Implementation of Application Integration Service

D44.4

Document Identification	
Date	11/08/2015
Status	Final
Version	1.0

Related SP / WP	SP4/ WP44	**Document Reference**	44.4
Related Deliverable(s)	D44.1, D44.2, D44.3	**Dissemination Level**	PU
Lead Participant	ATOS	**Lead Author**	Juan Carlos Pérez Baún
Contributors	Nuria Ituarte Aranda (ATOS) Miguel Colomer Pastor (ATOS) Juan Carlos Pérez Baún (ATOS) Charles Bastos Rodriguez (ATOS) Tino Liebeskind (AGETO) Jerzy Szczebak (CA)	**Reviewers**	TUD, FHG

This document is issued within the frame and for the purpose of the *FutureID* project. This project has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under Grant Agreement no. 318424.

1 Executive Summary

The Application Integration Service is an access control system that handles authentication for the service provider's applications.

The Application Integration Service covers two major roles in the FutureID architecture:

- Intercepting unknown users and requesting the FutureID infrastructure to authenticate them;
- Receiving and validating credentials in order to set up authenticated sessions for users.

The implementation of JBoss specific Application Integration Service is oriented towards the epSOS pilot integration, which runs on JBoss server. In this way this Application Integration Service is covering the market for Java applications servers.

We describe the implementation of the three different components of the Application Integration Service:

- The Access Filter and the FutureID client detection module;
- The Simple Credential Transformer;
- The Session Library.

This deliverable also provides the environment used for the implementation and some guides for deployment and configuration of the different Application Integration Service building blocks.

2 Document Information

2.1 Contributors

Name	Partner
Juan Carlos Pérez Baún	ATOS
Nuria Ituarte Aranda	ATOS
Miguel Colomer Pastor	ATOS
Charles Bastos Rodriguez	ATOS
Tino Liebeskind	AG
Jerzy Szczebak	CA

2.2 History

Version	Date	Author	Changes
0.1	29/09/2014	Juan Carlos Pérez Baún	Create structure
0.2	03/12/2014	Juan Carlos Pérez Baún Nuria Ituarte Aranda	Session Library and FAR specifications included
0.3	12/12/2014	Nuria Ituarte Aranda	SL updates. Section 9 and Annex B.
0.31	13/03/2015	Nuria Ituarte Aranda Juan Carlos Pérez Baún	Updating SCT structure
0.4	08/06/2015	Tino Liebeskind	Adding SCT section
0.41	09/06/2015		Internal Review
0.42	12/06/2015	Tino Liebeskind	Adjust images, add Appendix for SCT
0.5	24/06/2015	Jerzy Szczebak	Adding FC detection description
0.6	04/07/2015	Jerzy Szczebak	Revision Section 7
0.7	16/07/2015	Juan Carlos Pérez Baún	Internal revision
0.8	23/07/2015	Charles Bastos Rodriguez	Internal revision
0.9	06/08/2015	Jon Rios	Updates from reviewers
0.91	10/08/2015	Heiko Roßnagel	Updates from reviewers
1.0	11/08/2015	Nuria Ituarte	Final revision to submission

2.3 Table of Figures

2.4 Table of Tables

2.5 Table of Acronyms

AF	Access Filter
AIS	Application Integration Service
AJAX	Asynchronous JavaScript And XML
BS	Broker Service
FC	FutureID Client
FAR	FutureID Authentication Request
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
REST	REpresentational State Transfer
SCT	Simple Credential Transformer
SL	Session Library
STORK	Secure idenTity acrOss boRders linked

2.6 Referenced Documents

[1] - FutureID_D44.03_WP44_v1.0_Technical_Specification_for_AIS, https://dms-prext.fraunhofer.de/livelink/livelink.exe?func=ll&objaction=overviewversion&objid=4353210&vernum=2

[2] - FutureID_D44.02_WP44_v1.2_Application Integration Service Requirements_Final,

https://dms-prext.fraunhofer.de/livelink/livelink.exe?func=ll&objaction=overviewversion&objid=3303325&vernum=2

[3] - FutureID_D21.04_WP21_v1.1_Reference_Architecture, https://dms-prext.fraunhofer.de/livelink/livelink.exe?func=ll&objaction=overview&objid=3841750

3 Table of Contents

4 Project Description

The *FutureID* project builds a comprehensive, flexible, privacy-aware and ubiquitously usable identity management infrastructure for Europe, which integrates existing eID technology and trust infrastructures, emerging federated identity management services and modern credential technologies to provide a user-centric system for the trustworthy and accountable management of identity claims.

The *FutureID* infrastructure will provide great benefits to all stakeholders involved in the eID value chain. Users will benefit from the availability of a ubiquitously usable open source eID client that is capable of running on arbitrary desktop PCs, tablets and modern smart phones. *FutureID* will allow application and service providers to easily integrate their existing services with the *FutureID* infrastructure, providing them with the benefits from the strong security offered by eIDs without requiring them to make substantial investments.

This will enable service providers to offer this technology to users as an alternative to username/password based systems, providing them with a choice for a more trustworthy, usable and innovative technology. For existing and emerging trust service providers and card issuers *FutureID* will provide an integrative framework, which eases using their authentication and signature related products across Europe and beyond.

To demonstrate the applicability of the developed technologies and the feasibility of the overall approach *FutureID* will develop two pilot applications and is open for additional application services who want to use the innovative *FutureID* technology

Future ID is a three-year duration project funded by the European Commission Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 318424.

5 Introduction

This deliverable D44.4 describes the implementation of the Application Integration Service (AIS), namely the development of the JBoss specific AIS. This task will implement the AIS based on the technical specification from deliverable D44.3 **[1]** following the requirements provided by deliverable D44.2 **[2]**. This task will enable AIS to generate STORK-compatible authentication requests to the identity broker, also to receive and process STORK-compatible authentication responses to the application services component.

The AIS covers two major roles in the FutureID architecture:

- Intercepting unknown users and requesting the FutureID infrastructure to authenticate them;
- Receiving and validating credentials in order to set up authenticated sessions for users.

In more detail, in the former role, the User Agent (UA) requests a resource provided by the application A. The request is intercepted by the Access Filter (AF). If the user is already known, the request is passed to A. Otherwise, AF issues a FutureID Authentication Request (FAR) through the User Agent (UA) to the Solver & Executor (S&E). The AF uses the Session Library (SL) to determine whether a user is already known.

In the latter role, the S&E presents a user or session credential to the SCT suitable for this credential. The SCT verifies the credential and, on success, sets a user session by calling according services of the SL. In the verification process, the SCT can call TS services to determine whether the presented credential originates from an issuer who is trustworthy according to the SP's trust policy. Once an authenticated session is established, the SCT redirects the user to the originally requested resource of A.

