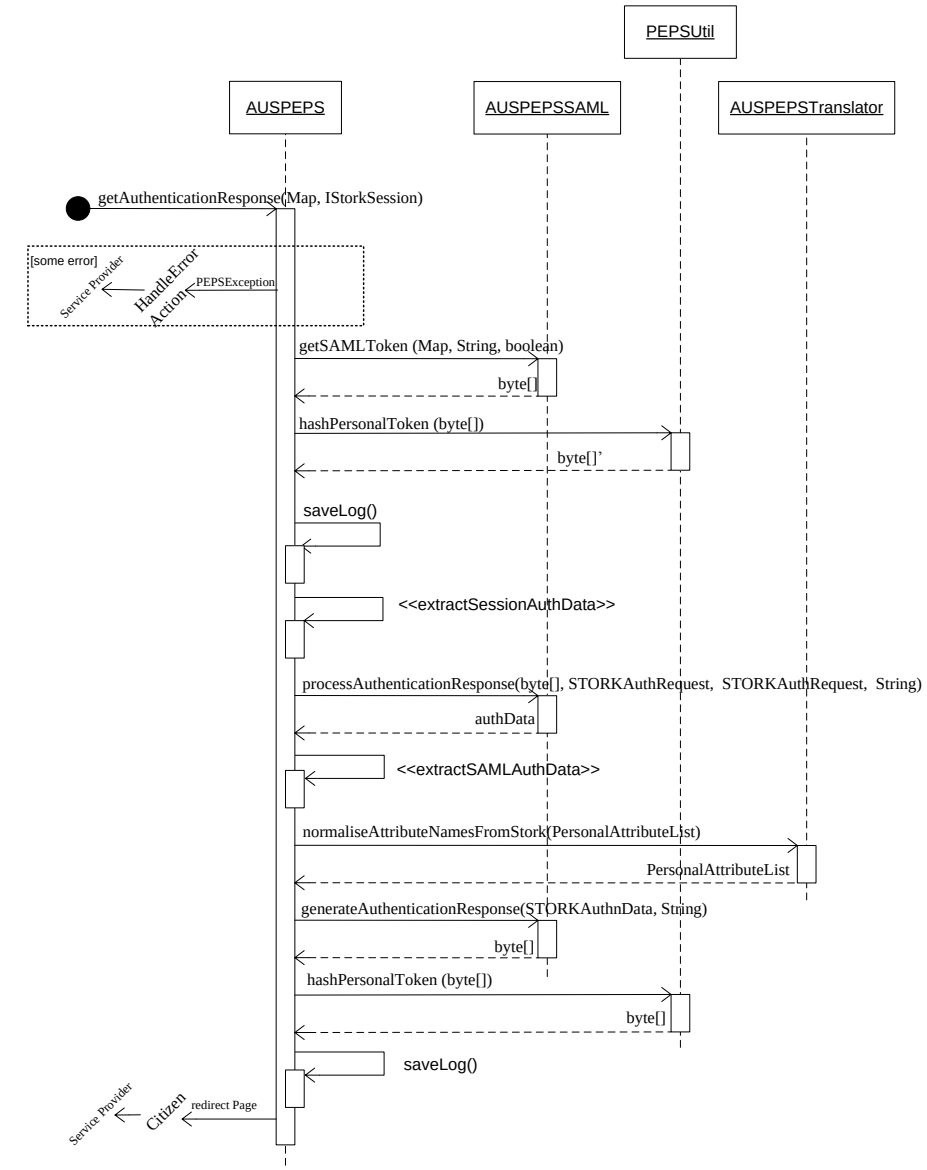
Returns	
Sequence Diagram	
Output Returns	none

Table 3: Authentication SPEPS Interfaces

2.5.2.3 Components

2.5.2.3.1 Component Diagram

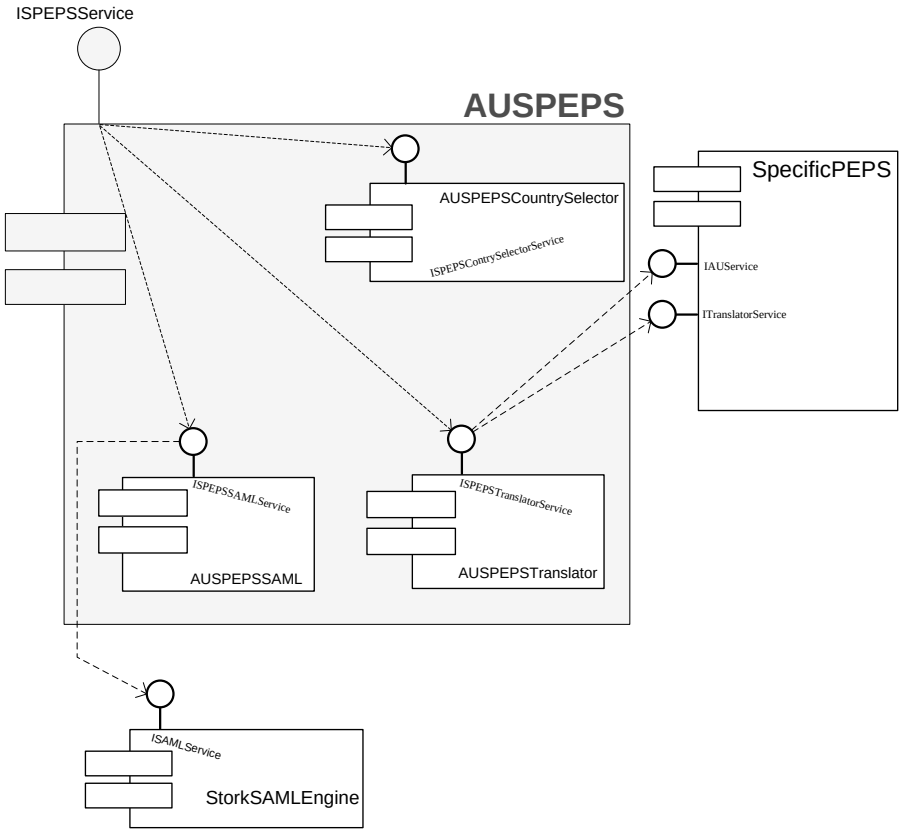


Figure 3: Authentication-SPEPS Component Diagram

2.5.2.3.2 AUSPEPSSAML component

2.5.2.3.2.1 Description

The AUSPEPSSAML creates SAML Authentication request to C-PEPS and receives, validates the SAML Authentication response from C-PEPS and creates Authentication responses back to SP.

2.5.2.3.2.2 Interfaces

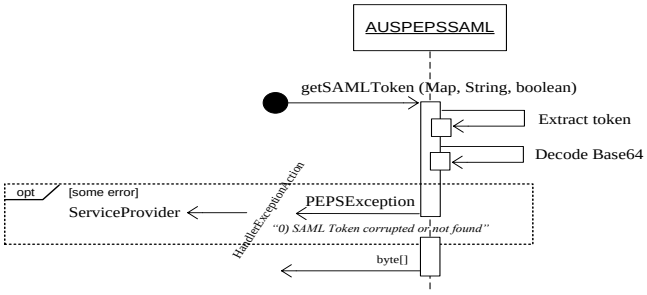
Interface Class	*ISPEPSSAMLService*
Methods	▪ **getSAMLToken** (*Map, String, boolean*) : *byte[]* ▪ **processAuthenticationRequest** (*byte[], Map*) : STORKAuthnRequest ▪ **processAuthenticationResponse** (*byte[],STORKAuthnRequest, STORKAuthnRequest,String*): STORKAuthnRequest ▪ **generateAuthenticationRequest(String, STORKAuthnRequest): STORKAuthnRequest** ▪ **generateSpAuthnRequest** (STORKAuthnRequest) : STORKAuthnRequest ▪ **generateCpepsAuthnRequest**(STORKAuthnRequest): STORKAuthnRequest ▪ **generateAuthenticationResponse** (STORKAuthnRequest, String) :

STORKAuthnRequest

- **generateErrorAuthenticationResponse** (String, String, String,String, String, String, String) : *byte[]*

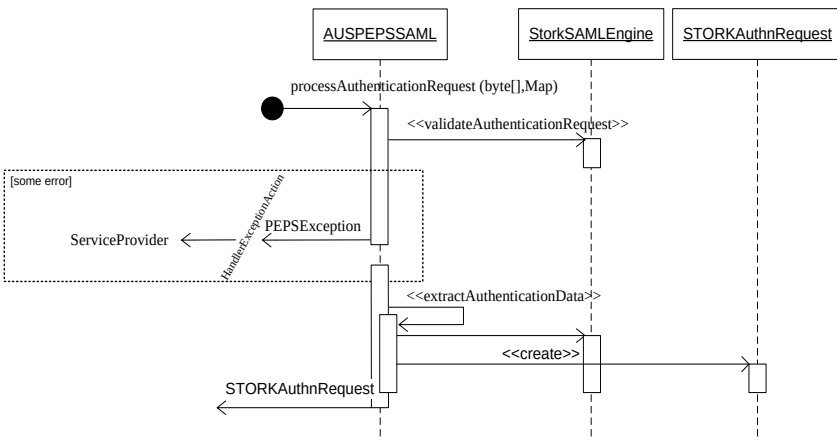
getSAMLToken (*Map, String, boolean*) : *byte[]*	
Description	Gets the SAML Token from the request.
Interface	*ISPEPSSAMLService*
Input Parameters	• Map • String • boolean
Output Returns	*byte[]*

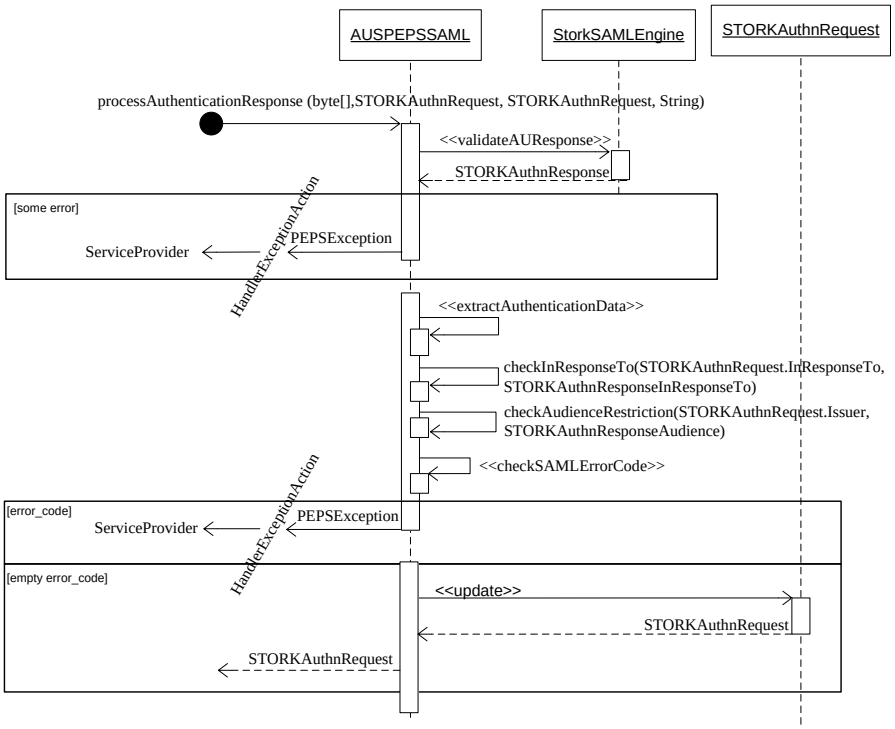
Sequence Diagram

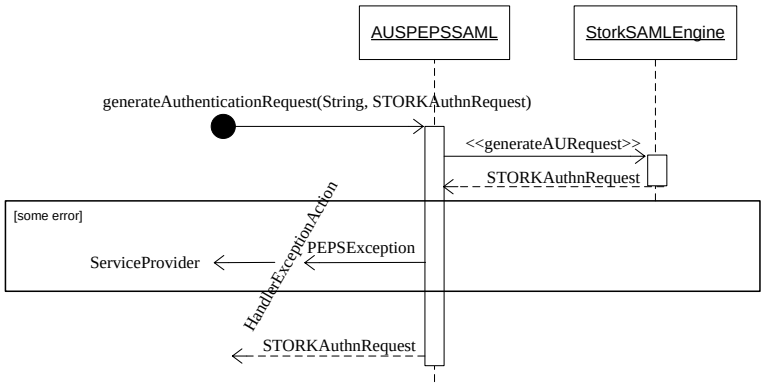
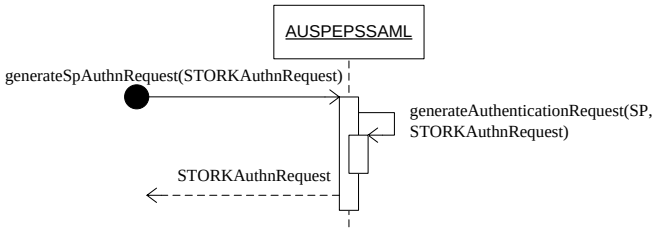
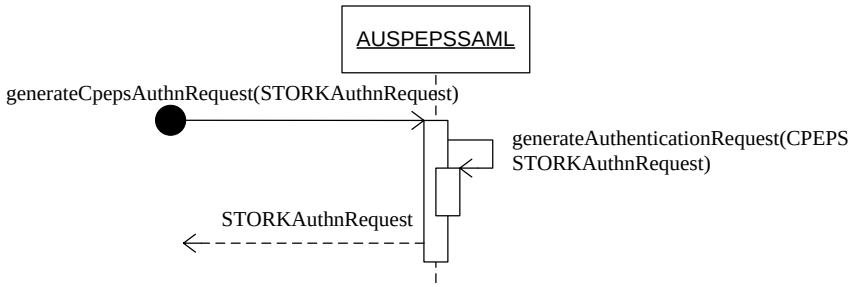


processAuthenticationRequest (*byte[],Map***): STORKAuthnRequest**	
Description	Validates Authentication Request from Service Provider.
Interface	*ISPEPSSAMLService*
Input Parameters	• byte[] • Map
Output Returns	STORKAuthnRequest

Sequence Diagram



	processAuthenticationResponse (*byte[],STORKAuthnRequest, STORKAuthnRequest, String***): STORKAuthnRequest**	
	Description	Validates Authentication Responses from Colleague PEPS.
	Interface	*ISPEPSSAMLService*
	Input Parameters	• byte[] • STORKAuthnRequest • STORKAuthnRequest • String -> IPUser
	Output Returns	STORKAuthnRequest
	Sequence Diagram	
	generateAuthenticationRequest(String, STORKAuthnRequest**):** STORKAuthnRequest	
	Description	Creates a SAML Authentication Message.
	Interface	*ISPEPSSAMLService*
	Input Parameters	• String • STORKAuthnRequest
	Output Returns	STORKAuthnRequest

	Sequence Diagram	
	generateSpAuthnRequest **(STORKAuthnRequest): STORKAuthnRequest**	
	Description	Creates the SP SAML Authentication Message (counry selectr).
	Interface	*ISPEPSSAMLService*
	Input Parameters	STORKAuthnRequest
	Output Returns	STORKAuthnRequest
	Sequence Diagram	
	generateCpepsAuthnRequest **(STORKAuthnRequest): STORKAuthnRequest**	
	Description	Creates the SP SAML Authentication Message (counry selectr).
	Interface	*ISPEPSSAMLService*
	Input Parameters	STORKAuthnRequest
	Output Returns	STORKAuthnRequest
	Sequence Diagram	
	▪ **generateAuthenticationResponse** (STORKAuthnRequest, String) : *STORKAuthnRequest*	
	Description	Creates a SAML Authentication Response Message.

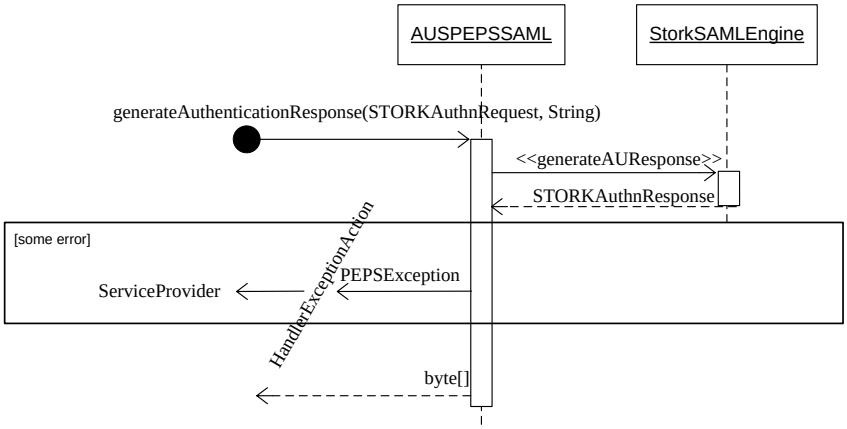
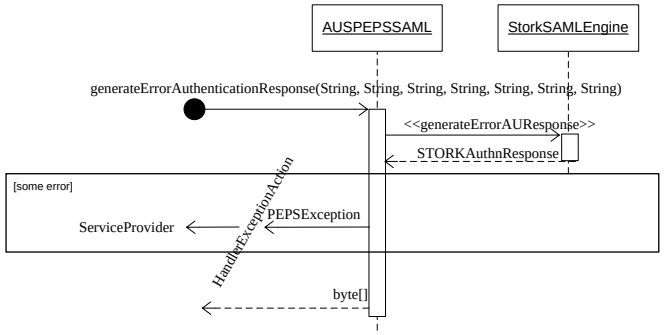
	Interface	*ISPEPSSAMLService*
	Input Parameters	• STORKAuthnRequest • String -> IPUser
	Output Returns	STORKAuthnRequest
	Sequence Diagram	
	generateErrorAuthenticationResponse(String, String, String, String, String, String, String): *byte[]*	
	Description	Creates a SAML Error Authentication Response Message with the error code received.
	Interface	*ISPEPSSAMLService*
	Input Parameters	• String -> InResponseTo • String -> Issuer • String -> Destination • String -> UserAddress • String -> StatusCode • String -> SubStatusCode • String -> Message
	Output Returns	byte[]
	Sequence Diagram	

Table 4: Authentication SPEPSSAML component interface

2.5.2.3.2.3 Other methods

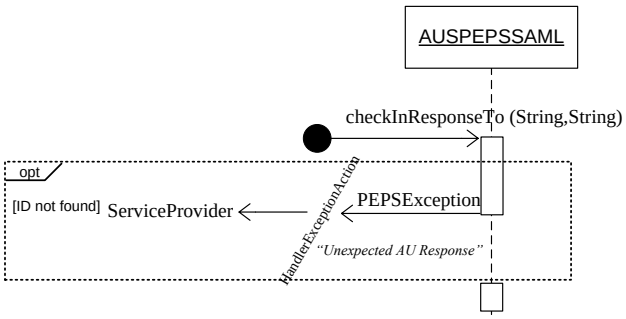
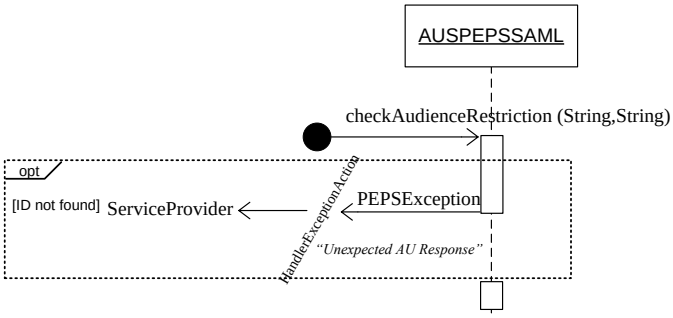
<table>
<tr><td rowspan="11">Methods</td><td colspan="2">▪ checkInResponseTo (String, String): void
▪ checkAudienceRestriction(String,String): void</td></tr>
<tr><td colspan="2">checkInResponseTo (String, String) : void</td></tr>
<tr><td>Description</td><td>Checks if the ID of the SAML Authentication Request that the Authentication Response is answered exists.</td></tr>
<tr><td>Interface</td><td></td></tr>
<tr><td>Input Parameters</td><td>• String → the original SAML Auhtentication Request ID
• String → the SAML Auhtentication Request ID</td></tr>
<tr><td>Output Returns</td><td>None</td></tr>
<tr><td colspan="2">Sequence Diagram
 
</td></tr>
<tr><td colspan="2">checkAudienceRestriction(String,String): void</td></tr>
<tr><td>Description</td><td>Compares the issuer to the audience restriction.</td></tr>
<tr><td>Interface</td><td></td></tr>
<tr><td>Input Parameters</td><td>• String → the original SAML Auhtentication Issuer
• String → the SAML Auhtentication Issuer</td></tr>
<tr><td>Output Returns</td><td>None</td></tr>
<tr><td colspan="2">Sequence Diagram
 
</td></tr>
</table>

Table 5: Authentication SPEPSSAML component other methods

2.5.2.3.3 AUSPEPSCountrySelector component

2.5.2.3.3.1 Description

The AUSPEPSCountrySelector creates the list of Country to be included on the Country Selector form, request to the AUSPEPSSAML component to create the authentication data and checks if a SP is allowed to access requested attributes.