6 FutureID JBoss specific AIS

6.1 JBoss specific AIS Architecture overview

The JBoss AIS consist of two main components the Access Filter (AF) and the Simple Credential Transformer (SCT). The former intercepts the request from the user and trigger the authentication process. The latter manage the SAML authentication response received from the Broker Service (BS) in order to grant access the user to the requested resource. Both components are connected with the Session Library (SL) component in charge of validating the session if already exists or creating a new one for the user.

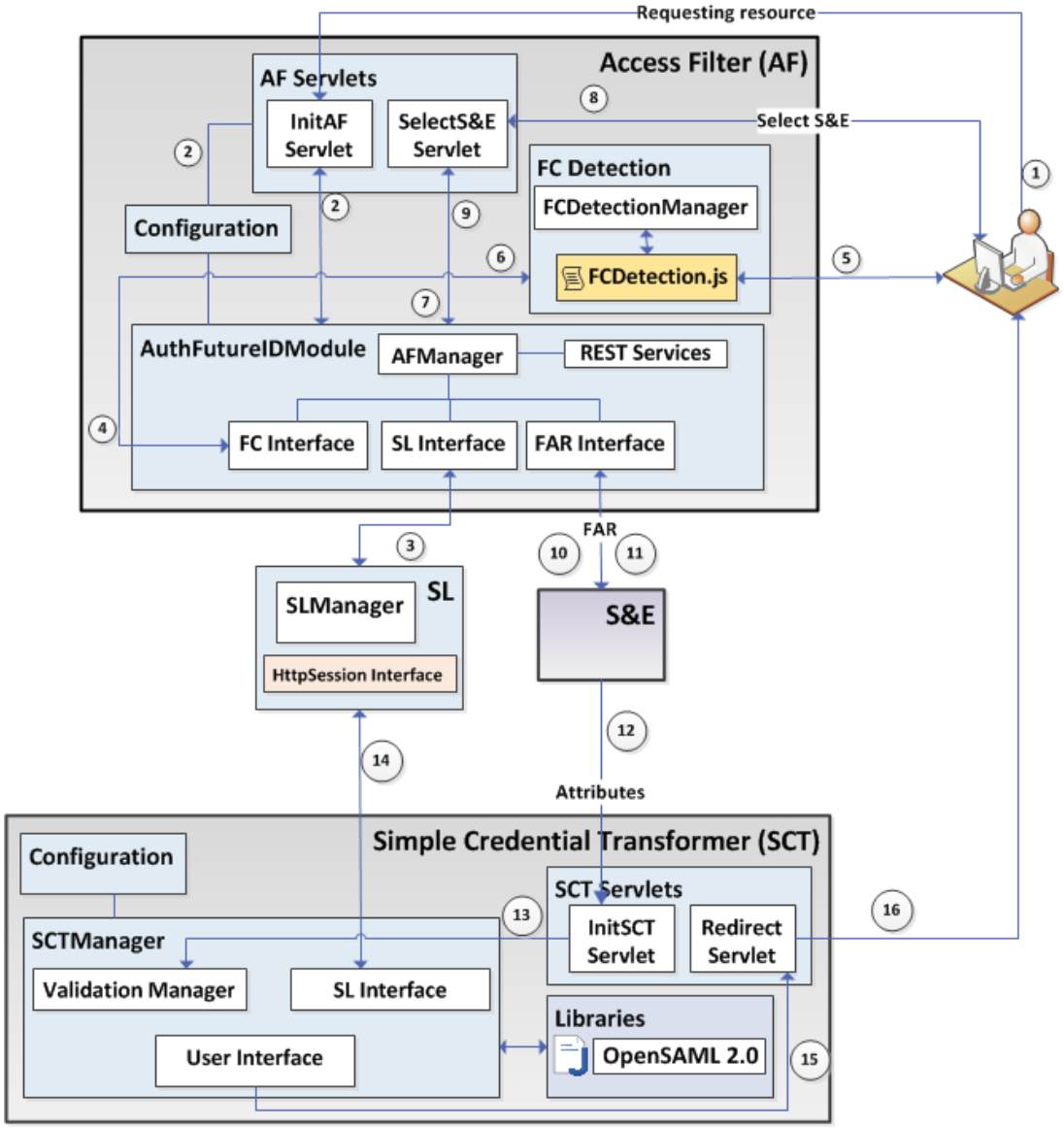

Figure 1: JBoss specific AIS architecture

Figure 1 shows the architecture of the JBoss AIS and the main flows during the authentication process. A description of each step is provided next:

1- The user requests a resource

2- The AF filters the request

3- The AF checks whether the user is known (SL)

4- The AF checks whether the FC is present

5- The FC detection component runs js to detect the FC

6- The presence of the FC is provided to AF

7- The AIS asks the user to choose S&E

8- The user selects S&E

9, 10- The FAR interface connects to S&E providing the FAR message location

11- The S&E asks for the FAR and the FAR interface provides the FAR

12- The SCT receives SAML Auth Response sent by BS/S&E.

13-.The SCT validates the SAMLAuthenticationResponse

14- The SCT creates a unique session and set attributes as server variables (SL)

15, 16- The SCT provides to the user the requested resource

Sections 7, 8 and 9 will provide a detailed description on AIS components.

7 Access Filter (AF)

Taking into account the FutureID AIS architecture, described in section 9 of D44.3 Technical Specification for AIS document **[1]** the AF will filter the access to a requested resource by the user and will activate, if necessary, the authentication process in the application server JBoss.

Figure 2 displays the different building blocks of AIS component.

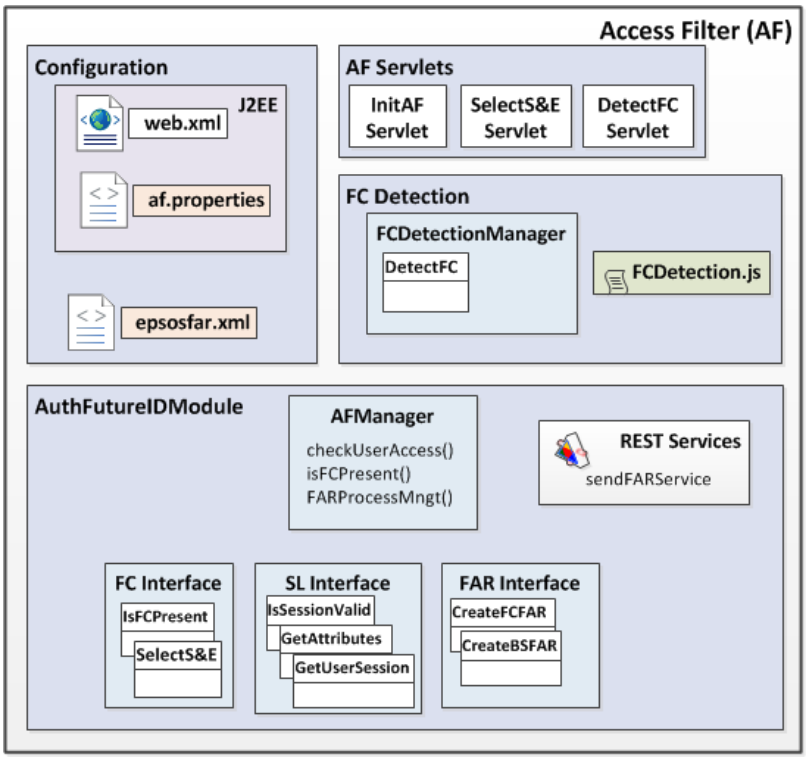

Figure 2: Access Filter components

The AF component is made of a service for authentication AuthFutureIDModule developed in Java which would be responsible for running the following actions:

- Checking whether the user's session is valid and the identity attributes are known;
- If not, send a FAR call to the BS. The requested attributes included in the FAR are provided by the *epsosfar.xml* configuration file;
- Check whether the FC is present in order to connect to the S&E associated to the FC or to an external one;
- Connect to the S&E component and sends it the FAR message.

7.1 FutureID scenarios

Depending on the presence of the FC in the user device we can consider two different scenarios.

7.1.1 FC present flows

In case the FC is present, the authentication flow is as follows: the UA requests a resource provided by the application A. The request is intercepted by AF. AF issues a FutureID Authentication Request (FAR) through the UA to the S&E. The SL is used to determine whether a user is known.

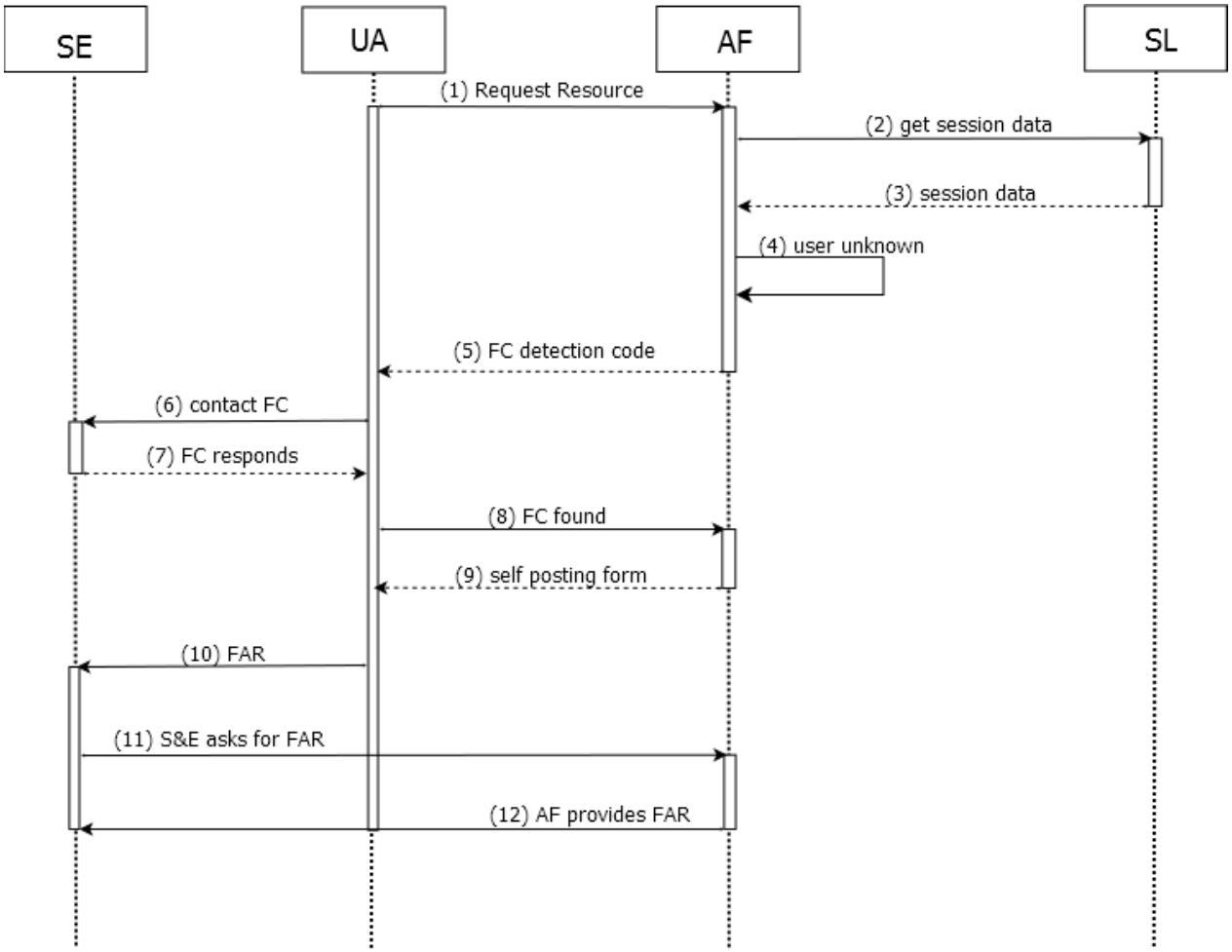

Figure 3: unknown user accessing a resource supplied by A, FC present

Figure 3 shows following steps:

1. The UA requests a resource. The AF intercepts the request;

2. The AF uses the SL to get the session data for the user in order to determine whether the user is known;

3. The SL doesn't provide the session data for this user;

4. The AF treats the user as an unknown user;

5. The AF checks whether the UA can access the FC, running the detection code;

6-7 The detection code runs on the UA trying to connect to the FC;

8. If FC responds AF knows that FC is present;

9-10. Then the AF makes a "self-posting form" sending a url as parameter, where the FAR message is located, to the S&E through the UA;

11. The S&E asks the FAR through the provided url;

12. The AF provides the FAR message to the S&E.

7.1.2 FC absent flows

In case the FC is not present, the UA requests a resource provided by the application A. The request is intercepted by AF. AF issues a FutureID Authentication Request (FAR) through the UA to the S&E. In this case the FC is absent and the user chooses a trusted external FutureID Broker who provides the S&E component that acts in the interest of the user. The SL is used by the AF to determine whether a user is already known.