2.5.2.3.3.2 Interfaces

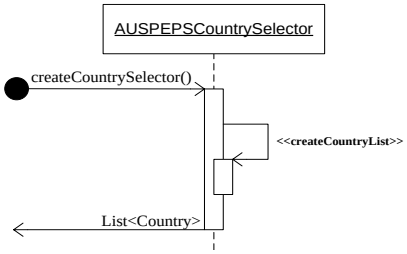
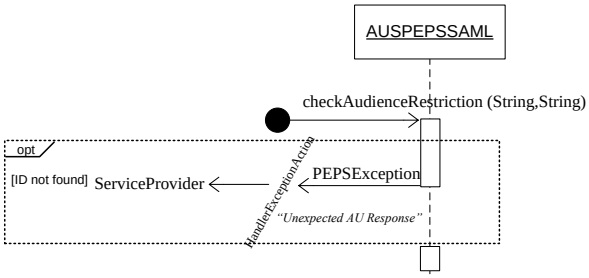
<table>
<tr><td>Interface Class</td><td colspan="2">ISPEPSCountrySelectorService</td></tr>
<tr><td>Methods</td><td colspan="2">▪ createCountrySelector () : List<Country>
▪ checkCountrySelectorRequest (Map, AUSPEPSSAML): void</td></tr>
<tr><td></td><td colspan="2">createCountrySelector() : HTMLPage</td></tr>
<tr><td></td><td>Description</td><td>Creates the list of the countries and Country Selector.</td></tr>
<tr><td></td><td>Interface</td><td>ISPEPSContrySelectorService</td></tr>
<tr><td></td><td>Input Parameters</td><td>None</td></tr>
<tr><td></td><td>Output Returns</td><td>List<Country></td></tr>
<tr><td></td><td colspan="2">Sequence Diagram
 
</td></tr>
<tr><td></td><td colspan="2">checkCountrySelector Request(Map, AUSPEPSSAML): void</td></tr>
<tr><td></td><td>Description</td><td>Check the paramenters in the Request</td></tr>
<tr><td></td><td>Interface</td><td>ISPEPSContrySelectorService</td></tr>
<tr><td></td><td>Input Parameters</td><td>• Map
• AUSPEPSSAML</td></tr>
<tr><td></td><td>Output Returns</td><td>void</td></tr>
<tr><td></td><td colspan="2">Sequence Diagram
 
</td></tr>
</table>

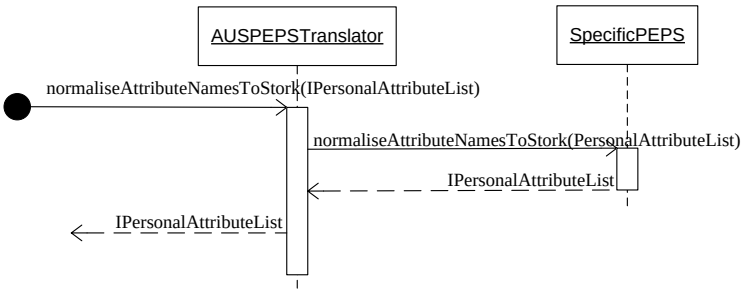
Table 6: Authentication SPEPSSAML component Interface

2.5.2.3.4 AUSPEPSTranslator component

2.5.2.3.4.1 Description

The AUSPEPSTranslator translates the attribute names ans values to and from Stork format.

2.5.2.3.4.2 Interfaces

<table>
<tr><td>Interface Class</td><td colspan="2">ISPEPSTranslatorService</td></tr>
<tr><td>Methods</td><td colspan="2">▪ normaliseAttributeNamesToStork (IPersonalAttributeList) : IPersonalAttributeList
▪ normaliseAttributeNamesFromStork (IPersonalAttributeList) : IPersonalAttributeList
▪ normaliseAttributeValuesToStork () : IPersonalAttributeList</td></tr>
<tr><td></td><td colspan="2">normaliseAttributeNamesToStork (IPersonalAttributeList):IPersonalAttributeList</td></tr>
<tr><td></td><td>Description</td><td>Converts AttributeList names to Stork format (checking a configuration file)</td></tr>
<tr><td></td><td>Interface</td><td>IAUTranslatorService</td></tr>
<tr><td></td><td>Input Parameters</td><td>• IPersonalAttributeList</td></tr>
<tr><td></td><td>Output Returns</td><td>• IPersonalAttributeList</td></tr>
<tr><td></td><td colspan="2">Sequence Diagram
 
</td></tr>
<tr><td></td><td colspan="2">normaliseAttributeNamesFromStork (IPersonalAttributeList): IPersonalAttributeList</td></tr>
<tr><td></td><td>Description</td><td>Converts AttributeList names from Stork format to Country format (checking a configuration file)</td></tr>
<tr><td></td><td>Interface</td><td>IAUTranslatorService</td></tr>
<tr><td></td><td>Input Parameters</td><td>IPersonalAttributeList</td></tr>
<tr><td></td><td>Output Returns</td><td>IPersonalAttributeList</td></tr>
</table>

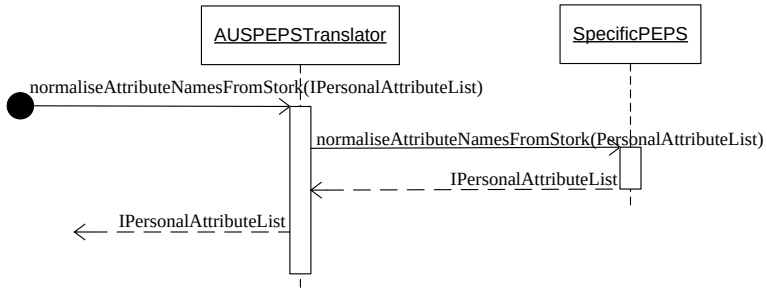
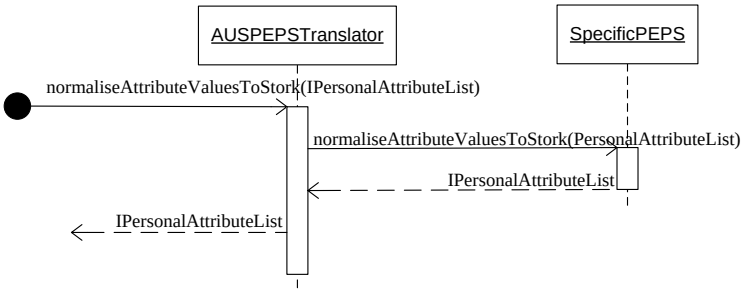
	Sequence Diagram 
	normaliseAttributeValuesToStork (IPersonalAttributeList): IPersonalAttributeList
	Description — Converts AttributeList values from Stork format to Country format (checking a configuration file)
	Interface — *IAUTranslatorService*
	Input Parameters — IPersonalAttributeList
	Output Returns — IPersonalAttributeList
	Sequence Diagram 

Table 7: AuthenticationSPEPSSAML component interface

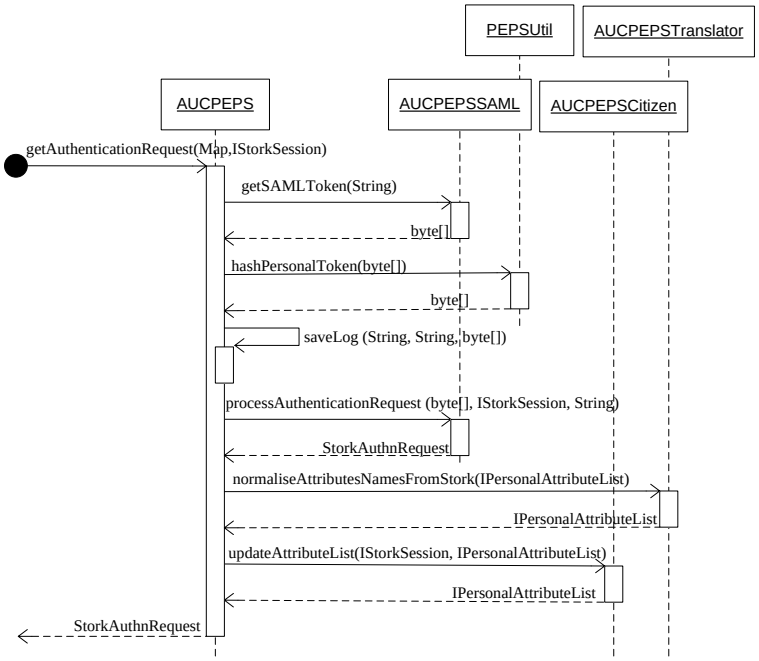
2.5.3 AUCPEPS component

2.5.3.1 Description

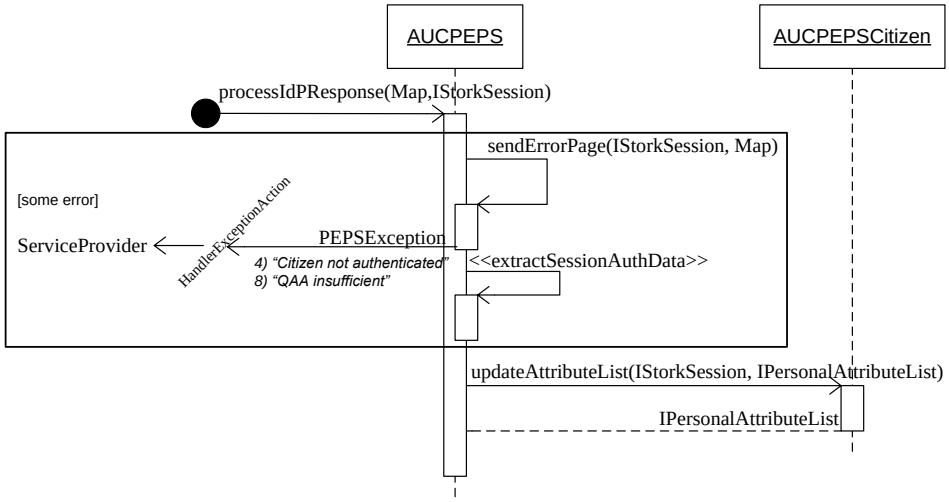
The AUCPEPS component receives Authentication Requests and replies with an Authentications Response. It acts as a gateway: gets the Requests from a Colleague PEPS to be handled on the AUCPEPSSAML, and forward the Authentication Responses created by the AUCPEPSSAML component to the Colleague PEPS.

2.5.3.2 Interfaces

Interface Class	*ICPEPSService*
Methods	▪ **processAuthenticationRequest (***Map ,IStorkSession***) :** *STORKAuthnRequest* ▪ **processCitizenConsent (***Map, IStorkSession, Boolean)* **:** *IPersonalAttributeList* ▪ **processIdPResponse(***Map, IStorkSession***)** : void ▪ **processAPResponse(***Map,IStorkSession***)** :StorkAuthnRequest

	▪ **processAuthenticationRequest (***Map ,IStorkSession***) : *STORKAuthnRequest***	
	Description	Validates the origin of the request, normalise data to Native Format and prepares to ask the ConsentTtype (if configured to do so).
	Interface	*ICPEPSService*
	Input Parameters	• Map • IStorkSession
	Output Returns	STORKAuthnRequest
	Sequence Diagram	
	processIdPResponse(*Map, IStorkSession***)** : void	
	Description	Validates the IdP Response and updates the attribute list if the IdP provided any attributes' value. .
	Interface	*ICPEPSService*
	Input Parameters	• Map • IStorkSession
	Output Returns	None

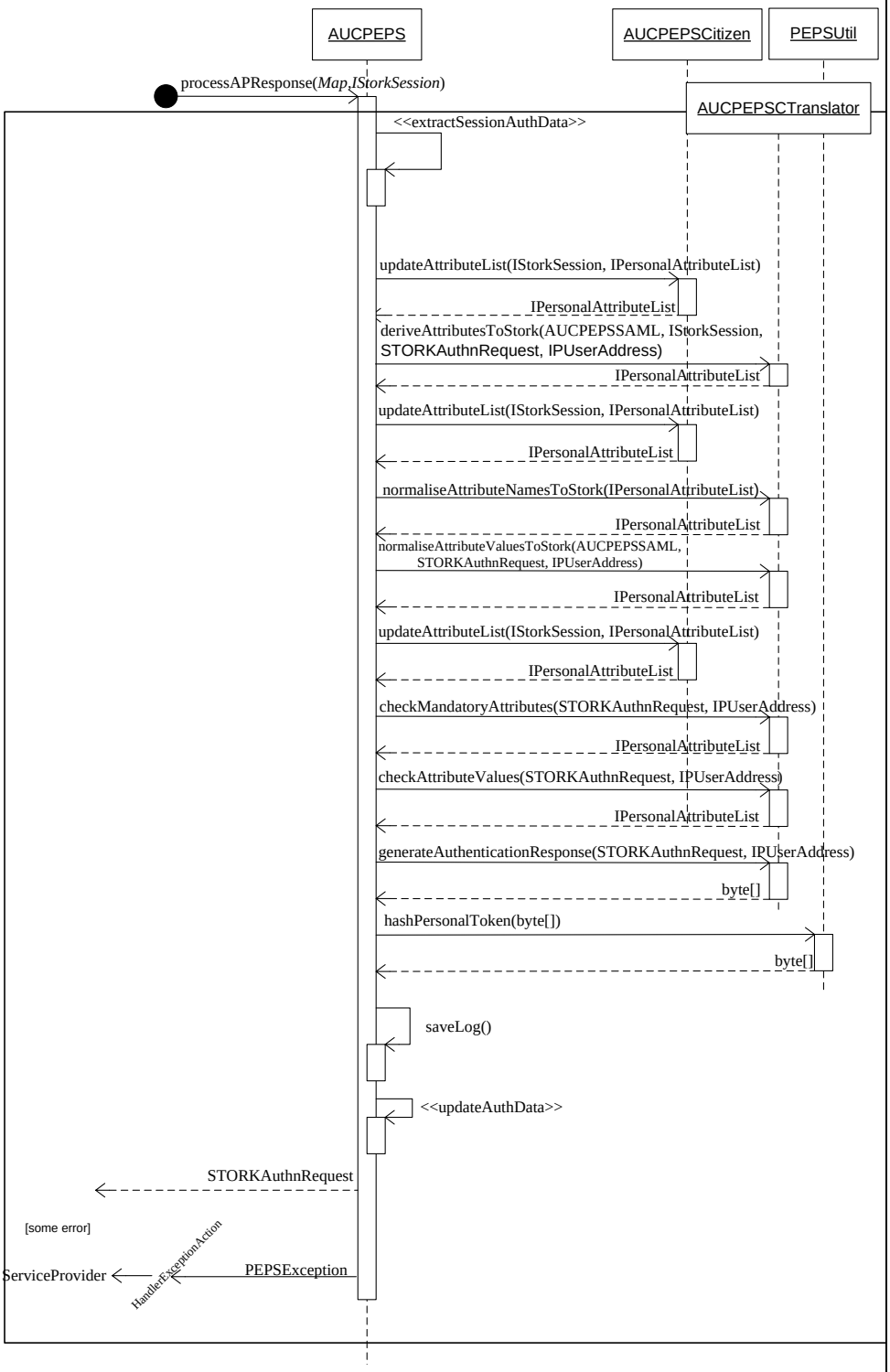
Sequence Diagram



processAPResponse(*Map,IStorkSession***)** :STORKAuthnRequest

Description	Validates the response, checks the mandatory attributes' values, normalizes/derives the attributes to STORK format and generates the SAML Response to S-PEPS..
Interface	*ICPEPSService*
Input Parameters	• Map • STORKAuthnRequest
Output Returns	

Sequence Diagram



processCitizenConsent (*Map, IStorkSession, boolean)* **:** *IPersonalAttributeList*

Description	Validates the consent sent by the citizen, redirects the citizen to the IdP for the login process.
Interface	*ICPEPSService*
Input Parameters	• Map • IStorkSession

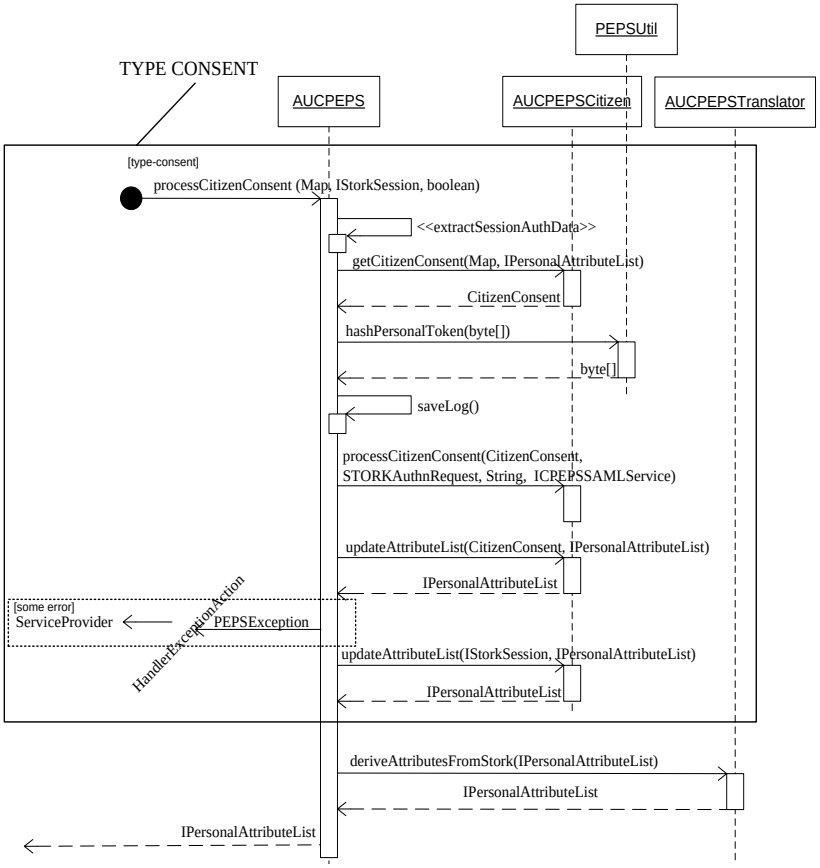
	• boolean
Output Returns	IPersonalAttributeList
Sequence Diagram	

Table 8: Authentication CPEPS Interfaces

2.5.3.3 Other methods

Methods	▪ **saveLog () : void** ▪ **sendErrorPage(IStorkSession, Map): void** ▪ generateSamlTokenFail(STORKAuthnRequest, PEPSErrors, IPUserAddress): String See AUSPEPS Other methods for description.

Table 9: Authentication CPEPS other methods

2.5.3.3.1 Components

2.5.3.3.1.1 Component Diagram

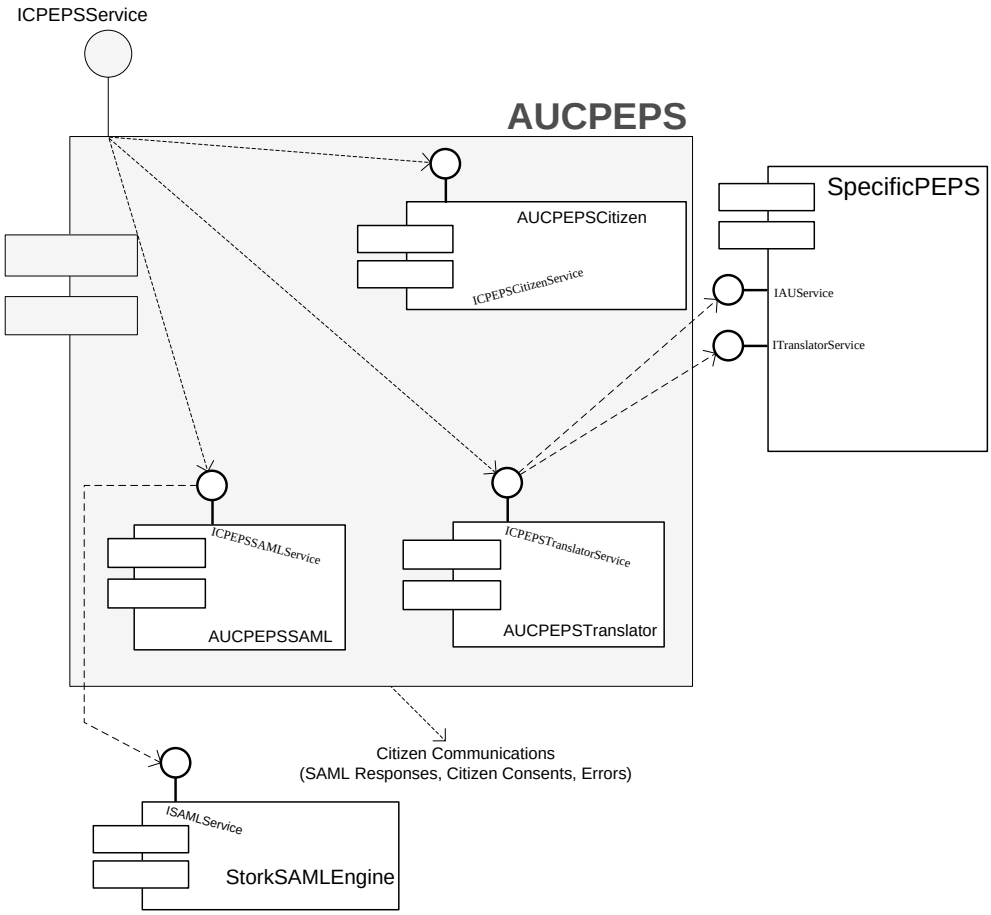


Figure 4: Authentication-CPEPS Component Diagram

2.5.3.3.1.2 AUCPEPSSAML component

2.5.3.3.1.2.1 Description

The AUCPEPSSAML component validates the SAML Authentication requests, creates and sends the Authentication responses back to Colleagues e/or IdPs. It also gets personal attributes from SAML Authentication requests.