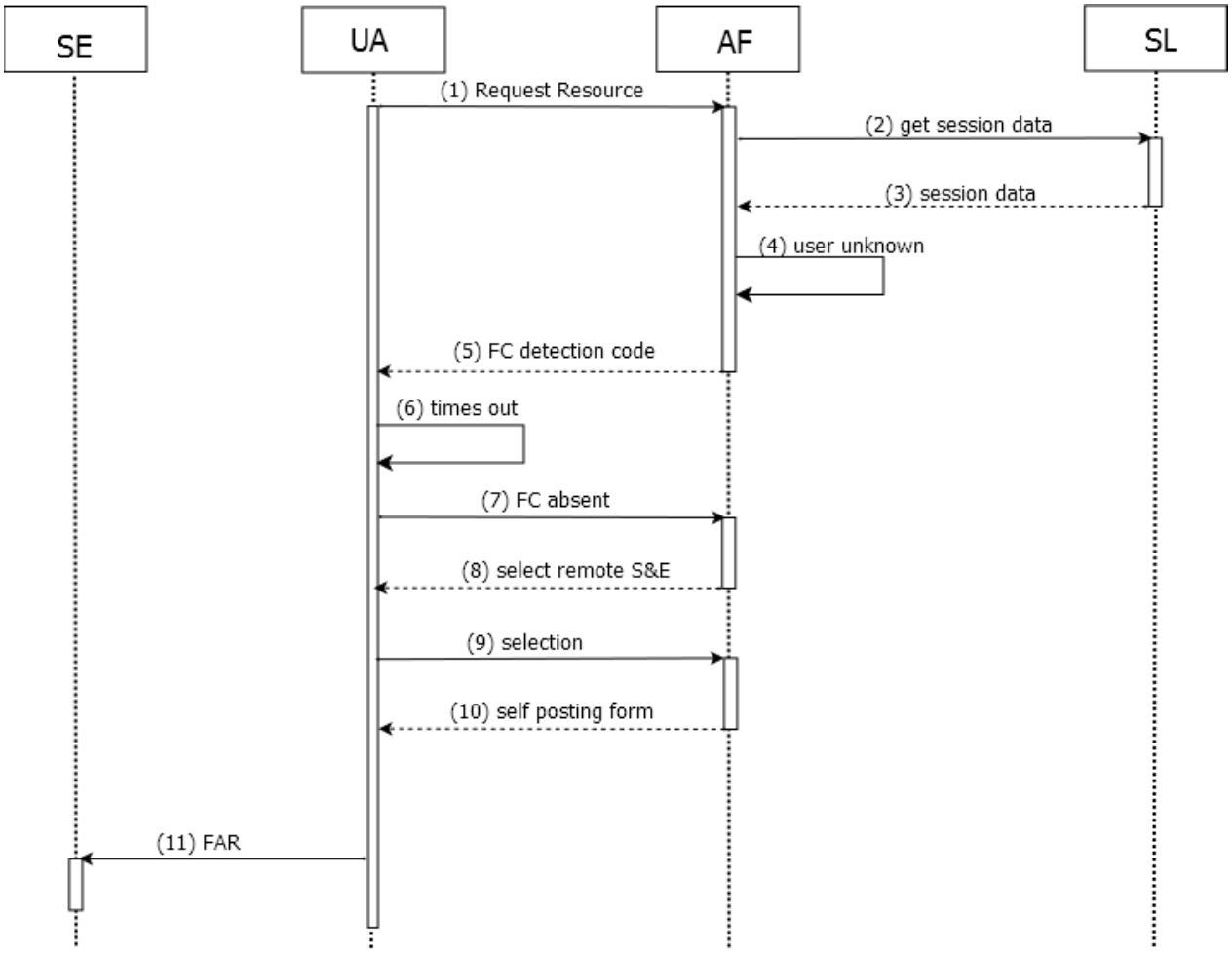

Figure 4: unknown user accessing a resource supplied by A, FC absent

Figure 4 shows following steps:

1. The UA requests a resource. The AF intercepts the request;

2. The AF uses the SL to get the session data for the user in order to determine whether the user is known;

3. The SL doesn't provide the session data for this user;

4. The AF treats the user as an unknown user;

5. The AF checks whether the UA contains the FC, running the detection code;

6-7 The detection code runs on the UA trying to connect to the FC, after the time out[1], the AF understands that the FC is absent;

8. The AF asks the user to choose a remote S&E;

9. The UA selects a remote S&E;

10-11. The AF makes a "self-posting form" sending a url, where the FAR message located, to the selected S&E through the UA;

12. The S&E asks the FAR through the provided url;

13. The AF provides the FAR message to the S&E.

7.2 JBoss AF Configuration

The configuration files structure is based on the next files (see Figure 5):

- web.xml
- af.properties
- epsosfar.xml

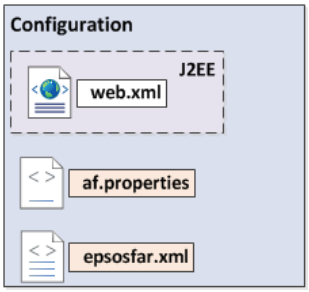

Figure 5: JBoss configuration files

[1] As FC is an application running on user's device the access to a local port is almost immediate, avoiding network issues.

7.2.1 web.xml

The web.xml file belongs to the J2EE standard. Both the epSOS application and the JBoss AIS contain this kind of file. This deployment descriptor file determines how URLs map to servlets (see section 7.3), which URLs require authentication, and additional information.

7.2.2 af.properties

This property file configures the AF´s behaviour. The following table gives an overview about the available properties.

Property	Description	Sample Value
af.sharingServer	If AIS and SP are running in the same server	true or false
far.login	Location of the FAR file which should be used to determine the required and optional user attributes for an authentication request	/opt/FUTURE-ID/FAR-login_ATOS_JBoss.xml
far.signup	Location of the FAR file which should be used to determine the required and optional user attributes for a registration request	/opt/FUTURE-ID/FAR-signup_ATOS_JBoss.xml
fc.url	User device URL for FutureID Client detection	http://localhost:24728/getVersion
fc.detectedCallback	AF servlet where the presence of FC is reported	http://localhost:8080/jboss_ais_art-web/FCDetectedServlet
fc.absentCallback	AF servlet where the absence of FC is reported	http://localhost:8080/jboss_ais_art-web/SelectSEServlet
fc.se.callurl	S&E URL to call when FC is present	http://localhost:24728/
bs.se.url	S&E URL to call when FC is absent. Define an external url for S&E	https://demo.futureid.org/se
se.farurllogin	Define the url where the S&E will call to get the far.xml for login	http://localhost:8080/FutureIDAIS/services/farRequest/sendFarServiceLogin
se.farurlsign	Define the url where the S&E will call to get the far.xml for sign up	http://localhost:8080/AccFilterWS/services/farRequest/sendFarServiceSign
sp.url.login	URL for SP application login	http://localhost:8080/EpsosFake/index.jsp
sp.url.main	URL for SP main entrance	http://localhost:8080/EpsosFake/epsosWelcome.jsp

Table 1: Properties of AF

7.2.3 epsosfar.xml

The information needed for the SAML authentication process will be provided by the epsosfar.xml configuration file. This file is based on SAML 2.0 metadata extensions. The

complete structure can be seen in Annex A. The epsosfar.xml file provides the following information:

- Attributes needed by the SP in order to create the FAR message:
  - Attributes to be requested by the SP as optional;
  - Attributes to be requested by the SP as mandatory;
- SCTs available;
- Issuers accepted.