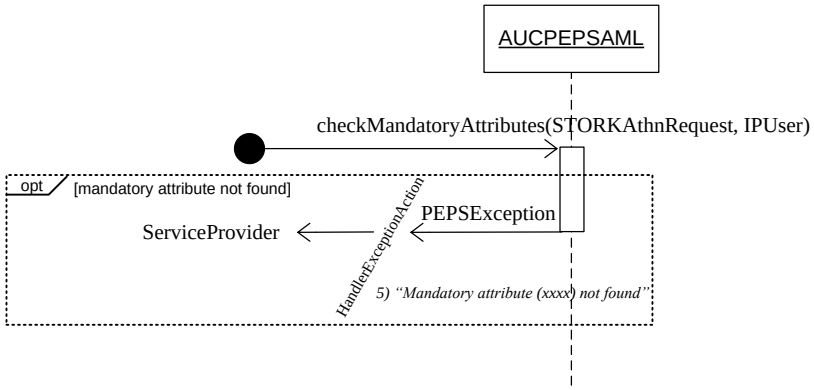
2.5.3.3.1.2.2 Interfaces

Interface Class	*ICPEPSSAMLService*
Methods	▪ **getSAMLToken** (*String*) : *byte[]* ▪ **processAuthenticationRequest** (*byte[]*,*IStorkSession*, *String*) : STORKAuthnRequest ▪ **generateAuthenticationResponse** (STORKAuthnRequest, String) : *byte[]* ▪ **generateErrorAuthenticationResponse (**STORKAuthnRequest, ErrorCode, ErrorSubCode, ErrorMsg, String**) : byte[]** See AUSPEPSManager Interfaces for description.

Table 10: Authentication CPEPSSAML component interface

2.5.3.3.1.2.2.1 Other Methods

Methods	▪ **checkMandatoryAttributes(STORKAuthnRequest, String): void** ▪ **checkAttributeValues(STORKAuthnRequest, String): void**	
	checkMandatoryAttributes(STORKAuthnRequest, String): void	
	Description	Checks if all mandatory attributes have values.
	Interface	• ICPEPSSAMLService
	Input Parameters	• STORKAuthnRequest • String -> IPUser
	Output Returns	Void
	Sequence Diagram	



	checkAttributeValues(STORKAuthnRequest, String): void	
	Description	Checks attribute values (e.g. gender, maritalStatus, …).
	Interface	• ICPEPSSAMLService
	Input Parameters	• STORKAuthnRequest • String -> IPUser
	Output Returns	Void
	Sequence Diagram	

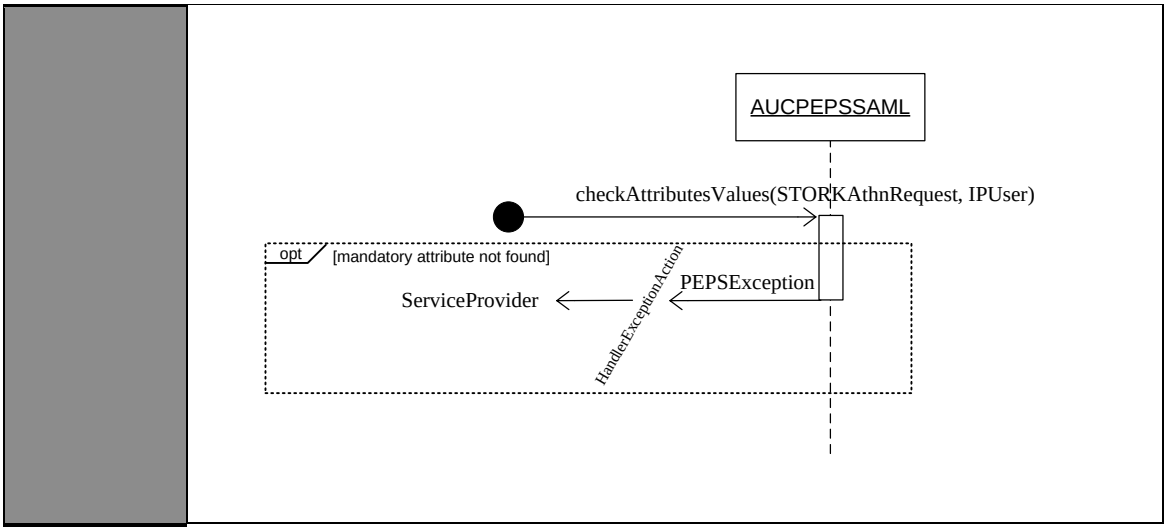


Table 11: Authentication CPEPSSAML component other methods.

2.5.3.3.1.3 AUCPEPSCitizen component

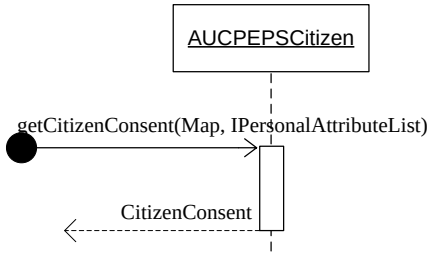
2.5.3.3.1.3.1 Description

The AUCPEPSCitizen component handles the Citizen Consent, and updates the attribute list state.

2.5.3.3.1.3.2 Interfaces

Interface Class	*ICPEPSCitizenService*	
Methods	▪ **getCitizenConsent (Map, IPersonalAttributeList) : CitizenConsent** ▪ **processCitizenConsent**(*CitizenConsent, STORKAuthnRequest, String, ICPEPSSAMLService*) : void ▪ **updateAttributeList(IStorkSession, IPersonalAttributeList)** : **IPersonalAttributeList** ▪ **updateAttributeList(CitizenConsent, IPersonalAttributeList)** : **IPersonalAttributeList** ▪ **updateAttributeListValues (IStorkSession, IPersonalAttributeList)** : **IPersonalAttributeList**	
	getCitizenConsent (Map, IPersonalAttributeList) : CitizenConsent	
	Description	Extract the Citizen Consent from the Citizen Response
	Interface	*ICPEPSCitizenService*
	Input Parameters	• Map • IPersonalAttributeList
	Output Returns	CitizenConsent

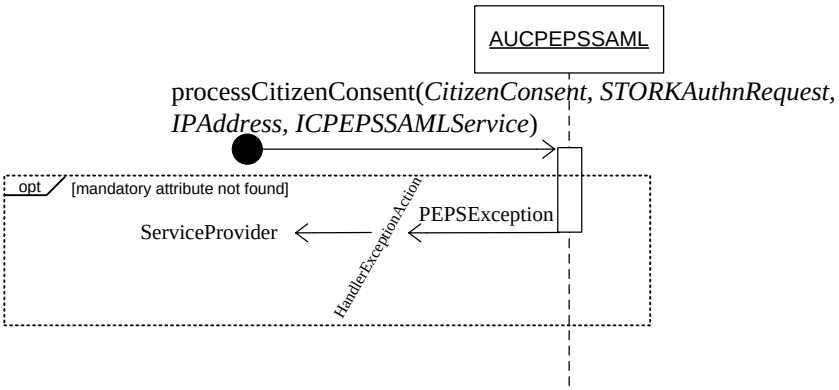
Sequence Diagram



processCitizenConsent(*CitizenConsent, STORKAuthnRequest, String, ICPEPSSAMLService*) : void

Description	Check Citizen Consents and mandatory attributes
Interface	*ICPEPSCitizenService*
Input Parameters	• CitizenConsent • STORKAuthnRequest • String -> IPUser • ICPEPSSAMLService
Output Returns	None

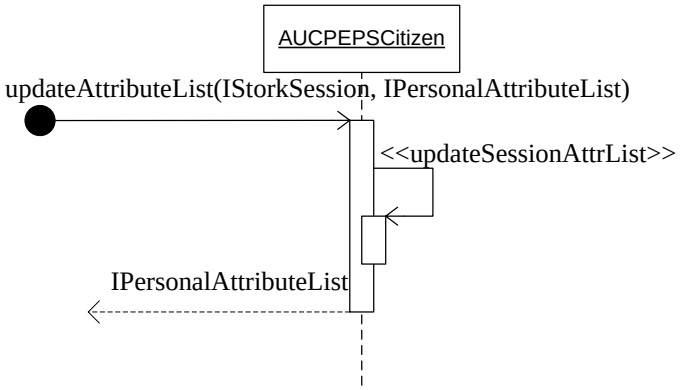
Sequence Diagram



updateAttributeList(IStorkSession, IPersonalAttributeList) : **IPersonalAttributeList**

Description	Updates the session attribute list.
Interface	*ICPEPSCitizenService*
Input Parameters	• IStorkSession • IPersonalAttributeList
Output Returns	IPersonalAttributeList

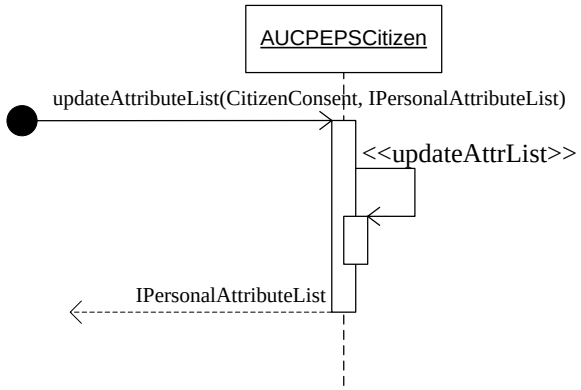
Sequence Diagram



- **updateAttributeList(CitizenConsent, IPersonalAttributeList)** : **IPersonalAttributeList**

Description	Updates the Attributes List with the given Citizen Consent.
Interface	*ICPEPSCitizenService*
Input Parameters	• CitizenConsent • IPersonalAttributeList
Output Returns	IPersonalAttributeList

Sequence Diagram



updateAttributeListValues (IStorkSession, IPersonalAttributeList) : **IPersonalAttributeList**

Description	Check Citizen Consents and mandatory attributes
Interface	*ICPEPSCitizenService*
Input Parameters	• IStorkSession • IPersonalAttributeList
Output Returns	IPersonalAttributeList

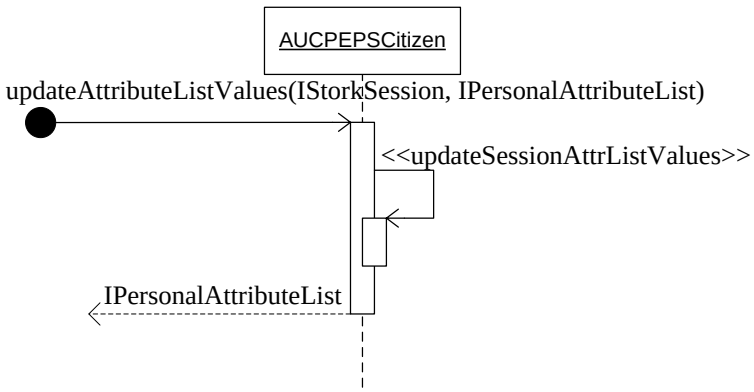
	Sequence Diagram 

Table 12: Authentication CPEPSCitizen component interface.

2.5.3.3.1.4 AUCPEPSTranslator component

2.5.3.3.1.4.1 Description

The AUCPEPSCitizen component translates the attribute names ans values to and from Stork format.

2.5.3.3.1.4.2 Interfaces

Interface Class	*ICPEPSTranslatorService*
Methods	▪ **normaliseAttributeNamesToStork (**IPersonalAttributeList**) : IPersonalAttributeList** ▪ **normaliseAttributeNamesFromStork (**IPersonalAttributeList**) : IPersonalAttributeList** ▪ **normaliseAttributeValuesToStork (**IPersonalAttributeList**) : IPersonalAttributeList** ▪ **normaliseAttributeValuesFromStork (**IPersonalAttributeList**) : IPersonalAttributeList** See AUSPEPSManager Interfaces for description.

Table 13: Authentication CPEPSTranslator component interface

2.5.3.3.1.4.3 Other Methods

Methods	▪ **deriveAttributesToStork(**ICPEPSSAMLService, IStorkSession, STORKAuthnRequest, String**): IPersonalAttributeList** ▪ **deriveAttributesFromStork(**IPersonalAttributeList**): IPersonalAttributeList**	
	▪ **deriveAttributesToStork(**ICPEPSSAMLService, IStorkSession, STORKAuthnRequest, String**): IPersonalAttributeList**	
	Description	Derives the attributes names to Stork Format.
	Interface	*ICPEPSTranslatorService*

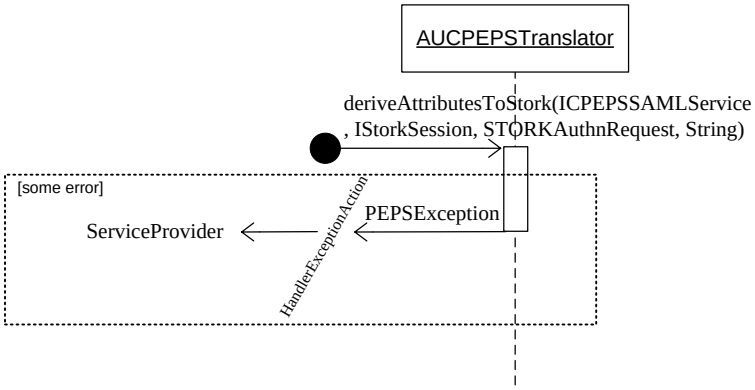
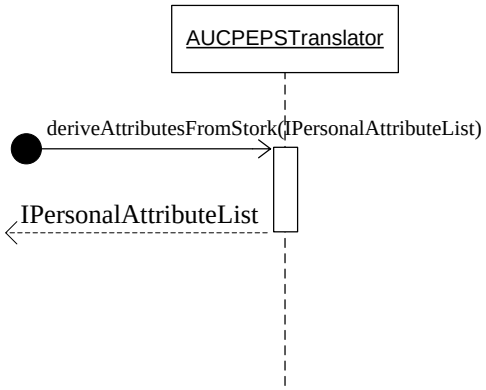
	Input Parameters	• ICPEPSSAMLService • IStorkSession • STORKAuthnRequest • String -> IPUser
	Output Returns	IPersonalAttributeList
	Sequence Diagram	
	▪ **deriveAttributesFromStork(IPersonalAttributeList): IPersonalAttributeList**	
	Description	Derives the attributes names from Stork Format.
	Interface	*ICPEPSTranslatorService*
	Input Parameters	IPersonalAttributeList
	Output Returns	IPersonalAttributeList
	Sequence Diagram	

Table 14: Authentication AUCPEPSSAML component other methods.

2.5.4 SpecificPEPS component

2.5.4.1 Description

SpecificPEPS is the module which each country must implement. At the agreed point, the common functionalities will activate the **IAUService** interface or **ITranslatorService** interface with a standard Java method invocation**,** which is part of the specific functionality.

2.5.4.2 Interfaces

<table>
<tr><td>Interface Class</td><td colspan="2">IAUService</td></tr>
<tr><td>Methods</td><td colspan="2">▪ prepareCitizenAuthentication(IPersonalAttributeList, Map, Map, IStorkSession): byte[]
▪ authenticateCitizen(IPersonalAttributeList, Map, Map): IPersonalAttributeList
▪ prepareAPRedirect(IPersonalAttributeList, Map, Map, IStorkSession): boolean
▪ getAttributesFromAttributeProviders(IPersonalAttributeList, Map, Map): IPersonalAttributeList
▪ getAttributesWithVerification(IPersonalAttributeList, Map, Map, IStorkSession, String) : boolean
▪ processAuthenticationResponse(byte[], IStorkSession): STORKAuthnResponse
▪ generateErrorAuthenticationResponse(String,String, String, String, String, String, String):byte[]
▪ comparePersonalAttributeLists(IPersonalAttributeList, IPersonalAttributeList): boolean</td></tr>
<tr><td></td><td colspan="2">▪ prepareCitizenAuthentication(IPersonalAttributeList, Map, Map, IStorkSession): byte[]</td></tr>
<tr><td></td><td>Description</td><td>Prepares the citizen to be redirected to the IdP.</td></tr>
<tr><td></td><td>Interface</td><td>IAUService</td></tr>
<tr><td></td><td>Input Parameters</td><td>• IPersonalAttributeList
• Map
• Map
• IStorkSession</td></tr>
<tr><td></td><td>Output Returns</td><td>byte []</td></tr>
<tr><td></td><td colspan="2">authenticateCitizen(IPersonalAttributeList, Map, Map): IPersonalAttributeList</td></tr>
<tr><td></td><td>Description</td><td>Get personal attributes from attribute providers</td></tr>
<tr><td></td><td>Interface</td><td>IAUService</td></tr>
<tr><td></td><td>Input Parameters</td><td>• IPersonalAttributeList
• Map
• Map</td></tr>
<tr><td></td><td>Output</td><td>boolean</td></tr>
</table>

	Returns	
	prepareAPRedirect(IPersonalAttributeList, Map, Map, IStorkSession**): boolean**	
	Description	Prepares the Citizen browser to be redirected to the AP.
	Interface	*IAUService*
	Input Parameters	• IPersonalAttributeList • Map • Map • IStorkSession
	Output Returns	boolean
	getAttributesFromAttributeProviders(IPersonalAttributeList, Map, Map): IPersonalAttributeList	
	Description	Returns the attributes values from the AP.
	Interface	*IAUService*
	Input Parameters	• IPersonalAttributeList • Map • Map
	Output Returns	**IPersonalAttributeList**
	getAttributesWithVerification(IPersonalAttributeList, Map, Map, IStorkSession, String**) :** Boolean	
	Description	Get the attributes from the AP with verification.
	Interface	*IAUService*
	Input Parameters	• IPersonalAttributeList • Map • Map • IStorkSession
	Output Returns	Boolean
	processAuthenticationResponse(byte[], IStorkSession**):** STORKAuthnResponse	
	Description	Validates a SAML Response.
	Interface	*IAUService*
	Input Parameters	• byte[] • IStorkSession
	Output Returns	**STORKAuthnResponse**
	generateErrorAuthenticationResponse(String,String, String, String, String, String, String**):byte[]**	
	Description	Generates a SAML Response in case of error.
	Interface	*IAUService*

	Input Parameters	• String -> The SAML's identifier to response. • String -> The issuer value. • String -> The assertion URL. • String -> The error code. • String -> The sub error code. • String -> The error message. • String -> The user IP address.
	Output Returns	• byte[]
	comparePersonalAttributeLists(IPersonalAttributeList, IPersonalAttributeList**) :** boolean	
	Description	Compares two given personal attribute lists.
	Interface	*IAUService*
	Input Parameters	• IPersonalAttributeList -> original • IPersonalAttributeList -> modified
	Output Returns	• Boolean
Interface Class	*ITranslatorService*	
Methods	▪ **normaliseAttributeNamesToStork(**I*PersonalAttributeList*) : I*PersonalAttributeList* ▪ **normaliseAttributeValuesToStork(**I*PersonalAttributeList*) : I*PersonalAttributeList* ▪ **normaliseAttributeNamesFromStork(**I*PersonalAttributeList*) : **I***PersonalAttributeList* ▪ **deriveAttributeFromStork**(IPersonalAttributeList): IPersonalAttributeList ▪ **deriveAttributeToStork**(IStorkSession, IPersonalAttributeList): IPersonalAttributeList ▪ **checkAttributeValues**(STORKAuthnRequest): boolean	
	normaliseAttributeNamesToStork(*PersonalAttributeList***) :** *PersonalAttributeList*	
	Description	Translates the attributes from local format to STORK format.
	Interface	*ITranslatorService*
	Input Parameters	• IPersonalAttributeList
	Output Returns	• IPersonalAttributeList
	normaliseAttributeValuesToStork(*PersonalAttributeList***) :** *PersonalAttributeList*	
	Description	Translates the attributes values from local format to STORK format.
	Interface	*ITranslatorService*
	Input Parameters	• IPersonalAttributeList
	Output Returns	• IPersonalAttributeList
	normaliseAttributeNamesFromStork (*PersonalAttributeList***) :** *PersonalAttributeList*	

	Description	Translates the attributes from STORK format to local format.
	Interface	*ITranslatorService*
	Input Parameters	• IPersonalAttributeList
	Output Returns	• IPersonalAttributeList
	deriveAttributeFromStork(IPersonalAttributeList): IPersonalAttributeList	
	Description	Derive Attribute Names To Stork format.
	Interface	*ITranslatorService*
	Input Parameters	• IPersonalAttributeList
	Output Returns	• IPersonalAttributeList
	deriveAttributeToStork(IStorkSession, IPersonalAttributeList): IPersonalAttributeList	
	Description	Derive Attribute Names from Stork format.
	Interface	*ITranslatorService*
	Input Parameters	• IStorkSession • IPersonalAttributeList
	Output Returns	• IPersonalAttributeList
	checkAttributeValues(STORKAuthnRequest): boolean	
	Description	Validate the values of the attributes.
	Interface	*ITranslatorService*
	Input Parameters	• STORKAuthnRequest
	Output Returns	• Boolean

Table 15: Specific PEPS component interfaces

2.5.5 SAMLEngine component

2.5.5.1 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.

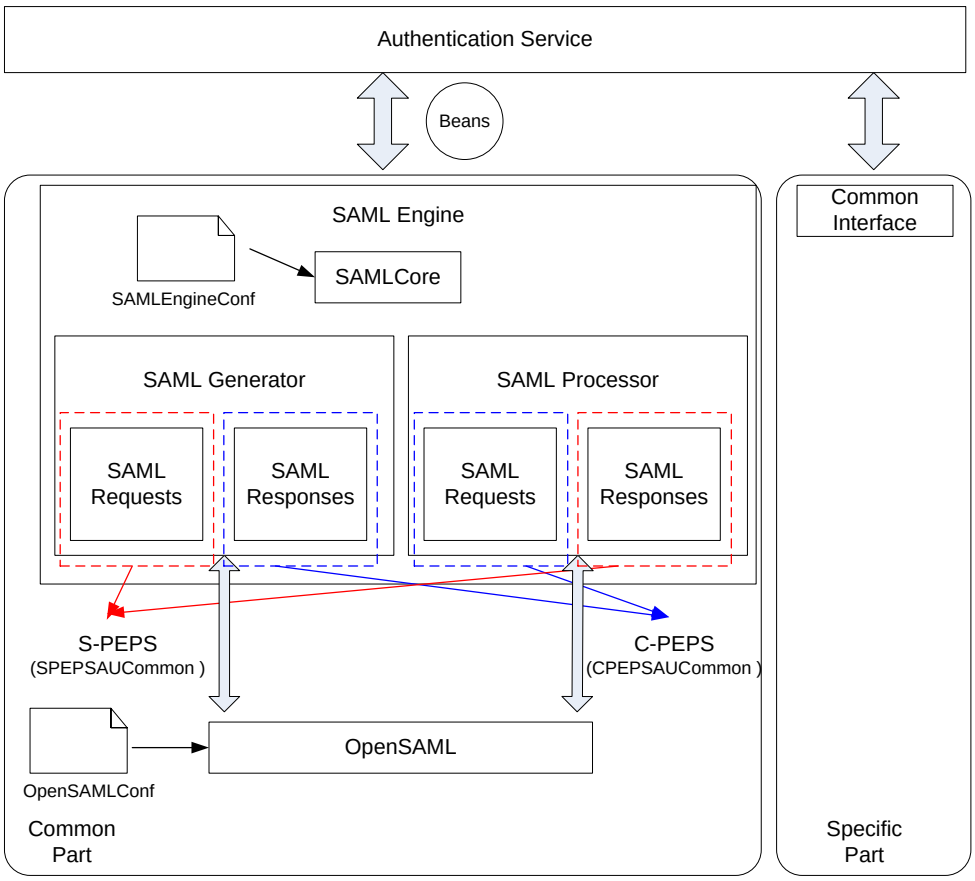


Figure 5 – Authentication/SAML engine: Model

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 is in charge of generating SAML Tokens, which can be either SAML Authentication (and Attribute query as an extension) requests or SAML Assertions (Authentication and Attribute statements) responses.

- SAML Processor

  This part of the engine is in charge of validating and processing 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

2.5.5.2 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.

2.5.5.2.1 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.

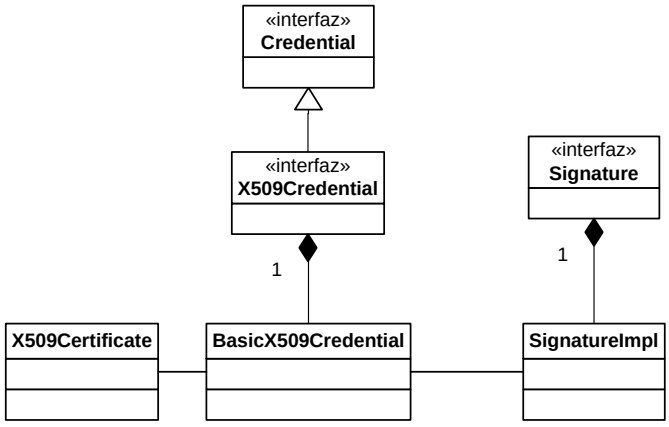


Figure 6 – OpenSAML Class Diagram for XML signature generation purposes

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

2.5.5.2.1.1 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.

2.5.5.2.1.2 org.opensaml.xml.signature.impl.SignatureImpl

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

2.5.5.2.1.3 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).

2.5.5.2.1.4 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.

2.5.5.2.1.5 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.

2.5.5.2.1.6 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.

2.5.5.2.2 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.

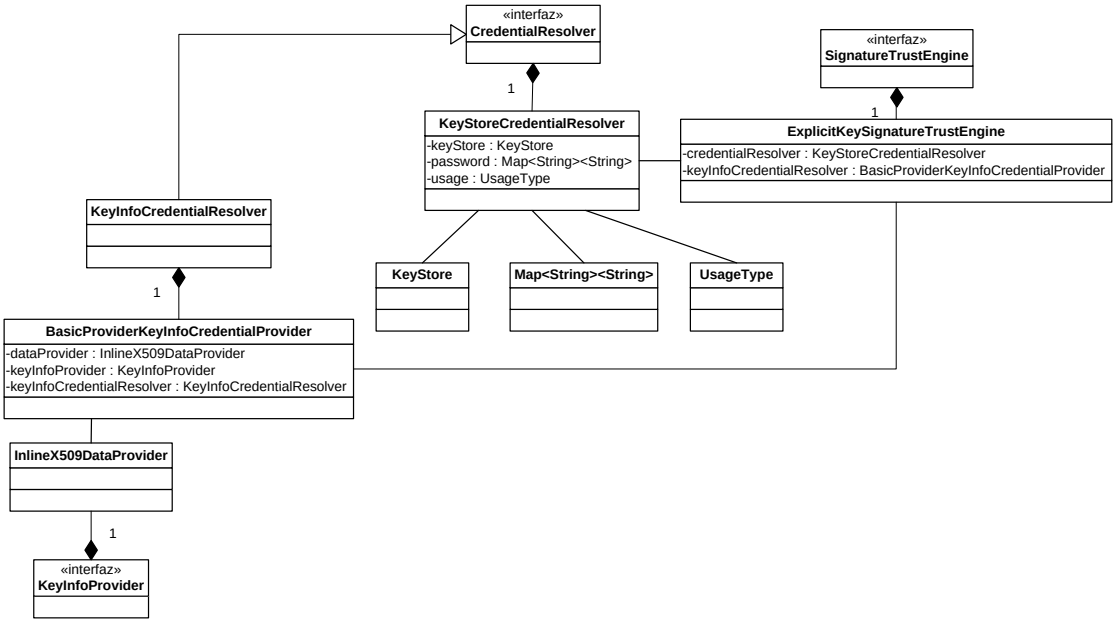


Figure 7 – OpenSAML Class Diagram for XML signature verification purposes

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

2.5.5.2.2.1 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.

2.5.5.2.2.2 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*.

2.5.5.2.2.3 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.

2.5.5.2.2.4 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.

2.5.5.2.2.5 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 4.1.4).

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.

2.5.5.3 Methods

<table>
<tr><td>Methods</td><td colspan="2">

generateSTORKAuthnRequest(STORKAuthnRequest): STORKAuthnRequest
generateSTORKAuthnResponse(STORKAuthnRequest,STORKAuthnResponse,string boolean): STORKAuthnResponse
generateSTORKAuthnResponseFail(STORKAuthnRequest,STORKAuthnResponse, string, boolean): STORKAuthnResponse
validateSTORKAuthnRequest(byte[]): STORKAuthnRequest
validateSTORKAuthnResponse(byte[],string): STORKAuthnResponse

</td></tr>
<tr><td></td><td colspan="2">generateSTORKAuthnRequest(STORKAuthnRequest): STORKAuthnRequest</td></tr>
<tr><td></td><td>Description</td><td>Generates a STORK Authentication Request object</td></tr>
<tr><td></td><td>Interface</td><td></td></tr>
<tr><td></td><td>Input Parameters</td><td>• STORKAuthnRequest</td></tr>
<tr><td></td><td>Output Returns</td><td>• STORKAuthnRequest</td></tr>
<tr><td></td><td colspan="2">generateSTORKAuthnResponse(STORKAuthnRequest,STORKAuthnResponse,string boolean): STORKAuthnResponse</td></tr>
<tr><td></td><td>Description</td><td>Generates a STORK Authentication Response object</td></tr>
<tr><td></td><td>Interface</td><td></td></tr>
<tr><td></td><td>Input Parameters</td><td>• STORKAuthnRequest
• STORKAuthnResponse
• String → ipAddress (of the Citizen Browser)
• Boolean → isHashing (yes: hashing attribute values)</td></tr>
<tr><td></td><td>Output Returns</td><td>• STORKAuthnResponse</td></tr>
<tr><td></td><td colspan="2">generateSTORKAuthnResponseFail(STORKAuthnRequest,STORKAuthnResponse, string, boolean): STORKAuthnResponse</td></tr>
<tr><td></td><td>Description</td><td>Generates a STORK Authentication Response failure object</td></tr>
<tr><td></td><td>Interface</td><td></td></tr>
<tr><td></td><td>Input Parameters</td><td>• STORKAuthnRequest
• STORKAuthnResponse
• String → ipAddress (of the Citizen Browser)
• Boolean → isHashing (yes: hashing attribute values)</td></tr>
<tr><td></td><td>Output Returns</td><td>• STORKAuthnResponse</td></tr>
<tr><td></td><td colspan="2">validateSTORKAuthnRequest(byte[]): STORKAuthnRequest</td></tr>
<tr><td></td><td>Description</td><td>Validates a STORK SAML Authentication Request token</td></tr>
<tr><td></td><td>Interface</td><td></td></tr>
<tr><td></td><td>Input Parameters</td><td>• byte[] → SAML token</td></tr>
<tr><td></td><td>Output</td><td>• STORKAuthnRequest</td></tr>
</table>

	Returns	
	validateSTORKAuthnResponse(*byte[],string*): STORKAuthnResponse	
	Description	Validates a STORK SAML Authentication Response token
	Interface	
	Input Parameters	• byte[] → SAML token
	Output Returns	• STORKAuthnResponse

Table 16: SAML Component interfaces

2.5.5.4 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.

2.5.5.4.1 Basic Class Diagram

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

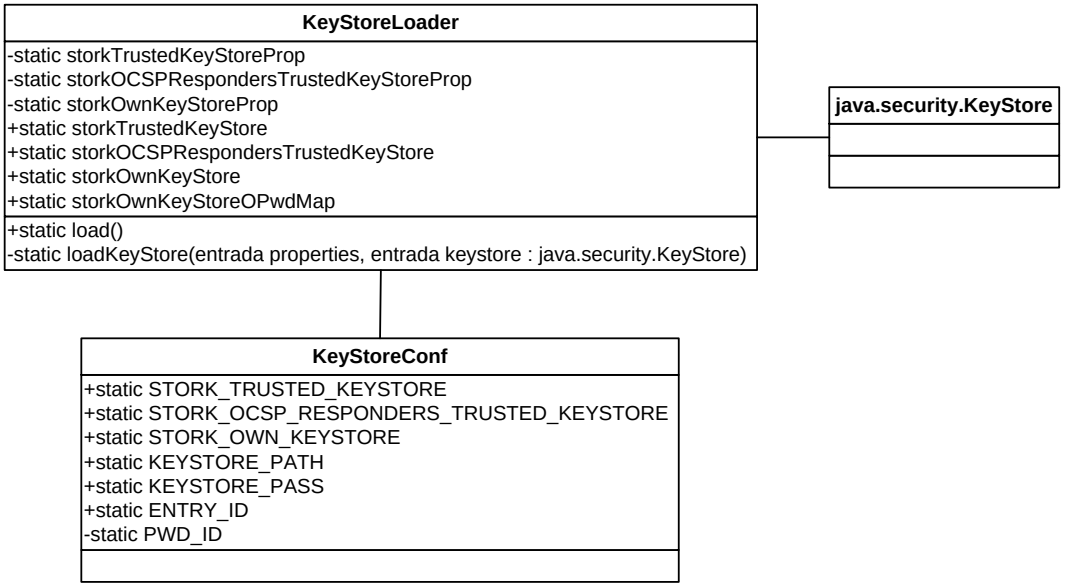


Figure 8 – KeyStore Management Classes

2.5.5.4.1.1 eu.stork.peps.keystoresKeyStoreLoader Class

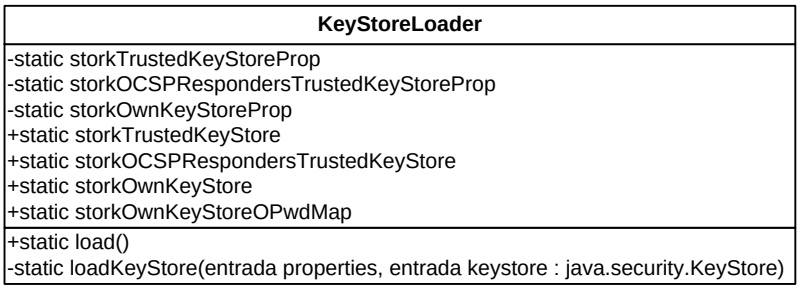


Figure 9 – KeyStoreLoader Class

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.

2.5.5.4.1.2 eu.stork.peps.keystoresKeyStoreConf Class

KeyStoreConf
+static STORK_TRUSTED_KEYSTORE +static STORK_OCSP_RESPONDERS_TRUSTED_KEYSTORE +static STORK_OWN_KEYSTORE +static KEYSTORE_PATH +static KEYSTORE_PASS +static ENTRY_ID -static PWD_ID

Figure 10 – KeyStoreConf Class

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

2.5.6 ValidationPEPS

2.5.6.1 Description

2.5.7 Validation Engine

The Validation Service is in charge of implementing the logic of the PEPS validation service, both for S-PEPS and for the C-PEPS. The functionality is split into the common validation part and the specific validation part.

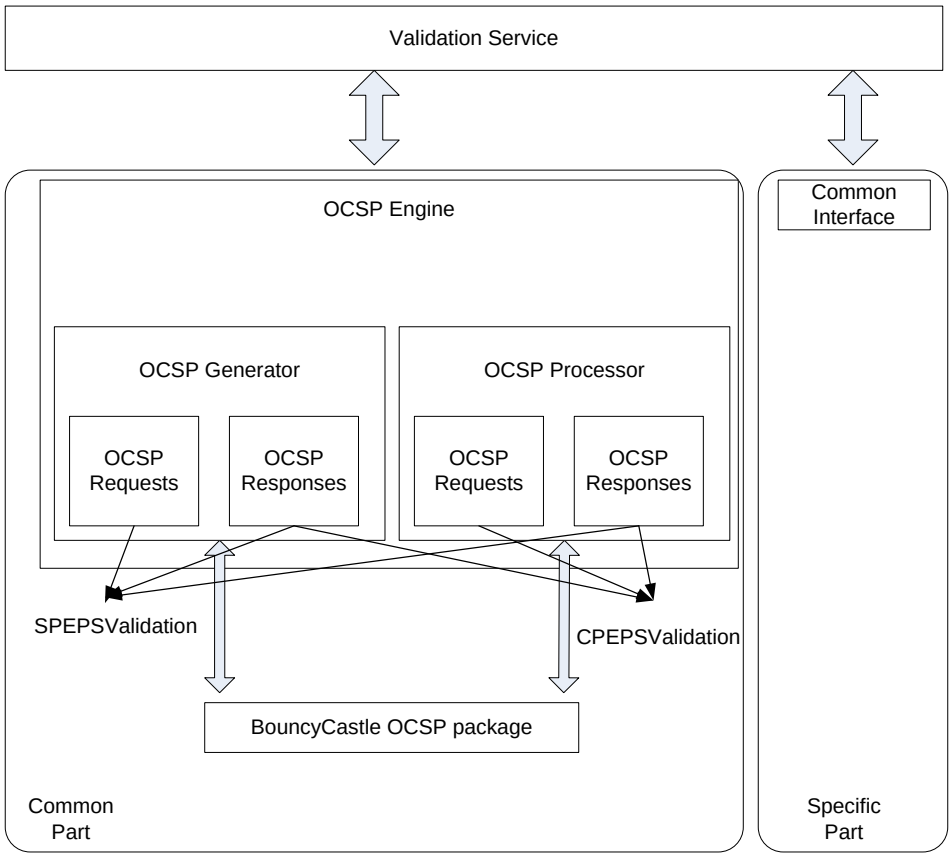


Figure 11 - Validation/ OCSP Engine: Model

The OCSP Engine is responsible for implementing the operations (like issuing, creation, a.s.o) on the OCSP messages, both on the OCSP requests and on the OCSP responses.

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

The C-PEPS functionality is covered by the OCSP Requests and the OCSP Responses parts of the OCSP Processor and the OCSP Responses of the OCSP Generator. That is, the C-PEPS will process the requests received from the S-PEPS, the responses received from the Specific Validation Engine and it will generate OCSP responses to be sent back to the S-PEPS.

The OCSP Engine manages the OCSP objects by means of the BouncyCastle OCSP library.

This library containes several classes that can be used by the OCSP Engines, such the BasicOCSPResp (to create basic OCSP response messages), OCSPReqGenerator to generate OCSP requests, or OCSPRespGenerator to generate OCSP responses.

Additional functions will have to be implemented in the OCSP Engine to create wrap the OCSP into HTTP messages (create the so-called Http-based OCSP requests and Http-based OCSP response messages as defined in the Appendix 1 of the RFC2560) and to validate the OCSP requests and the OCSP responses.

The ValidationPEPS component is in charge with implementing the business logic of the PEPS validation service. The following figure illustrates a high level view of the architecture used for the validation service in STORK.