7.2.3.1 Attributes in the Authentication process

The attributes sent by the user to AIS are both the mandatory attributes and the optional attributes selected by the user.

The AIS will send the following attributes (Table 2) to BS in the FutureID authentication request. Then this FAR reaches the BS.

Attribute	Mandatory	Optional
FirstName	X	
LastName	X	
Street		X
StreetNumber		X
City		X
State		X
Country		X
ZipCode		X
Nationality		X
DateOfBirth		X
IDType		X
IDIssuer		X
IssuingState		X
IDValidUntil		X
eIdentifier	X	
Age		X
AgeVerification		X

Table 2: Attributes of authentication request and response (Broker service)

The BS will connect with IdP and the information contained in the attributes received on the response from IdP is mapped to the BS response. Finally the AIS will receive an assertion from the BS. A high level flow showing how the FAR is involved in the authentication process can be seen in Figure 6.

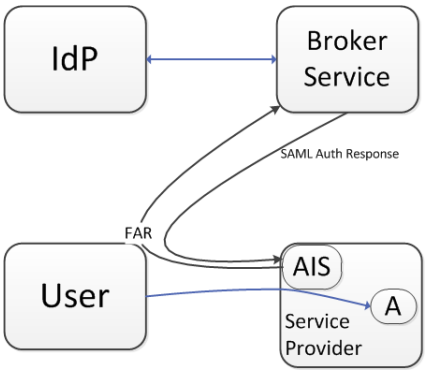

Figure 6: Flow of attributes for authentication process.

7.3 AF servlets

As the JBoss AIS component is based on Java technologies, servlets are used to respond to request from SP and communicate with the FutureID platform and the user.

7.3.1 InitAF servlet

This servlet receives the authentication request from the SP and also orchestrates the user validation process granting user access to the requested resource or triggering the authentication process.

7.3.2 SelectS&E servlet

Servlet responsible for generating a web page which enables user to select an external IdP in case the FC is not present or do not operate in the S&E mode.

7.3.3 DetectFC servlet

This servlet is responsible for generating a web page, which loaded into the browser, it's able to detect whether the FC is running and operates in the S&E mode. This is realized through the means of AJAX calls to the FC.

7.4 AuthFutureID service

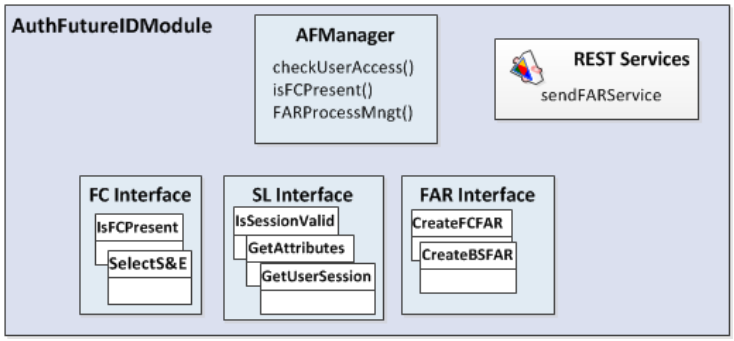

Figure 7: AuthFutureIDModule components.

The AuthFutureID service is implemented in Java and it works as a web service. Figure 7 shows the different components of the AuthFutureID service. The functionalities provided by the AuthFutureID module are outlined below, and the main methods are described in next subsections:

- AuthFutureID checks whether the user's session is valid and the identity attributes are known;
- Checks whether the FC is present in the user device;
- If session is not valid, send a FAR call to the BS. The attributes requested included in the FAR are provided by the epsosfar.xml configuration file;
- It contains a service providing the FAR message;
- It provides an interface to connect to the SL in order to check the user session and needed data.

7.4.1 AFManager

checkUserAccess()

Name	checkUserAccess
In parameters	futureIDSessionID, UrlResource
Out parameters	boolean
Description	Orchestrates the user access to the requested resource. Grants user the access to requested resource if user session data are matched. Denies the access otherwise.
Exceptions	AFException

isFCPresent()

Name	isFCPresent
In parameters	none
Out parameters	boolean fcpresent
Description	Manages the FC dtetection process. Returns

	true if FC is present and false otherwise.
Exceptions	AFException

initFARProcess()

Name	initFARProcess
In parameters	boolean fcpresent, URL resourceRequested, String typeOfAuthentication
Out parameters	none
Description	Initiates the creation of FAR message. Receives whether the FC is present, the requested resource and if the user wants to authenticate by FutureID or to register.
Exceptions	AFException

7.4.2 FC Interface

IsFCPresent

Name	isFCPresent
In parameters	none
Out parameters	boolean
Description	Trigger the FC detection process through the detectFC.jsp. Returns true if FC is present and false otherwise
Exceptions	AFException

SelectS&E

Name	selectSE
In parameters	none
Out parameters	String SE
Description	Triggers the S&E selection process. Returns the name of the S&E choosen by the user.
Exceptions	AFException

7.4.3 SL Interface

Interface and implementation class that retrieve user data from session, through the Session Library (SL).

GetUserSession

Name	getUserSession
In parameters	String session
Out parameters	SessionData

Description	Retrieves the session data using the session identifier passed as parameter
Exceptions	AFException

IsSessionValid

Name	isSessionValid
In parameters	SessionData sessionData, boolean isLogin
Out parameters	boolean
Description	Check whether the session passed as parameter for the provided action (login or sign-up) is valid or not.
Exceptions	AFException

7.4.4 FutureID Authentication Request

Interface and implementation classes managing FAR (FutureID Authentication Request) message have been implemented. FAR message will be included in a SAML message as a XML message.

7.4.4.1 FAR Message

The final specification of FAR message has been established in this deliverable and is included in Annex A.

Figure 8 depicts how the FAR message is created by the AF component and reaches the BS. After the connection with the IdP the provided final response is not a response as is, but an IdP-initiated SAML response. This case is valid when the FC is absent, then the S&E element is not present on the user side.

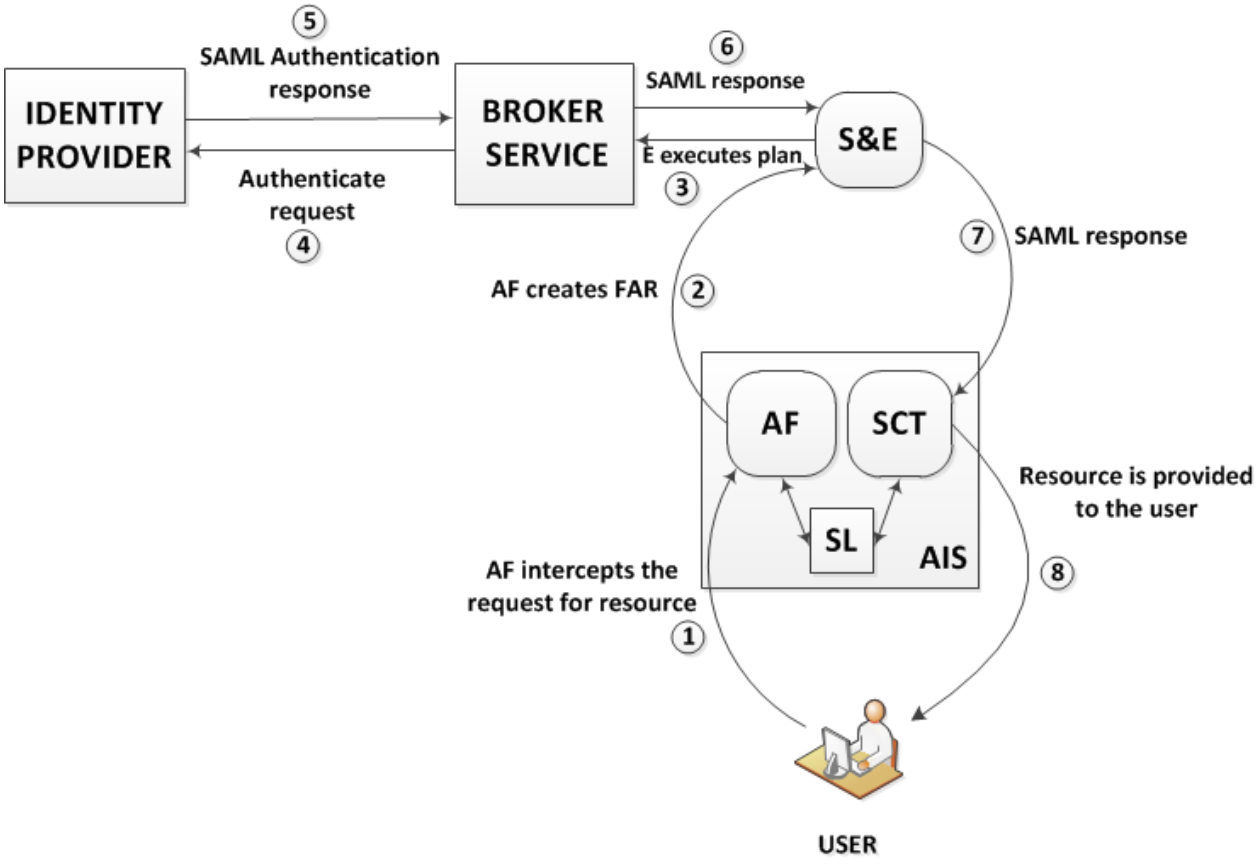

Figure 8: FAR message for FC absent

As can be seen in step 2 from Figure 8 the FAR is an independent process. The AF initiates the FAR to the S&E component. Then the authentication process is carried out by the Identity Provider (steps 4 and 5). Finally another component of the Broker Service generates the IdP-SAML-Response that reaches the SCT component (step 7). For this reason it is not necessary to include a session identifier in the dynamic part of FAR.

7.4.4.2 FARInterface

CreateFCFAR

Name	createFCFAR
In parameters	none
Out parameters	void
Description	Initiates the FAR process when FC is present
Exceptions	AFException

CreateBSFAR

Name	createBSFAR
In parameters	String selectedSolver
Out parameters	void
Description	Initiates the FAR process when FC is absent and taking into account the selected solver passed as parameter
Exceptions	AFException

7.5 REST Services

The URL where the AF has to send the FAR message will be provided by ECSEC (it has been made configurable, see af.properties file in section 7.2.2). One URL for each kind of FAR message has been created:

- a URL to the S&E of the FC, when FC is present and operates in S&E mode;
- a URL to the external S&E or does not work in the S&E mode, when FC is absent.

The AF sends to the S&E the URL where the S&E will get the xml file with the FAR message. The AF uses a REST service to provide the appropriate xml file. The AIS (SCT component) will wait for the SAML Authentication Response.

The entry point for this rest service is included in next table.

Name	Entry point
sendFARService	https://futureid.atosresearch.eu: /FutureIDAIS/services/farRequest/sendFarServiceLogin

Table 3: sendFAR service entry point

7.6 FC Detection

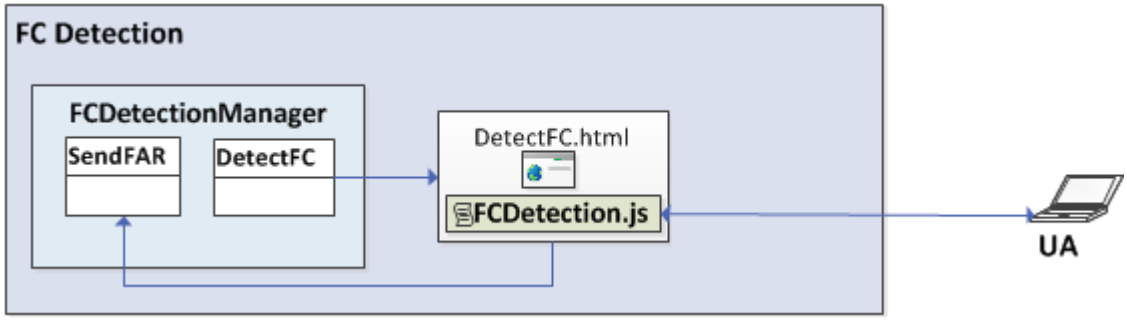

Figure 9: FC detection process

FC detection is a common process (see Figure 9) for both FutureID AIS the JBoss specific AIS and the Apache specific AIS implementation.

The solution for FutureID client detection is based on Java Servlets and Jscript technology. Central part of the FC detection mechanism lies in AJAX calls to the FutureID client performed by Jscript code. For this purpose a `CheckFCPresenceServlet.java` servlet was created. This servlet is reachable by calling:

`[SP URL address]/checkFCPresence`

Response to this call is a code of a `CheckFCPresence.jsp` file filled with parameters containing URL parameters. These parameters are needed for accessing the FutureID client, sending HTML form data either to the servlet responsible for creating FAR message or the servlet responsible for selecting another S&E.