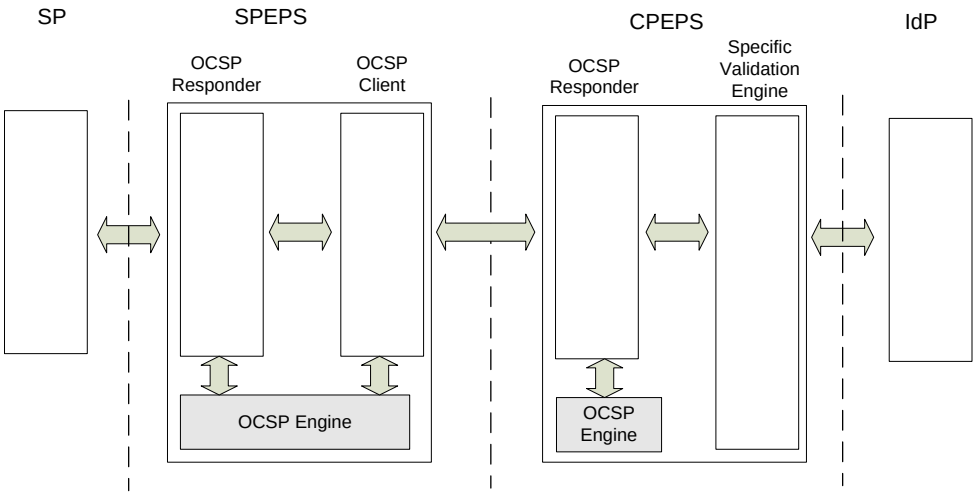
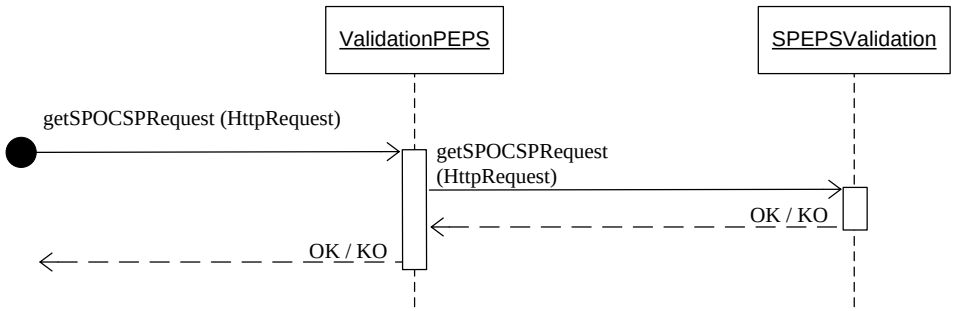


Figure 12: Validation PEPS Architecture

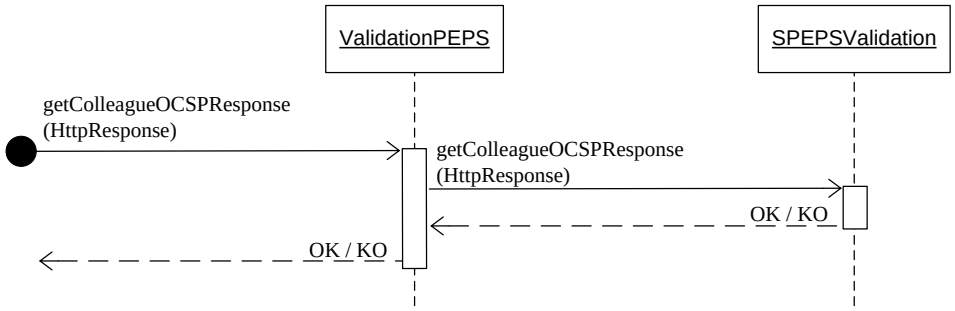
2.5.7.1 Interfaces

Interface Class	*IValidationService*	
Methods	▪ **getSPOCSPRequest (***HttpRequest OCSPRequest***) :** *boolean* ▪ **getColleagueOCSPResponse (***HttpResponse OCSPResponse***) :** *boolean* ▪ **getColleagueOCSPRequest (***HttpRequest OCSPRequest***) :** *boolean*	
	getSPOCSPRequest (*HttpRequest OCSPRequest***) :** *boolean*	
	Description	Gets an OCSP request from the SP
	Interface	*IValidationService*

Input Parameters	• OCSP request
Output Returns	• OCSP request or Error
Sequence Diagram	



getColleagueOCSPResponse(*HttpResponse OCSPResponse***) :** *boolean*	
Description	Gets an OCSP response from the colleague PEPS.
Interface	*IValidationService*
Input Parameters	• OCSPResponse
Output Returns	• OCSP Response or Error
Sequence Diagram	



getColleagueOCSPRequest (*HttpRequest OCSPRequest***) :** *boolean*	
Description	Gets an OCSP response from the colleague PEPS.
Interface	*IValidationService*
Input Parameters	• OCSPRequest
Output Returns	• Boolean (True: no errors / False: some error)

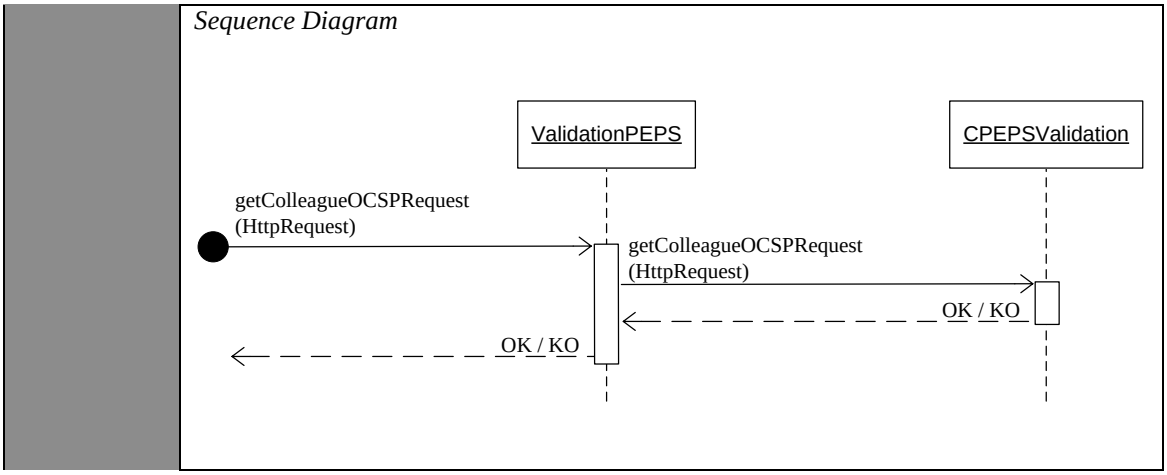


Table 17: Validation PEPS Interfaces

2.5.7.2 Components

2.5.7.2.1 Component Diagram

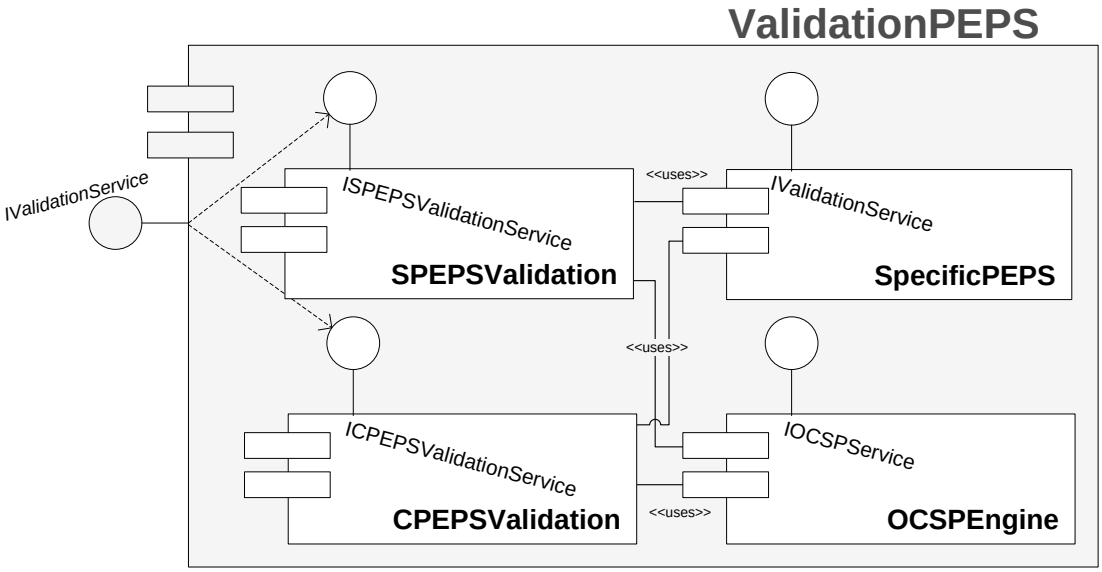


Figure 13: Validation-PEPS Component Diagram

2.5.7.2.2 SPEPSValidation component

2.5.7.2.2.1 Description

2.5.7.2.2.2 Interfaces

Interface Class	*ISPEPSValidationService*	
Methods	▪ **getOCSPRequest (***HttpRequest OCSPRequest***) : *boolean*** ▪ **getColleagueOCSPResponse (***HttpResponse OCSPResponse***) : *OCSP Response***	
	getOCSPRequest (*HttpRequest OCSPRequest***) : *OCSP Request***	
	Description	Gets an OCSP request from the SP.
	Interface	*ISPEPSValidationService*

	Input Parameters	• HttpRequest
	Output Returns	• Boolean (True: no errors / False: some error)
	Sequence Diagram	See 2.5.7.2.2.4
	getOCSPResponse (*HttpResponse OCSPResponse***) :** *boolean*	
	Description	
	Interface	*ISPEPSValidationService*
	Input Parameters	• HttpResponse
	Output Returns	• Boolean (True: no errors / False: some error)
	Sequence Diagram	See 2.5.7.2.2.4

Table 18: Validation SPEPS Interfaces

2.5.7.2.2.3 Other methods

Methods	▪ **sendOCSPResponse** (*HttpResponse OCSPResponse*) **:** *boolean* ▪ **sendOCSPRequest** *(HttpRequest OCSPRequest) : boolean*	
	sendOCSPResponse (*HttpResponse OCSPResponse*) **:** *boolean*	
	Description	Sends an OCSP response to the SP.
	Interface	-
	Input Parameters	• OCSPResponse
	Output Returns	• Boolean (True: no errors / False: some error)
	Sequence Diagram	See 2.5.7.2.2.4
	sendOCSPRequest *(HttpRequest OCSPRequest) : boolean*	
	Description	Send an OCSP request to the colleague PEPS.
	Interface	-
	Input Parameters	• OCSP Request
	Output Returns	• Boolean (True: no errors / False: some error)
	Sequence Diagram	See page 2.5.7.2.2.4

Table 19: Validation SPEPS other methods

2.5.7.2.2.4 Sequence Diagram

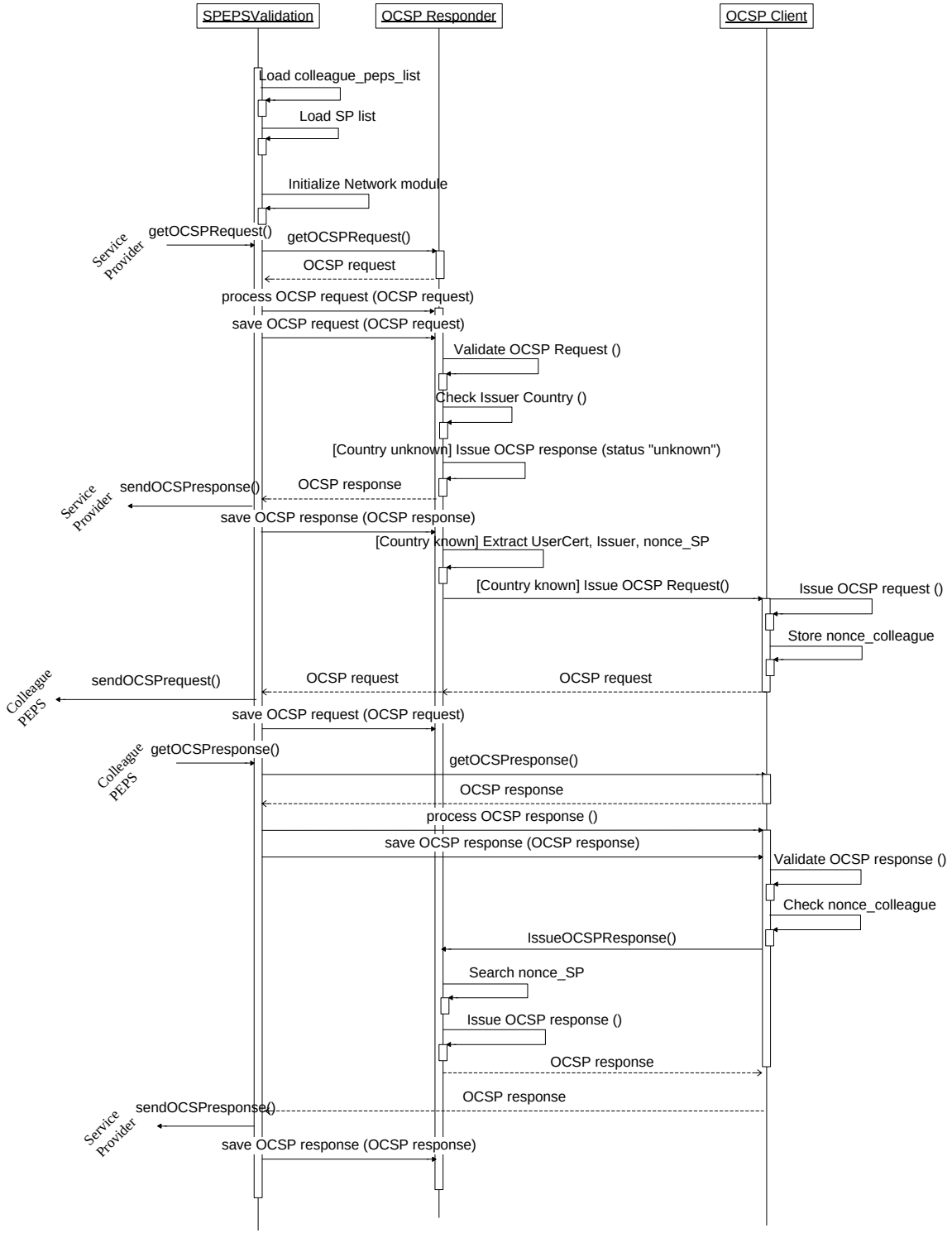


Figure 14: Validation-PEPS Sequence Diagram

2.5.7.2.2.5 Components

2.5.7.2.2.5.1 Component Diagram

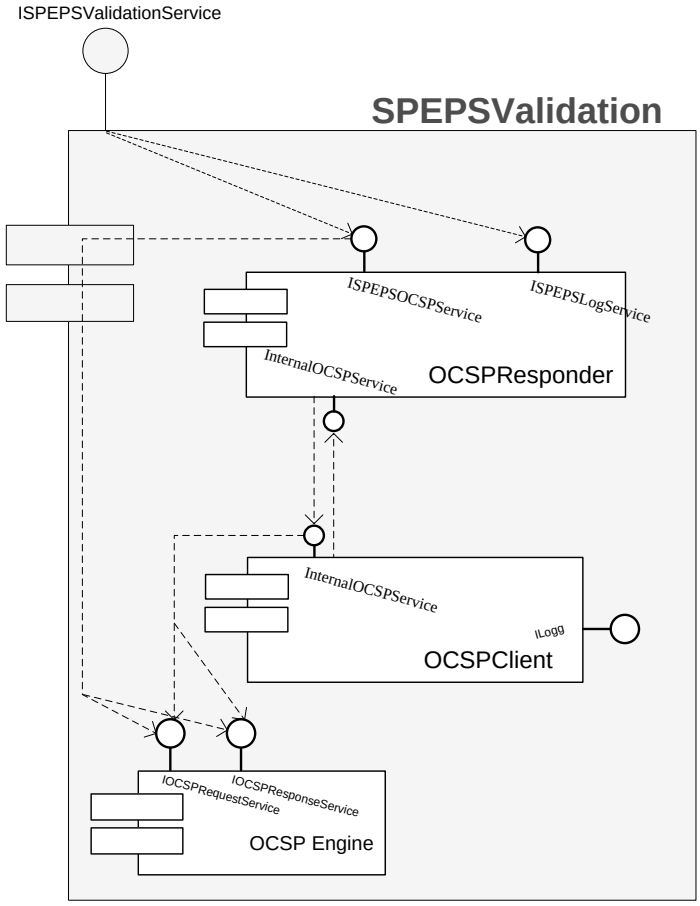


Figure 15: Validation-SPEPS Component Diagram

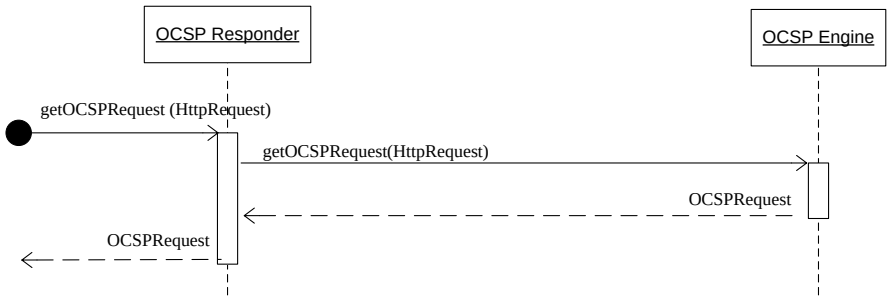
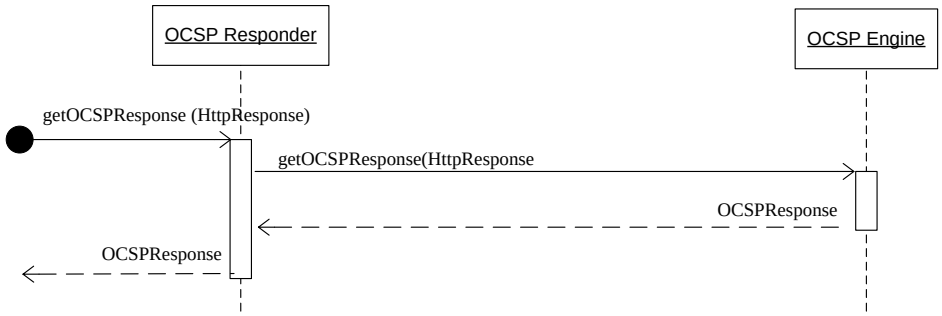
2.5.7.2.2.5.2 OCSPResponder component

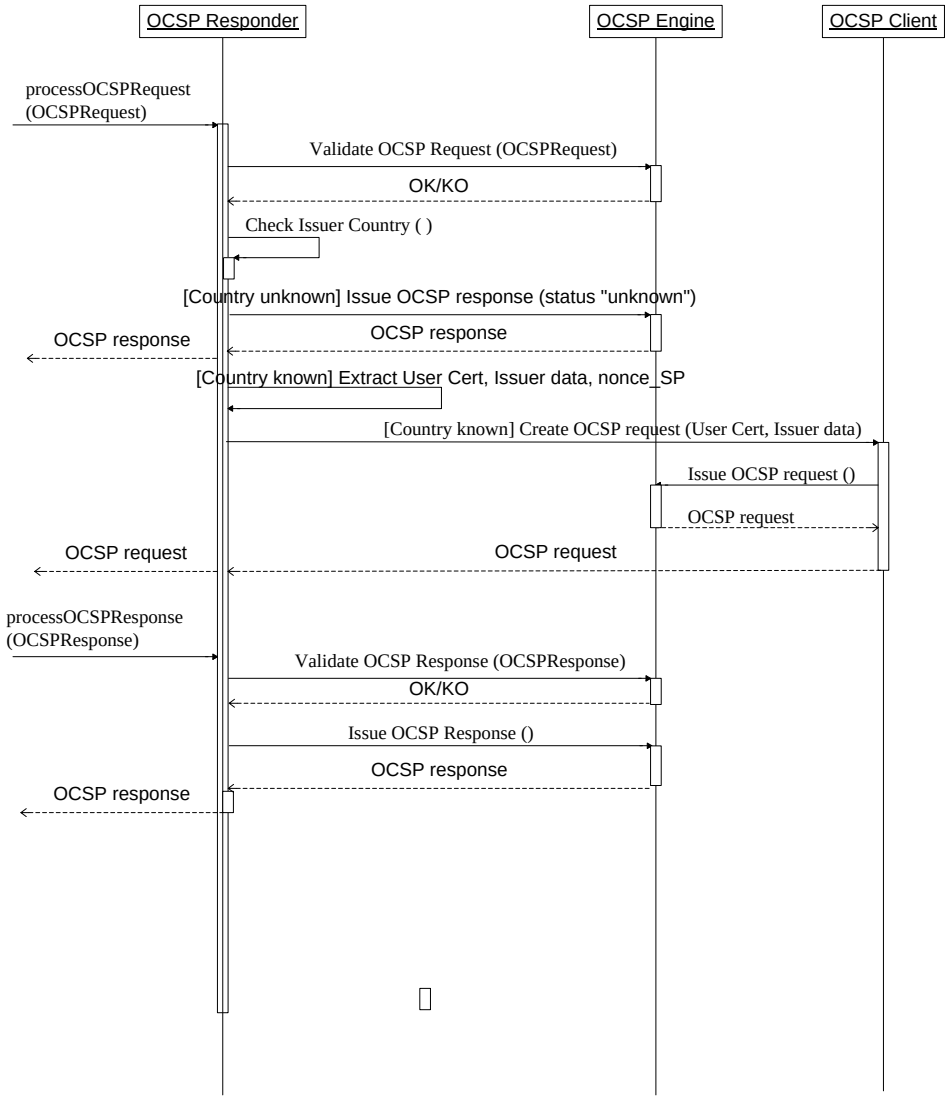
2.5.7.2.2.5.2.1 Description

This component is in charge with processing the OCSP requests received from the SP and the OCSP responses received from the colleague PEPS. In addition, it is in charge with generating the OCSP responses to be sent back by the SPEPSValidation to the SP.