These parameters are stored inside the af.properties file (war resources):

Parameter	Value	Description
`fcURL`	`http://127.0.0.1:24727/getStatus`	A URL address to which AJAX call is made. This assumes that FutureID client is running and is capable of receiving HTTP requests. As a response an xml data is send
`fcSendFARCallback`	`http://localhost:8080/jboss_ais_art-web/testSendFAR`	A URL address to which HTML form is sent in case of FutureID client was detected and returned xml indicates that FutureID client state is the one that supports S&E feature. In current implementation it is a test servlet
`fcSelectSECallback`	`http://localhost:8080/jboss_ais_art-web/testSelectSE`	A URL address to which HTML form is send in case of FID client was not detected or returned xml indicates that FutureID client state does not support S&E feature. In current implementation it is a test servlet

Table 4: FC detection parameters included in af.properties configuration file

Once Jscript code of a `CheckFCPresence.jsp` file loads into the page it performs AJAX requests to the FutureID client. There are two possible options, either response comes from FID client or not.

If there is a response to the AJAX call it is expected to be in the xml format. The xml data is then further analysed. Even if there was a response form FutureID client, returned xml must indicate that FID client state is the one that supports S&E feature. A special node named “AdditionalFeatures” carries this information. If such information exists, a jscript code responsible for receiving it, submits a proper HTML form to the address stored inside fcSendFARCallback parameter. As shown on the picture below it is the address of a servlet responsible for FAR message creation. Inside the form data there can also be information about a version of a detected FutureID client.

If those condition are not fulfilled, then the web page gets redirected to the servlet responsible for selecting another S&E.

That is, in the second case when a response does not come or it came but information stored inside the special node named “AdditionalFeatures” indicates that FC does not operate in the S&E mode, the page gets redirected to servlet responsible for selecting another S&E.

Figure 10 below shows relation between different servlets and communication between AF and UA.

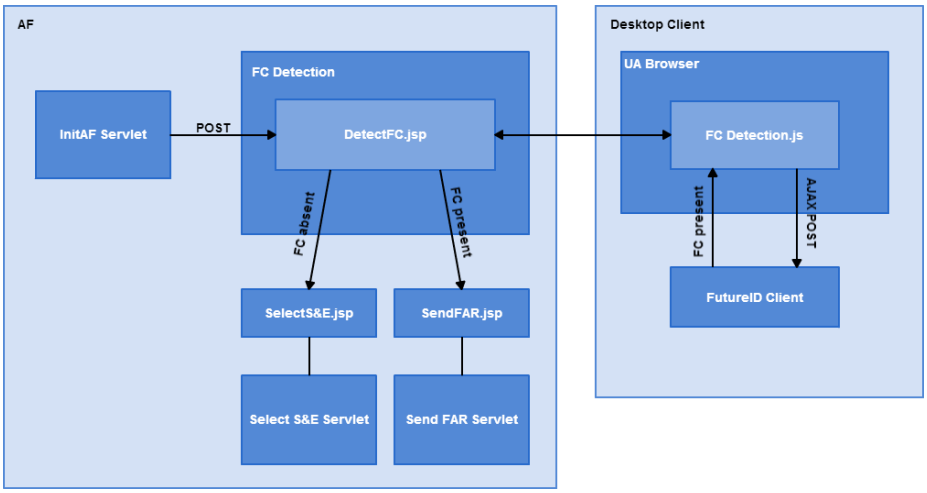

Figure 10: Detailed flow of FC detection process

7.6.1 FCDetectionManager

DetectFC

Name	
In parameters	none
Out parameters	XML with the description of FC status.
Description	FC accepts requests under the following url: http://127.0.0.1:24727/getStatus Returned xml in the section: "ns2:AdditionalFeatures" indicates whether FC operates in the S&E mode or not.
Exceptions	No response or generic server response exceptions.

7.6.2 FCDetection.js

The FCDetection javascript code is embedded into a jsp page and is a common component for both JBoss specific AIS and Apache specific AIS implementation.

8 Simple Credential Transformer (SCT)

The Simple Credential Transformer (SCT), as an AIS component, receives assertion tokens from the S&E, which contain needed attributes to grant access to the resources, requested by the user. In this section the process and the component of the SCT will be described.

8.1 SCT overall overview

As last step in the authentication process the Simple Credential Transformer (SCT) comes into play. The SCT receives an HTTP Post request from the S&E or BS. This request contains two hidden fields as request parameters – the SAML authentication response message and the RelayState URL.

The SAML authentication response message comprises the result of S&Es authentication plan (authentication successful or not) and the user credentials (given user attributes). This message is Base64 encoded to avoid encoding problems during transportation. The RelayState URL indicates the resource the user wants to call up. The URL typically has two query parameters attached. The type parameter indicates what kind of authentication the user wants to perform (either he wants to login on the requested resource or register to a new resource) and the HMAC parameter is used to verify the integrity of the RelayState URL. With the URL itself (without HMAC parameter) and a configurable secret (see 8.3.2) the S&E has calculated a security digest via cryptographic hash function SHA256 and add it to the RelayState URL. By comparing the delivered message authentication code from S&E with the digest computed in SCT, the URL will be protected against any manipulation during transport.

Besides interpretation and computation of the RelayState parameter, the validation of the SAML authentication response message is the essential purpose of the SCT. This is done by using the OpenSAML2 libraries and comprises several validation steps for status, validity, and security analysis (see 8.2.2). The implementation also uses an internal AssertionCache (see 8.3.1) for caching already processed SAML assertions in order to prevent Replay attacks. Extracting the user credentials from SAML message to verify them against the credential policy (required and optional data fields) given by the appropriate FAR message (xml file) is also done inside the validation process.

After successful SAML validation the SCT is in charge of creating a user session in the Session Library (SL). For that SCT is using the HTTPS REST interface provided by the SL component to invoke the SLManager and transmit the current user credential information as session data.

The final function of the SCT is to redirect the user to its requested resource if the whole authentication process was successful and the user has gotten valid user session. Otherwise an error page will appear informing the user about the reasons for this unsuccessful authentication.

8.1.1 SCT process flows

This section and Figure 11 gives an overview of the process flow.