2.5.7.2.2.5.2.2 Interfaces

Interface Class	*ISPEPSOCSPService*	
Methods	▪ **getOCSPRequest** (*HttpRequest OCSPRequest*) : *OCSPRequest* ▪ **processOCSPRequest** (*OCSPRequest*) : *OCSP request or OCSP response* ▪ **processOCSPResponse (***OCSPRequest, colleague_peps_list)*: boolean ▪ **getOCSPResponse** *(HttpResponse OCSPResponse*) : *OCSPResponse*	
	getOCSPRequest (*HttpRequest OCSPRequest*) : *OCSPRequest*	
	Description	Gets the OCSP request from the SP
	Interface	*ISPEPSOCSPService*
	Input	OCSP request

	Parameters	
	Output Returns	*OCSPRequest*
	Sequence Diagram	
	getOCSPResponse (*HttpResponse OCSPresponse*) : *OCSPResponse*	
	Description	Gets the OCSP response from the colleague PEPS.
	Interface	*ISPEPSOCSPService*
	Input Parameters	HttpResponse
	Output Returns	*OCSPResponse*
	Sequence Diagram	
	processOCSPRequest (*OCSPRequest*)**:** *OCSPRequest or OCSPResponse*	
	Description	Process the OCSP request received from the SP.
	Interface	*ISPEPSOCSPService*
	Input Parameters	*OCSPRequest*
	Output Returns	• OCSPRequest, or • OCSPResponse, or • some error

	Sequence Diagram 
	processOCSPResponse(*OCSPResponse***):** *OCSPResponse*
	Description: Processes the OCSP response received from the colleague PEPS.
	Interface: *ISPEPSOCSPService*
	Input Parameters: OCSPResponse
	Output Returns: OCSPResponse
	Sequence Diagram See ProcessOCSPRequest.
Interface Class	*ISPEPSLoggService*
Methods	▪ **saveOCSPRequest** (*OCSPRequest*) : *boolean* ▪ **saveOCSPResponse** (*OCSPResponse*) : *boolean*
	saveOCSPRequest (*OCSPObject*) : *boolean*

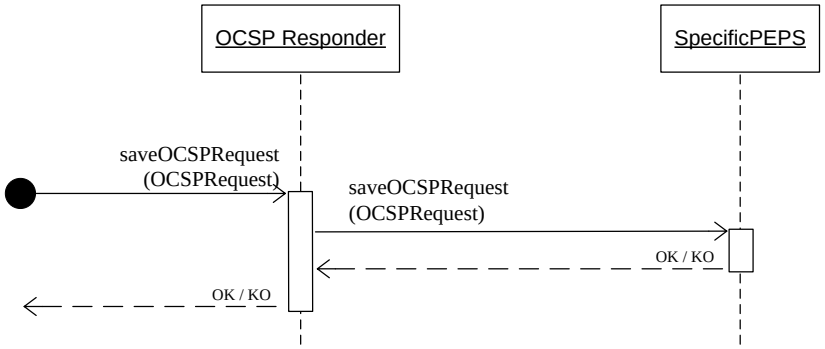
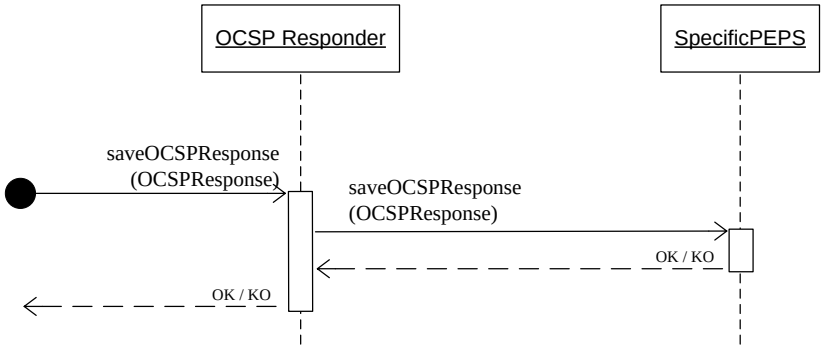
	Description	
	Interface	*ISPEPSLoggService*
	Input Parameters	OCSPObject
	Output Returns	Boolean (True: no errors / False: some error)
	Sequence Diagram	
	saveOCSPResponse (*OCSPObject*) : *boolean*	
	Description	Log operation.
	Interface	*ISPEPSLoggService*
	Input Parameters	OCSPResponse
	Output Returns	Boolean (True: no errors / False: some error)
	Sequence Diagram	

Table 20: SPEPSValidation- OCSPResponder component interface

2.5.7.2.2.5.2.3 Other Methods

Methods	▪ **Load colleague_peps_list (colleague_peps_list): boolean** ▪ **Load SP list** *(SP list): Boolean* ▪ **Load S-PEPS certificate and key** *(S-PEPS certificate and key file): boolean* ▪ **Load S-PEPS certificate and key for OCSP request signing** *(S-PEPS OCSP req signing certificate and key file): boolean* ▪ **Check Issuer Country** *(OCSPRequestMessage, colleague_peps_list): boolean* ▪ **Store nonce_SP** *(OCSP request) : boolean* ▪ **Search nonce_SP** *(nonce list): nonce* ▪ **Initialize Network module (IPaddress, port number)** **Extract UserCert_Issuer_nonce_colleague** *(OCSPRequest): validationStructure*	
	Load colleague_peps_list *(colleague_peps_list): boolean*	
	Description	Loads the colleague peps list
	Interface	
	Input Parameters	colleague_peps_list
	Output Returns	Boolean (True: no errors / False: some error)
	Sequence Diagram	See 2.5.7.2.3.4
	Load SP list *(SP list): boolean*	
	Description	Loads the SP list
	Interface	
	Input Parameters	SP list
	Output Returns	Boolean (True: no errors / False: some error)
	Sequence Diagram	See 2.5.7.2.3.4
	Load S-PEPS certificate and key *(S-PEPS certificate and key file): boolean*	
	Description	Loads the X.509v3 certificate of S-PEPS (with extension OCSP signing) and the corresponding key required for signing OCSP responses.
	Interface	
	Input Parameters	S-PEPS certificate (with extension OCSP signing) and the corresponding key.
	Output Returns	Boolean (True: no errors / False: some error)
	Sequence Diagram	See 2.5.7.2.3.4
	Load S-PEPS certificate and key for OCSP request signing *(S-PEPS OCSP req signing certificate and key file): boolean*	
	Description	Loads the X.509v3 certificate of S-PEPS and the corresponding key used for signing the OCSP requests sent to the colleague PEPS.

	Interface	
	Input Parameters	S-PEPS' OCSP request signing certificate and the corresponding key.
	Output Returns	Boolean (True: no errors / False: some error)
	Sequence Diagram	See 2.5.7.2.3.4
	Check Issuer Country (*OCSPRequest, colleague_peps_list): boolean*	
	Description	Checks whether the Issuer Country is known
	Interface	
	Input Parameters	• OCSP Request • colleague peps list
	Output Returns	Boolean (True: no errors / False: some error)
	Sequence Diagram	See 2.5.7.2.3.4
	Store nonce_SP *(OCSP request) : nonce list*	
	Description	Stores the nonce value in the OCSP request received from the SP.
	Interface	
	Input Parameters	• OCSP Request • colleague peps list
	Output Returns	Nonce list, or error.
	Sequence Diagram	See 2.5.7.2.3.4
	Search nonce_SP *(nonce list): nonce*	
	Description	Searches the nonce value corresponding to the OCSP request received from a particular SP.
	Interface	
	Input Parameters	nonce list
	Output Returns	Nonce, or error
	Sequence Diagram	See 2.5.7.2.3.4
	Initialize Network module (OCSPResponderIPaddress, portnumber): boolean	
	Description	Initializes the network parameters (e.g. IP address, port number) used by the SPEPSValidation module to communicate with the SP and the colleague PEPS.
	Interface	
	Input Parameters	IP address and port number of the OCSP responder of the SPEPS.

	Output Returns	Boolean (True: no errors / False: some error)
	Sequence Diagram	See 2.5.7.2.3.4
	Extract UserCert_Issuer_nonce_colleague *(OCSPRequest): validationStructure*	
	Description	Extracts from the OCSP request the user certificate, the Issuer data and the nonce value used in the communication with the colleague PEPS.
	Interface	
	Input Parameters	OCSP Request
	Output Returns	validationStructure composed of the user certificate data, Issuer data and nonce_colleague value
	Sequence Diagram	See 2.5.7.2.3.4

Table 21: SPEPSValidation- OCSPResponder component other methods

2.5.7.2.2.5.3 OCSPClient component

2.5.7.2.2.5.3.1 Description

This component is in charge with creating OCSP requests responses to be sent by the SPEPSValidation module to the Colleague PEPS.

2.5.7.2.2.5.3.2 Interfaces

Interface Class	*InternalOCSPService*	
Methods	▪ **Create OCSP Request** (*User Cert, Issuer data, PEPS key*) : *OCSPRequest*	
	Create OCSP Request (*User Cert, Issuer data, PEPS key*) : *OCSPRequest*	
	Description	Creates the OCSP request to be sent to the colleague PEPS
	Interface	*InternalOCSPService*
	Input Parameters	• User Cert data • Issuer data • PEPS key used for OCSP request signing
	Output Returns	OCSPRequest

	Sequence Diagram

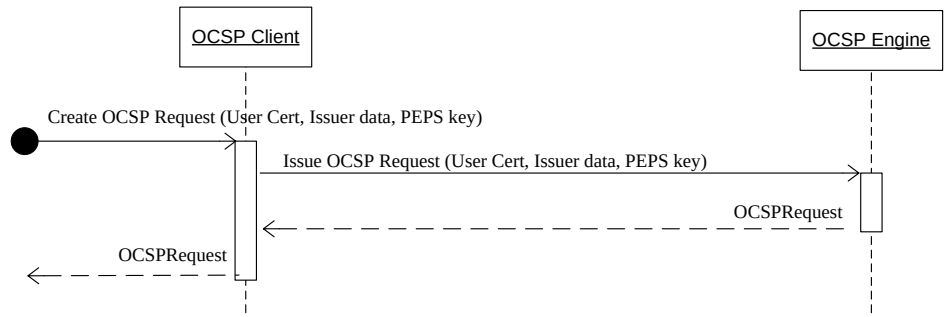


Table 22: SPEPSValidation- OCSPClient component interface

2.5.7.2.3 CPEPSValidation component

2.5.7.2.3.1 Description

2.5.7.2.3.2 Interfaces

Interface Class	*ICPEPSOCSPService*	
Methods	▪ **getOCSPRequest (***HttpRequest OCSPRequest***) :** *boolean*	
	getOCSPRequest (*HttpServletRequest , HttpServletResponse***) :** *boolean*	
	Description	Gets the OCSP Request from the colleague PEPS.
	Interface	*ICPEPSOCSPService*
	Input Parameters	• HttpRequest
	Output Returns	• Boolean (True: no errors / False: some error)
	Sequence Diagram	See 2.5.7.2.3.4

Table 23: Validation- CPEPSValidation Interfaces

2.5.7.2.3.3 Other methods

Methods	▪ **sendOCSPResponse** (*HttpResponse OCSPResponse*) **:** *boolean*	
	sendValidationResponse (*OCSPResponseMessage, HttpServletResponse***) :** *boolean*	
	Description	Sends the OCSP response to the colleague PEPS
	Interface	-
	Input Parameters	• HttpResponse
	Output Returns	• Boolean (True: no errors / False: some error)
	Sequence Diagram	See 2.5.7.2.3.4

Table 24: Validation- CPEPSValidation other methods

2.5.7.2.3.4 Sequence Diagram

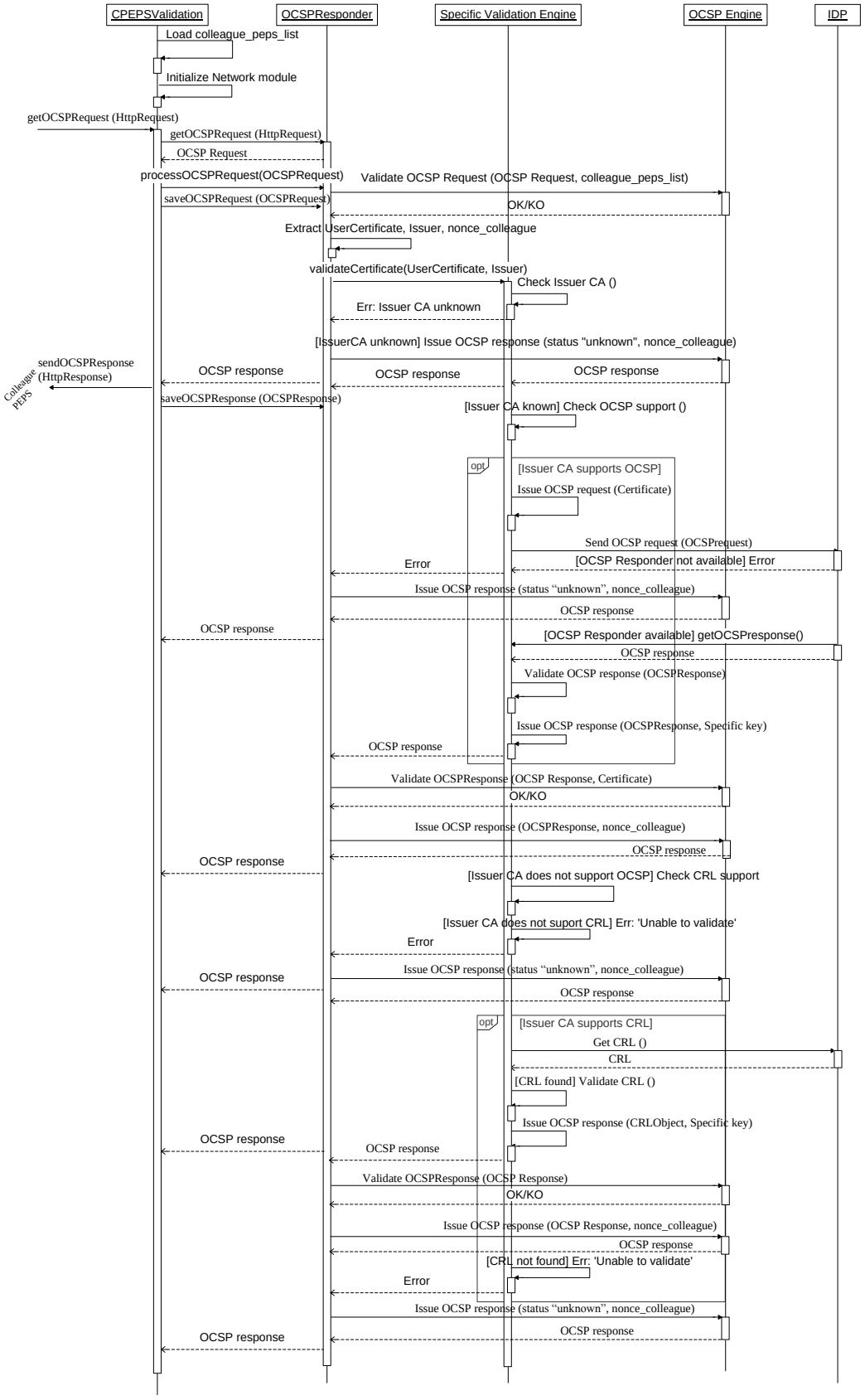


Figure 16: Validation-CPEPS Sequence Diagram

2.5.7.2.3.5 Components

2.5.7.2.3.5.1 Component Diagram

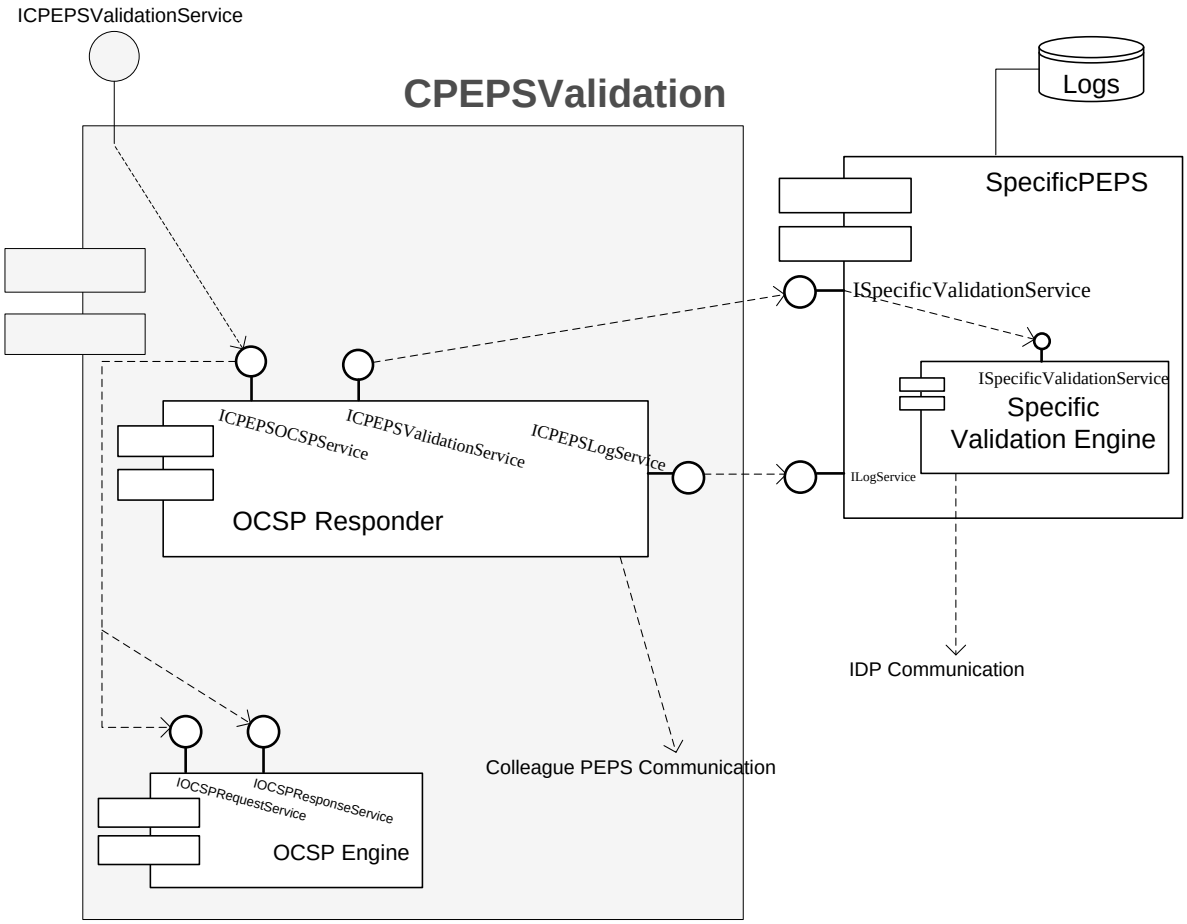


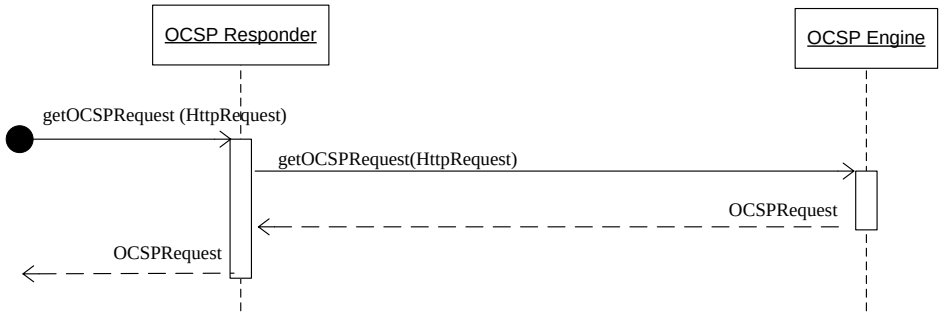
Figure 17: Validation-CPEPS Component Diagram

2.5.7.2.3.5.2 OCSPResponder component

2.5.7.2.3.5.2.1 Description

2.5.7.2.3.5.2.2 Interfaces

Interface Class	*ICPEPSOCSPService*	
Methods	▪ **getOCSPRequest** (*HttpRequest OCSPRequest*) : *OCSPRequest* ▪ **processOCSPRequest** (*OCSPRequest*) : *boolean*	
	getOCSPRequest (*HttpRequest OCSPRequest*) : *OCSPRequest*	
	Description	Extracts the OCSP request from the HTTP based OCSP request.
	Interface	*ICPEPSOCSPService*
	Input Parameters	• HttpRequest

	Output Returns	• OCSPRequest
	Sequence Diagram	
	processOCSPRequest (*OCSPRequest*)**:** *OCSPResponse*	
	Description	Processes the OCSP Request. This function subsequently calls functions of the OCSP engine (e.g. to validate the request), and the validateCertificate functions of the Specific Validation Engine. Finally, this function constructs an OCSP response (by calling a dedicate function of the OCSP engine) in base of the output returned by the validateCertificate function.
	Interface	*ICPEPSOCSPService*
	Input Parameters	• OCSPRequest
	Output Returns	• OCSPResponse, or • some error
	Sequence Diagram	See ***Figure 16***
Interface Class	*ICPEPSLogService*	
Methods	▪ **saveOCSPRequest** (*OCSPRequest*) : *boolean* ▪ **saveOCSPResponse** (*OCSPResponse*) : *boolean*	
	saveOCSPRequest (*OCSPObject*) : *boolean*	
	Description	
	Interface	*ICPEPSLogService*
	Input Parameters	• OCSPRequest
	Output Returns	• Boolean (True: no errors / False: some error)

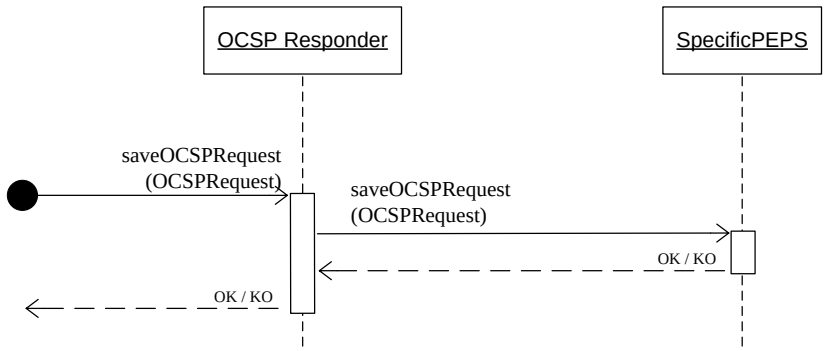
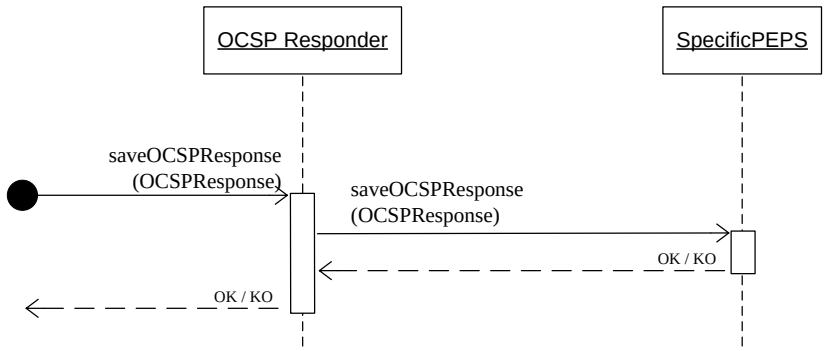
	Sequence Diagram 
	saveOCSPResponse (*OCSPResponset*) : *boolean*
	Description
	Interface — *ICPEPSLogService*
	Input Parameters — OCSPResponse
	Output Returns — Boolean (True: no errors / False: some error)
	Sequence Diagram 

Table 25: Validation- CPEPSValidation component interface

2.5.7.2.3.5.2.3 Other Methods

Methods	▪ **Load colleague_peps_list** *(colleague_peps_list): boolean* ▪ **Extract UserCertificate_Issuer_nonce_colleague** *(OCSPRequest): validationStructure* ▪ **Initialize Network module (***OCSPResponderIPaddress, portnumber): boolean*
	Load colleague_peps_list (colleague_peps_list): boolean
	Description — Loads the colleague peps list
	Interface — -

	Input Parameters	• colleague_peps_list
	Output Returns	• Boolean (True: no errors / False: some error)
	Sequence Diagram	See ***Figure 16***
	Load C-PEPS certificate and key *(C-PEPS certificate and key file): boolean*	
	Description	Loads the X.509v3 certificate of C-PEPS (with extension OCSP signing) and the corresponding key required for signing OCSP responses.
	Interface	-
	Input Parameters	C-PEPS certificate (with extension OCSP signing) and the corresponding key.
	Output Returns	Boolean (True: no errors / False: some error)
	Sequence Diagram	See ***Figure 16***
	Extract UserCertificate_Issuer_nonce_colleague *(OCSPRequest): validationStructure*	
	Description	Extracts from the OCSP request the user certificate, the Issuer data and the nonce value used in the communication with the colleague PEPS.
	Interface	-
	Input Parameters	OCSP Request
	Output Returns	validationStructure composed of the user certificate data, Issuer data and nonce_colleague value
	Sequence Diagram	See ***Figure 16***
	Initialize Network module *(OCSPResponderIPaddress, portnumber): boolean*	
	Description	Initializes the network parameters (e.g. IP address, port number) used by the CPEPSValidation module to communicate with the colleague PEPS.
	Interface	-
	Input Parameters	IP address and port number of the OCSP responder of the CPEPS.
	Output Returns	Boolean (True: no errors / False: some error)
	Sequence Diagram	See ***Figure 16***

Table 26: Validation CPEPSManager component other methods.

2.5.7.2.4 SpecificPEPS: Specific Validation Engine component

2.5.7.2.4.1 Description

2.5.7.2.4.2 Interfaces

Interface Class	*ISpecificValidationService*	
Methods	▪ **validateCertificate** (*UserCertificate data, Issuer data*) : *OCSPResponse*	
	validateCertificate (*UserCertificate , Issuer data*) : *OCSPResponse*	
	Description	
	Interface	*ICPEPSValidationService*
	Input Parameters	• UserCertificate data • Issuer data
	Output Returns	• OCSPResponse signed by the Validation Engine of the Specific PEPS component
	Sequence Diagram See Fig. ***Figure 16***	

Table 27: Specific PEPS Validation Engine component interfaces

2.5.7.2.4.3 Other Methods

Methods	▪ **Load Issuer CA list** *(Issuer CA list): boolean* ▪ **Check Issuer CA** *(OCSPRequestMessage, Issuer CA_list): boolean* ▪ **Check OCSP support** *(OCSPRequestMessage): boolean* ▪ **Get CRL** *(Issuer CA list): CRLObject* ▪ **Validate CRL Object (***CRLObject, Issuer CA list): boolean* ▪ **Issue OCSP request** *(Certificate): OCSP Request* ▪ **Validate OCSP Response** *(OCSP Response): boolean* ▪ **Issue OCSP response** *(OCSPResponse, Specific key): OCSP Response* ▪ **Issue OCSP response (***CRLObject, Specific key): OCSP response*	
	Load Issuer CA list (*Issuer CA list): boolean*	
	Description	Loads the SP list
	Interface	-
	Input Parameters	Issuer CA list
	Output Returns	Boolean (True: no errors / False: some error)
	Check Issuer CA (*UserCertificate, Issuer Data, Issuer CA list): boolean*	
	Description	Checks whether the Issuer CA is known
	Interface	-
	Input Parameters	• UserCertificate, Issuer Data • Issuer CA list

	Output Returns	• Boolean (True: no errors / False: some error)
	Check OCSP support *(UserCertificate, Issuer Data, Issuer CA list): boolean*	
	Description	Checks whether the IDP supports OCSP
	Interface	
	Input Parameters	• UserCertificate, Issuer Data • Issuer CA list
	Output Returns	Boolean (True: no errors / False: some error)
	Get CRL (*(Issuer CA list*) : CRLObject	
	Description	Downloads the CRL from the IDP
	Interface	-
	Input Parameters	Issuer CA list
	Output Returns	• CRL object, or • -some error
	Validate CRL Object (*CRLObject, Issuer CA list): boolean*	
	Description	Validates the CRL received from the IDP using the corresponding IDP certificate stored in the Issuer CA list.
	Interface	*ISpecificValidationService*
	Input Parameters	• CRL Object • Issuer CA list
	Output Returns	Boolean (True: no errors / False: some error)
	Issue OCSP request *(Certificate): OCSP Request*	
	Description	Issues an OCSP request signed with the Specific PEPS' certificate for OCSP request signing.
	Interface	*IOCSPRequestService*
	Input Parameters	• OCSP Request • SP list
	Output Returns	Boolean (True: no errors / False: some error)
	Sequence Diagram	See ***Figure 16***
	Validate OCSP Response *(OCSP Response): boolean*	
	Description	Validates the OCSP response (checks the format, the signature validation, checks whether the responder certificate is trusted , a.s.o.) received from the IDP
	Interface	*ISpecificValidationService*
	Input Parameters	• OCSP Response • IDP certificate for OCSP signing (should be in Issuer CA list)

	Output Returns	Boolean (True: no errors / False: some error)
	Sequence Diagram	See ***Figure 16***
	Issue OCSP response *(OCSPResponse, Specific key): OCSP Response*	
	Description	Issues an OCSP response in base of the OCSP response received from IDP. The newly created OCSP response is signed with the corresponding key of the Specific PEPS for OCSP response signing.
	Interface	*IOCSPResponseService*
	Input Parameters	• OCSP Response • Specific key (for OCSP signing)
	Output Returns	OCSP response, or some error
	Sequence Diagram	See ***Figure 16***
	Issue OCSP response (*CRLObject, Specific key): OCSP response*	
	Description	Issues an OCSP response in base of the CRL object received from the IDP. The newly created OCSP response is signed with the corresponding key of the Specific PEPS certificate for OCSP response signing.
	Interface	*IOCSPResponseService*
	Input Parameters	• CRL Object • Specific key (of Specific PEPS) used for OCSP response signing
	Output Returns	OCSP response, or some error
	Sequence Diagram	See ***Figure 16***

Table 28: Specific PEPS component interfaces

2.5.7.2.5 OCSPEngine component

2.5.7.2.5.1 Description

See chapter 2.6.2.

2.5.7.2.5.2 Methods

<table>
<tr><td>Methods</td><td colspan="2">

OCSPEngine()
Issue OCSP request (User Cert, Issuer data, PEPS key) : OCSP request
Issue OCSP response (status “unknown”, PEPS certificate and key): OCSP response
Issue OCSP response (OCSP response, nonce_SP, PEPS certificate and key) : OCSP response
Issue OCSP response (OCSP Response, nonce_colleague, PEPS certificate and key) : OCSP response
Validate OCSP Response (OCSP Response, Certificate): boolean
Validate OCSP Request (OCSP Request, SP list) : boolean
Validate OCSP Request (OCSP Request, colleague_peps_list): boolean
Validate OCSP Response (OCSP Response, colleague_peps_list): boolean

</td></tr>
<tr><td></td><td colspan="2">Validate OCSP Request (OCSP Request, SP list) : boolean</td></tr>
<tr><td></td><td>Description</td><td>Validates the OCSP request (checks the format, the signature validation, checks whether the requester certificate is trusted , a.s.o.) received from SP</td></tr>
<tr><td></td><td>Interface</td><td>IOCSPRequestService</td></tr>
<tr><td></td><td>Input Parameters</td><td>• OCSP Request
• SP list</td></tr>
<tr><td></td><td>Output Returns</td><td>Boolean (True: no errors / False: some error)</td></tr>
<tr><td></td><td>Sequence Diagram</td><td>See Figure 16</td></tr>
<tr><td></td><td colspan="2">Issue OCSP request (UserCert, Issuer data, PEPS key): OCSP Request</td></tr>
<tr><td></td><td>Description</td><td>Issues an OCSP request signed with the SPEPS’ key for OCSP request signing.</td></tr>
<tr><td></td><td>Interface</td><td>IOCSPRequestService</td></tr>
<tr><td></td><td>Input Parameters</td><td>• OCSP Request
• SP list</td></tr>
<tr><td></td><td>Output Returns</td><td>OCSP Request</td></tr>
<tr><td></td><td>Sequence Diagram</td><td>See Figure 16</td></tr>
<tr><td></td><td colspan="2">Validate OCSP Request (OCSP Request, colleague_peps_list) : boolean</td></tr>
<tr><td></td><td>Description</td><td>Validates the OCSP request (checks the format, the signature validation, checks whether the requester certificate is trusted , a.s.o.) received from colleague PEPS</td></tr>
<tr><td></td><td>Interface</td><td>IOCSPRequestService</td></tr>
<tr><td></td><td>Input Parameters</td><td>• OCSP Request
• colleague peps list</td></tr>
<tr><td></td><td>Output Returns</td><td>Boolean (True: no errors / False: some error)</td></tr>
</table>

	Sequence Diagram	See ***Figure 16***
	Validate OCSP Response *(OCSP Response, Certificate): boolean*	
	Description	Validates the OCSP response (checks the format, the signature validation, checks whether the responder certificate is trusted , a.s.o.) received from Specific PEPS (the Specific Validation Engine component)
	Interface	*ISpecificValidationService*
	Input Parameters	• OCSP Response • Specific PEPS certificate (for OCSP response signing)
	Output Returns	Boolean (True: no errors / False: some error)
	Sequence Diagram	See ***Figure 16***
	Validate OCSP Response *(OCSP Response, colleague_peps_list): boolean*	
	Description	Validates the OCSP response (checks the format, the signature validation, checks whether the responder certificate is trusted , a.s.o.) received from colleague PEPS
	Interface	*ISpecificValidationService*
	Input Parameters	• OCSP Response • colleague_peps_list
	Output Returns	Boolean (True: no errors / False: some error)
	Sequence Diagram	See ***Figure 16***
	Issue OCSP response *(status "unknown", PEPS certificate and key): OCSP response*	
	Description	Issues an OCSP response with the status ``unknown''. The OCSP response is signed with the SPEPS' key fro OCSP response signing.
	Interface	*IOCSPResponseService*
	Input Parameters	• status 'unknown' • PEPS certificate and key for OCSP response signing (could be SPEPS or CPEPS)
	Output Returns	• OCSP response. • or some error
	Sequence Diagram	See ***Figure 16***
	Issue OCSP response *(OCSP response, nonce_SP, PEPS certificate and key) : OCSP response*	
	Description	Issues an OCSP response in base of the OCSP response received from the colleague PEPS. The newly created OCSP response must contain the nonce extracted from the OCSP request received from the SP and is signed with the corresponding PEPS key for OCSP response signing.
	Interface	*IOCSPResponseService*

	Input Parameters	• OCSP Response • nonce_SP • S-PEPS certificate and key
	Output Returns	• OCSP response, or • some error
	Sequence Diagram	See ***Figure 16***
	Issue OCSP response *(OCSP Response, nonce_colleague, PEPS certificate and key) : OCSP response*	
	Description	Issues an OCSP response in base of the OCSP response received from the Specific PEPS. The newly created OCSP response must contain the nonce extracted from the OCSP request received from the colleague PEPS and is signed with the corresponding PEPS key for OCSP response signing.
	Interface	*IOCSPResponseService*
	Input Parameters	• OCSP Response • nonce_SP • C-PEPS certificate and key
	Output Returns	• OCSP response, or • some error
	Sequence Diagram	See ***Figure 16***

Table 29:OCSPEngine component interfaces

2.5.7.3 Object Model

2.5.7.3.1 Class Diagram

2.5.7.3.2 Packages

2.5.7.3.2.1 EU.STORK.PEPS.VALIDATION

2.5.7.3.2.1.1 ValidationPEPS

Description	Chapter 2.5.6
Type	Component
Interfaces	IValidationService
Methods	

Table 30: Validation PEPS Package

2.6 Specific Development

2.6.1 Introduction

The specific development was completely detached from the PEPS to allow each Member State to have their own independent and decoupled implementation. The way it was proposed at first, Specific code was tightly coupled to the PEPS code. This turned out to be infeasible while communicating with an external IdP/AP. On this new proposal each Member State just has to implement a few Actions (and/or the SpecificPEPS class) and the code will run on every newest version of PEPS.

2.6.2 Diagram

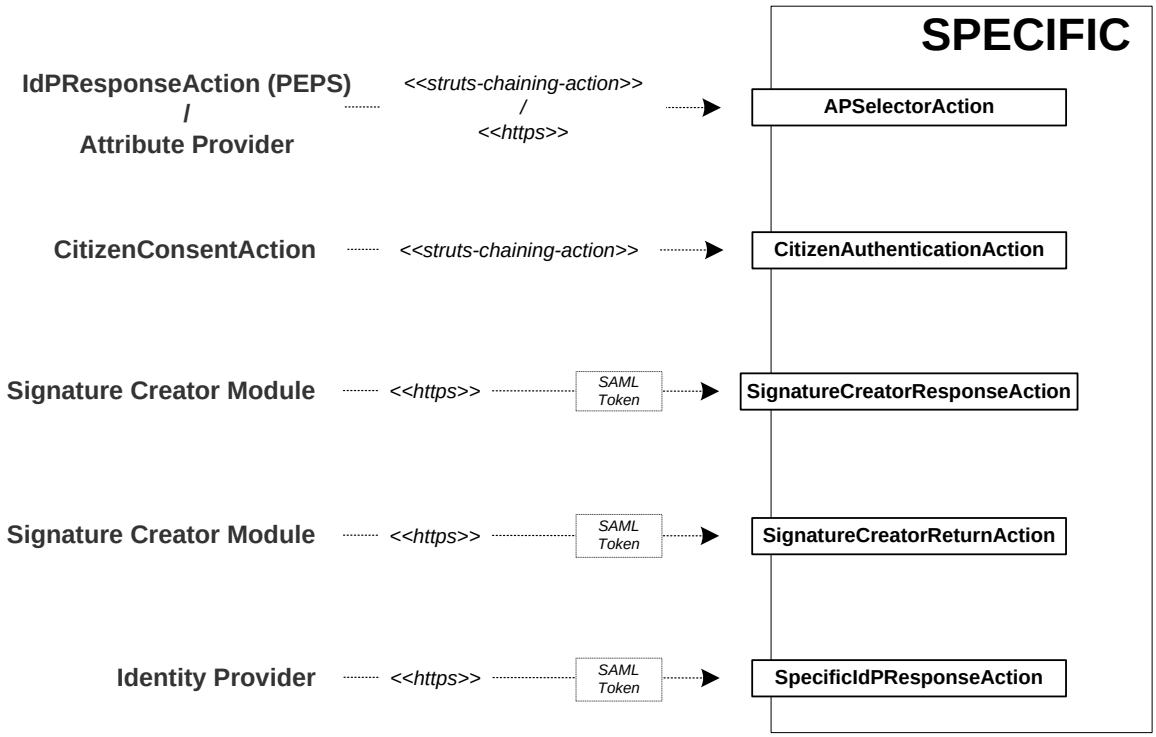


Figure 18: Specific Development global overview

2.6.3 Actions

Action Class	*APSelectorAction*	
Methods	**execute () :** *String*	
	Description	Receives and handles the request from the IdPResponseAction, and request attributes values to the Attribute Provider (external/internal).It handles too the response from the Attribute Provider(s) and request the missing attributes to further Attribute Provider(s) (if available) . Also, it can request the signedDoc attribute value to the Signature Creator Module (if available).
	Action	*APSelectorAction*
	Input Parameters	
	Output	String (“external-ap”: to redirect the Citizen to an external AP;

	Returns	"internal-ap": strtuts-redirect to the PEPS's APResponseAction; "external-sig-module": to redirect the Citizen to the Signature Creator Module)
Action Class	*CitizenAuthenticationAction*	
Methods	**execute () :** *void*	
	Description	Handles the request from CitizenConsentAction and redirects the Citizen to and external IdP or proceed to an internal Authentication.
	Servlet	*CitizenAuthenticationAction*
	Input Parameters	
	Output Returns	String ("internal-authentication": struts-redirect to SpecificIdPResponseAction; "external-authentication": to redirect user to an external IdP)
Action Class	SignatureCreatorResponseAction	
Methods	**execute () :** *String*	
	Description	Receives and validates the response from the Signature Creator Module and replies the address to receive the Citizen browser's "control".
	Servlet	SignatureCreatorResponseAction
	Input Parameters	
	Output Returns	String
Action Class	SignatureCreatorReturnAction	
Methods	**execute () :** *String*	
	Description	Receives the control from the Signature Creator Module and struts-redirect to the PEPS's APResponseAction.
	Servlet	SignatureCreatorReturnAction
	Input Parameters	
	Output Returns	String
Action Class	*SpecificIdpResponse*	
Methods	**execute () :** *String*	
	Description	Handles the response from the IdP (external/internal) and translates to the PEPS "expecting" response.
	Servlet	*SpecificIdpResponse*
	Input Parameters	
	Output Returns	String

Table 31: Specific Actions

3 Standard format of log files

3.1 Introduction

In STORK a log is produced which is meant to guarantee traceability, without storing any personal data. The objective of producing this log is fulfilled in the PEPS through calls from the STORK system to the Log4J libraries and utilities, which are configured to produce a file in the format which is described in the following paragraphs.

Please note that any Member State is free to substitute these flat files by databases, and to use these data for statistical purposes. As a matter of fact, Log4J connectors for Oracle and MySQL are available.

3.2 Entries in the log file

Standard PEPSes store log information in a flat file, with one log entry on each line.

Entries can be requests and replies; thus in an S-PEPS one transaction will be reflected 4 times:

- when receiving a request from a SP
- when sending a request to a C-PEPS
- when receiving the reply from this C-PEPS
- when sending the reply to the SP.

In a C-PEPS a transaction will be refelected twice.

All data-items in a line are separated from eachother by a hash-sign (#).

3.3 Requests in the log file

Requests in the logfile have the following data items:

- datetime
- opType:

  One of:

  - request received from SP
  - request sent to C-PEPS
  - request received from S-PEPS
- origin

  The Service provider or PEPS IP and domain name which sent the request.
- Destination

  The PEPS IP and domain name where the request is (being) sent.
- sp_application
- sp_provider_name
- country

In case of the S-PEPS the citizen country. In case of the C-PEPS the SP_country

- qaalevel
- msg_hash
- msg_id

  A unique identifier of the message.

3.4 Responses in the log file

Responses in the logfile have the following data items:

- datetime
- opType:

  One of:

  - response sent to S-PEPS
  - response received from C-PEPS
  - response sent to SP
- orig_msg_id

  The value of msg_id in the originating request
- message

  The status message of the response
- msg_hash
- msg_id

  A unique identifier of this message.

Please note that responses can be related to the corresponding request through the orig_msg_id in the response, which must have the same value in the msg_id of the request.

3.5 Example fragment of a log file

The following is an example of such a log file.

```
2011-08-23 14:06:54.444#S-PEPS receives request from SP#http://sp:8080/SP/ReturnPage#http://peps:8080/PEPS/ColleagueRequest#null#DEMO-SP#LOCAL#3#sbF6fEJrm0gk4y2Jm6t3Wzcz5C+piL+l/S4VPh55rJTPD1AsX2V0H38OLxJMLiLVYGCeq2lrsX8md7V/ZUbJ8Q==#_6e843195728a5ff37b8025b1bb86dd7d#

2011-08-23 14:06:54.945#S-PEPS generates request to C-PEPS#http://peps:8080/PEPS/ColleagueResponse;jsessionid=331A7D8ABCF03F1956993AECCA7542D3#http://peps:8080/PEPS/ColleagueRequest#null#DEMO-SP#LOCAL#3#8TIoJHEMDeKT6KLzXUgsuV2qyHzdZkUPv3pwMlj0OaRP6gTjm8htfbnCNKC85nvQx3PAuPoDFPbwHy22kYT+rA==#_6e843195728a5ff37b8025b1bb86dd7d#_95b81e300a5acaa1cf5f916e58c32e11#

2011-08-23 14:07:05.093#C-PEPS receives request from S-PEPS#http://peps:8080/PEPS/ColleagueResponse;jsessionid=331A7D8ABCF03F1956993AECCA7542D3#http://peps:8080/PEPS/ColleagueRequest#null#DEMO-
```

SP#PT#3#8TIoJHEMDeKT6KLzXUgsuV2qyHzdZkUPv3pwMlj0OaRP6gTjm8htfbnCNKC85nvQx3PAuPoDFPbwHy22kYT+rA==#_95b81e300a5acaa1cf5f916e58c32e11#

2011-08-23 14:07:59.083#C-PEPS generates response to S-PEPS#_95b81e300a5acaa1cf5f916e58c32e11#Success/Get Citizen Consent#3isNDApTA0fo6RkQQGWBD/wNY0NhSZ5qgxWkUbMO4eJlJarr0NOo+TjcIX/jXi1IIwVIRD3J6iYLs5sgbuiFUg==#_55a9d1b6917fa0a37539e5b2b9acec8b#

2011-08-23 14:08:03.085#S-PEPS receives response from C-PEPS#_95b81e300a5acaa1cf5f916e58c32e11#_6e843195728a5ff37b8025b1bb86dd7d#Success/Get Citizen Consent#z4PhNX7vuL3xVChQ1m2AB9Yg5AULVxXcg/SpIdNs6c5H0NE8XYXysP+DGNKHfuwvY7kxvUdBeoGlODJ6+SfaPg==#_55a9d1b6917fa0a37539e5b2b9acec8b#

2011-08-23 14:08:03.176#S-PEPS generates response to SP#_6e843195728a5ff37b8025b1bb86dd7d#Success/Get Citizen Consent#+SdwRDLRnkuK2+E69P4srdJHn58vN97ra2D5hgE4g+EQIGG5W554yhkTw+GMjydT7OTgpd90HPA7dvrf0USb0A==#_af03e31e6c2dd6bbb1b810e8565b2eb6#

4 Implementation Issues

4.1 Circle of Trust: Mesh Architecture

This section details the architecture proposed to establish the circle of trust among PEPS respecting their certificate-based digital identity.

4.1.1 Approach

Taking into account proposed STORK architecture and design, next Mesh Architecture will be deployed in the STORK core where SAML Tokens and OCSP messages will be exchanged among the existent PEPS.

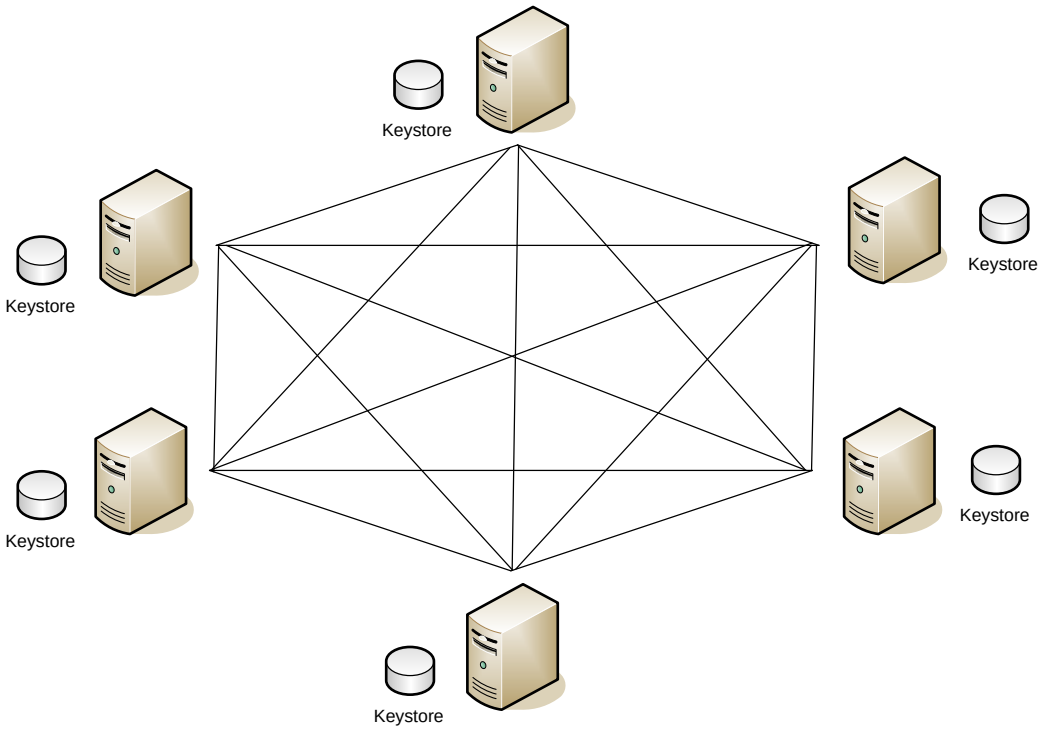


Figure 19 – Circle of Trust in the Mesh Architecture

As can be seen in the figure above, each PEPS manages a local keystore (see subsection 4.1.2). On the other hand, the types of messages that can be exchanged between two PEPS are the next:

- A SAML request sent by a SAML requester to a SAML responder.
- A SAML response sent by a SAML responder to a SAML requester.
- An OCSP request sent by an OCSP client to an OCSP responder.
- An OCSP response sent by an OCSP responder to an OCSP client.

Each PEPS acts as both a requester and a responder. Depending on the service delivered, the type of message exchanged corresponds to a SAML token (Authentication Service) or an OCSP message (Validation Service).

Every message must be protected from potential attacks, like unauthorized modifications, identity masquerading attacks and eavesdropping. For that purpose, each message must be signed and encrypted by the sender by using asymmetric cryptography.

In order to avoid the deployment of an adhoc PKI to issue and manage the certificates to be used by the PEPS, the Mesh Architecture supposes an explicit circle of trust created among them. This is achieved by creating and distributing keystores that contain every PEPS certificate. Furthermore, each PEPS certificate must be a self-signed certificate. As a result, the generation of a common trusted keystore to be used by all PEPS is straightforward.

As stated in section 4.2.2.2 Authorized Responders of RFC 2560 [2], an OCSP Responder can be the CA itself or a third party authorized to do so.

> Systems or applications that rely on OCSP responses MUST be capable of detecting and enforcing use of the id-ad-ocspSigning value as described above. They MAY provide a means of locally configuring one or more OCSP signing authorities, and specifying the set of CAs for which each signing authority is trusted. They MUST reject the response if the certificate required to validate the signature on the response fails to meet at least one of the following criteria:
>
> 1. Matches a local configuration of OCSP signing authority for the certificate in question; or
>
> 2. Is the certificate of the CA that issued the certificate in question; or
>
> 3. Includes a value of id-ad-ocspSigning in an ExtendedKeyUsage extension and is issued by the CA that issued the certificate in question.

Because each PEPS only trusts in its national CAs, the certificate used by the PEPS OCSP Responder cannot be issued by a national CA. As a result, options 2 and 3 cannot be applied. Option 1 above is aligned with the Mesh Architecture and self-signed certificate philosophy herein proposed.

Next subsections provide the detail respecting how to enforce the circle of trust herein proposed.

4.1.2 Keystores

A keystore stores cryptographic material, such as cryptographic keys and digital certificates, normally in a protected way (e.g. by means of password). A keystore permits an entity to establish a trust relationship with third parties by including their certificates (or the corresponding CA certificates) in the keystore. Thereby, the entity can explicitly (inclusion of the certificate) or implicitly (inclusion of the –intermediate/root - CA certificate) trust in that third party.

As shown in Figure 1 above, each PEPS manages a local keystore. We propose a design based on three different local keystores managed by each PEPS, two of them focused on enforcing the circle of trust (CoT) previously mentioned while the third one oriented to store the cryptographic material such as the PEPS private keys.

The CoT-enforcing keystores proposed are the next:

- **STORKTrustedKeyStore.jks**

  This keystore contains the self-signed certificates of every PEPS and used to sign SAML Tokens and OCSP request.

  A certificate used to sign a SAML request, a SAML response or an OCSP request can be the same, as no special extensions or distinguishable information is required (see section 4.1.4). As a result, the PEPS can use the same key pair to protect those messages.

- **STORKOCSPRespondersTrustedKeyStore.jks**

This keystore contains the self-signed certificates of every PEPS OCSP Responders and used to sign OCSP responses.

Every certificate herein contained must have the ocsp-signing extended key usage extension marked as critical (see section 4.1.4.3).

The two keystores above just contain digital certificates, but not private keys. The next keystore contains the signing keys necessary for the PEPS to sign the information exchanged. This keystore varies according to each PEPS:

- **STORKOwnKeyStore.jks**

  This keystore contains two self-signed certificates (also included in the keystores above) and the corresponding private keys owned by the particular PEPS.

  One key is used to sign OCSP responses sent to other PEPS. The corresponding certificate must have the ocsp-signing extension marked as critical (see section 4.1.4.3). The other key is used to sign SAML requests, SAML responses and OCSP requests.

As can be seen, the proposed keystore format is the standard Java KeyStore format (JKS).

NOTE: Other alternatives cover the creation of just one certificate per PEPS which includes the ocsp-signing extension. This single certificate would be used for signing any message sent through the STORK network. Notwithstanding next subsections suppose a design based on two different certificates.

4.1.3 Scalability and Management

The adhesion of partners from new Member States and changes in the existent certificates owned by the PEPS obliges to update the information in a multicast approach. Next subsections briefly explain how to cope with these two issues.

4.1.3.1 New Member State adhesion

If a new Member State (MS) wants to adhere to STORK in a future, it should perform the next steps:

- Generate two self-signed certificates with the corresponding private keys, one of them for signing OCSP responses (including ocsp-signing extension), and the other to sign OCSP requests and SAML requests/responses. Constraints on the certificate formats can be found in subsection 4.1.4.
- Create *STORKOwnKeyStore* keystore with both certificates and corresponding private keys.
- Propagate both certificates to the rest of PEPS in order to allow them to update the *STORKTrustedKeyStore* and *STORKOCSPRespondersTrustedKeyStore* keystores. The mechanism implemented to send these two certificates to the rest of the PEPS must assure the reliance of the information transmitted. As the certificate is self-signed, the mechanism must additionally provide peer authentication. For instance, out-of-band mechanisms could be used.
- Finally, obtain *STORKTrustedKeyStore* and *STORKOCSPRespondersTrustedKeyStore* keystores from any PEPS, with the certificates already updated.

It is important to remark that a design based on two different certificates allows a MS to decide if adhering to STORK implementing the authentication service (based on SAML), the validation service (based on OCSP) or both of them without having to change the structure of the corresponding certificate. Furthermore, using two certificate permits to define the key usage purposes in a clearer manner.

4.1.3.2 Update of current certificates

If a PEPS certificate has to be revoked (e.g. a private key compromise) or expires, then the new one must be transmitted using out-of-band mechanisms or an agreed online mechanism to the rest of the PEPS in order to update the corresponding keystores (*STORKTrustedKeyStore* and/or *STORKOCSPRespondersTrustedKeyStore*).

Furthermore, the affected PEPS must update the *STORKOwnKeyStore* with the new key pair and corresponding certificate.

4.1.3.3 Status validity management

How can the revocation status of a PEPS certificate be retrieved if no supporting PKI exists? The approach proposed is similar to a PGP (Pretty Good Privacy) circle of trust, where each entity explicitly manages the trust or distrust in other entities.

In our case, a PEPS certificate is considered valid if it is found in the adequate keystore (*STORKTrustedKeyStore* or *STORKOCSPRespondersTrustedKeyStore*). Otherwise, it is simply rejected.

For that reason, when a new certificate is distributed, either by a new PEPS or due to a revocation/renewal, each PEPSs must update its keystores. During the update process, if an entry is found for the received new PEPS certificate, then the current one must be replaced by the new one.

4.1.4 Certificate Formats

As explained in section 4.1.2, each PEPS will own two different certificates. This subsection specifies the common requirements that those certificates must fulfil as well as the specific requirements that each certificate must comply with.

4.1.4.1 Common requirements

Next requirements are common constraints for both certificates. The certificate template taken into account corresponds to that defined in RFC 5280 [1].

```
Certificate  ::=  SEQUENCE  {
     tbsCertificate       TBSCertificate,
     signatureAlgorithm   AlgorithmIdentifier,
     signatureValue       BIT STRING  }

TBSCertificate  ::=  SEQUENCE  {
     version         [0]  EXPLICIT Version DEFAULT v1,
     serialNumber         CertificateSerialNumber,
     signature            AlgorithmIdentifier,
     issuer               Name,
     validity             Validity,
     subject              Name,
     subjectPublicKeyInfo SubjectPublicKeyInfo,
     issuerUniqueID  [1]  IMPLICIT UniqueIdentifier OPTIONAL,
                          -- If present, version MUST be v2 or v3
     subjectUniqueID [2]  IMPLICIT UniqueIdentifier OPTIONAL,
                          -- If present, version MUST be v2 or v3
     extensions      [3]  EXPLICIT Extensions OPTIONAL
                          -- If present, version MUST be v3
     }
```

4.1.4.1.1 Serial number

The *serialNumber* value is up to the PEPS issuing the certificate. As only one certificate will be active per PEPS at the same time, this value has not actual relevance.

4.1.4.1.2 Issuer

A homogeneous *issuer* distinguished name should be used among every PEPS. A proposal is the next:

CN = servername, OU = PEPS, O = STORK, C = {ES, EN, BE,…}

4.1.4.1.3 Validity

It should be agreed the same *validity* period for every PEPS certificate. A proposal is a period of 10 years from the issuance moment.

4.1.4.1.4 Subject

As a self-signed certificate, the *subject* must be the same as the issuer distinguished name.

CN = servername, OU = PEPS, O = STORK, C = {ES, EN, BE,…}

4.1.4.1.5 Extensions

RFC 5280 defines the *extensions* field as follows:

```
Extensions  ::=  SEQUENCE SIZE (1..MAX) OF Extension

Extension  ::=  SEQUENCE  {
        extnID      OBJECT IDENTIFIER,
        critical    BOOLEAN DEFAULT FALSE,
        extnValue   OCTET STRING
                    -- contains the DER encoding of an ASN.1 value
                    -- corresponding to the extension type
                    -- identified
                    -- by extnID
        }
```

Following extensions are mandatory for PEPS certificates. Each one indicates the OID to be included in *extnID* field, the *critical* value and the ASN.1 type for *extnValue*.

KeyUsage

```
id-ce-keyUsage OBJECT IDENTIFIER ::=  { id-ce 15 }

critical ::= TRUE

KeyUsage ::= BIT STRING {
     digitalSignature        (0),
     nonRepudiation          (1), -- recent editions of X.509 have
                          -- renamed this bit to contentCommitment
     keyEncipherment         (2),
     dataEncipherment        (3),
     keyAgreement            (4),
     keyCertSign             (5),
     cRLSign                 (6),
     encipherOnly            (7),
     decipherOnly            (8) }
```

Every PEPS certificate should be issued with next key usages: *digitalSignature*, *nonRepudiation*, *keyEncipherment* (if SAML Tokens have to be encrypted with a symmetric key), *dataEncipherment* (if SAML Tokens have to be encrypted with the asymmetric public

key of the receiver PEPS), *keyAgreement* (for the SSL/TLS connections with Web browsers of end users) and *keyCertSign* (as it is a self-signed certificate).

Certificate Policy

A certificate policy should be agreed among all MS and generated for its usage. However, we propose to postpone this issue for future phases.

Basic Constraints

```
id-ce-basicConstraints OBJECT IDENTIFIER ::=  { id-ce 19 }

critical ::= FALSE

BasicConstraints ::= SEQUENCE {
      cA                      BOOLEAN DEFAULT FALSE,
      pathLenConstraint       INTEGER (0..MAX) OPTIONAL }
```

As the certificate is self-signed by the PEPS itself, the *cA* field must be set to TRUE. *PathLenConstraint* must be established to zero.

Extended Key Usage

```
id-ce-extKeyUsage OBJECT IDENTIFIER ::= { id-ce 37 }

critical ::= TRUE

ExtKeyUsageSyntax ::= SEQUENCE SIZE (1..MAX) OF KeyPurposeId

KeyPurposeId ::= OBJECT IDENTIFIER
```

The KeyPurposeId to be included in the extended key usage for the PEPS certificates is the next:

```
id-kp-serverAuth OBJECT IDENTIFIER ::= { id-kp 1 }
```

being

```
id-kp OBJECT IDENTIFIER ::= { id-pkix 3 }
```

This extended key usage is needed for establishing SSL/TLS connections with Web browsers of end users.

4.1.4.2 Certificate for SAML Tokens and OCSP Requests

This self-signed certificate does not have any type of requirement nor constraint except those identified in subsection 4.1.4.1.

4.1.4.3 Certificate for OCSP Responders

According to RFC 2560 [2], the certificate issued to an OCSP Responder must contain the *id-kp-OCSPSigning* OID in an extended key usage certificate extension (extendedKeyUsage extension).

Therefore, the key usage OID to be included in the extended key usage is the next:

```
id-kp-OCSPSigning  OBJECT IDENTIFIER ::= { id-kp 9 }
```

This extension must be marked as critical taking into account STORK design.

On the other hand, and as stated in RFC 2560 – 4.2.2.2.1 Revocation Checking of an Authorized Responder, there are three ways for the client to verify the revocation status of the certificate used by the OCSP responder. The most convenient option for the STORK architecture is to include the non-critical certificate extension *id-pkix-ocsp-nocheck* in the OCSP Responder certificate.

```
id-pkix-ocsp-nocheck OBJECT IDENTIFIER ::= { id-pkix-ocsp 5 }

critical ::= FALSE

extnValue must be NULL
```

4.2 Configuration files

4.3 Dependencies

The philosophy followed in the project has been to use open frameworks and standards. In this sense, next libraries are proposed for the implementation of the Authentication and Validation services in the PEPS:

- **OpenSAML** [3], which is a set of open source (Apache 2 licensed) C++ & Java libraries meant to support developers working with the Security Assertion Markup Language (SAML). OpenSAML 1 supports SAML 1.0, 1.1 and 2.0. It also implements XML signature generation and verification processes.
- **Bouncy Castle JCE** [4] is a cryptographic provider that offers a complete library that implements ASN.1 based PKI operations and data structures. It covers the whole specification described in OCSP standard, among others.

Dependencies:

- JAXP 1.3(Xerces and Xalan)
- java-xmltooling
- java-openws
- java-opensaml2
- Maven (maven2eclipse plugin)
- Junit, xmlUnit
- Log4J or logback (logback-core and logback-classic)

5 References

[1] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, W. Polk. RFC 5280 – Internet X.509 Public Key Infrastructure - Certificate and Certificate Revocation List (CRL) Profile. International Engineering Task Force (IETF). May 2008.

[2] M. Myers, R. Ankney, A. Malpani, S. Galperin, C. Adams. RFC 2560 – Internet X.509 Public Key Infrastructure – Online Certificate Status Protocol – OCSP. International Engineering Task Force (IETF). June 1999.

[3] OpenSAML: https://spaces.internet2.edu/display/OpenSAML/Home/

[4] Bouncy Castle: http://www.bouncycastle.org/