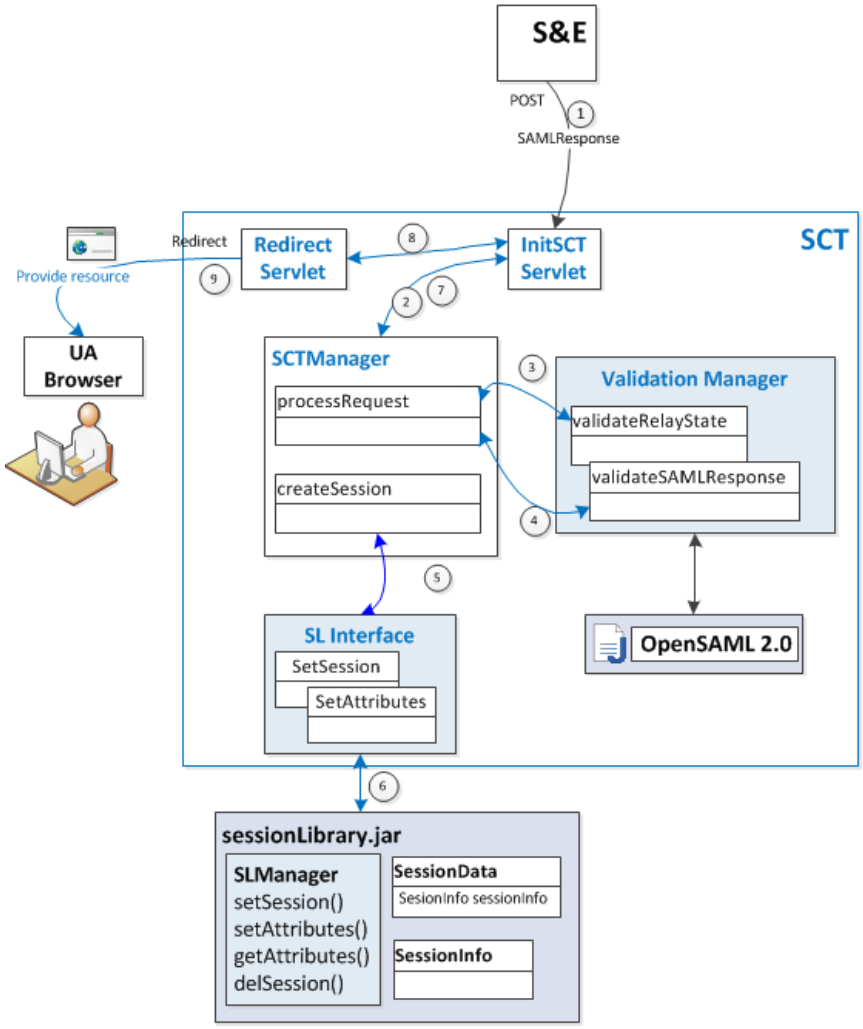

Figure 11: SCT process flows

1: S&E triggers HTTPS Post Request to transmit SAML Response message

2: Request is intercepted by InitSCTServlet to extract request parameters. This servlet invokes SCTManager in order to process parameter data.

3: SCTManager instructs the ValidationManager to validate and process the RelayState parameter

4: SCTManager instructs ValidationManager to validate SAML authentication response message using OpenSAML 2.0

5: SCTManager invokes the SLInterface to submit the user data for session creation

6: SLInterface creates a user session with a suitable set of SessionData attributes

7: SCTManager conveys processing results to the InitSCTServlet

8: InitSCTServlet dispatches result to the RedirectServlet

9 RedirectServlet redirects user to the requested resource or an error page

8.2 SCT components

The Simple Credential Transformer consists of the following components depicted in **Figure 12**:

- SCTManager – as the main component, manages the processing of the incoming request parameters (see 8.2.1);
- ValidationManager – executes the validation of incoming parameters by utilizing the OpenSAML 2.0 library to access and verify the SAML message (see 8.2.2);
- SLInterface – a successful SAML validation and processing of RelayState URL is followed by the creation of a vaid user session in the Session Library (see 8.2.3) ;
- SCTServlets – used for the interaction with the user, intercepting the requests and deliver a processing result (see 8.2.4).

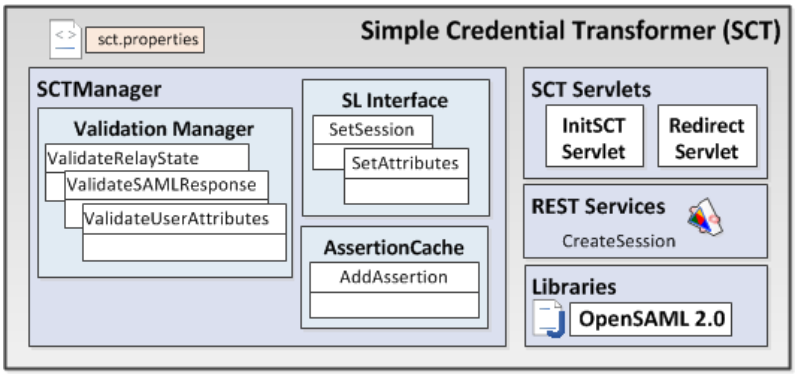

Figure 12: SCT components

8.2.1 SCTManager

The SCTManager is an element which orchestrates all necessary tasks for processing incoming requests to an appropriate response. In detail the SCTManager receives the given user attributes from the InitSCTServlet, invokes the verification of the given RelayState Parameter as well as the validation of the given SAML-message. The SCTManager also controls the session

handling by creating new user sessions in the session library component (see section 9.1) and filling the internal session cache. Beside this, the handling of occurring errors and creation of result object are done by this class.

8.2.1.1 getInstance()

Name	**getInstance**
In parameters	none
Out parameters	SCTManager instance The instance of the SCTManager class
Description	The SCTManager is implemented as a singleton. By calling this method, you will get the current instance of the SCTManager class.
Exceptions	none

8.2.1.2 processRequest()

Name	**processRequest**
In parameters	byte[] samlResponse Contains the given SAML message as byte array-------- String relayState Contains the redirect url with parameters for HMAC security token and request type
Out parameters	SCTResult result Self-implemented java object to gather all step results and occurring errors during processing
Description	Executes and controls the processing of incoming requests by performing • Verification of RelayState parameter • Validation of SAML message • Creation of user sessions • Error handling • Result creation
Exceptions	none

8.2.2 ValidationManager

This element is responsible for verification and validation of incoming request parameters. A list of provided functionalities by this class is given below:

- Initialize OpenSAML2 library
- Initialize and operate assertion cache (see 8.3.1), if activated via configuration file (see 8.3.2)
- Validate RelayState parameter
  - Determine HMAC and request type query parameters
  - Detect request type ('login' or 'signup') and select corresponding FAR file
  - Validate HMAC security token
- Validate SAML message

- Verify authentication status code of the message
- Validate message Issuer
  - Issuer data is present and assertion issuer is on the configurable list of trusted issuers (see 8.3.2). At this point it is possible to integrate an external Trust Service (TS) to guarantee trust status of the issuer.
- Validate message validity period
  - AssertionCondition (IssueTimestamp, validity start, validity end) are present and IssueTimestamp is before validity period, current time is after IssueTimestamp and current time is within validity period.
- Validate message credentials
  - Signature of the SAML Authentication Response message is validated.
- Validate message attributes
  - Determine delivered user attributes in SAML message
  - Load required/requested attributes from FAR-file
  - Verify that all required attributes are given by SAML message

All public methods in this class do not throw any exceptions that means, occurring errors are handled by the implementation itself, logged and forwarded as an error in the result objects.

8.2.2.1 getInstance()

Name	**getInstance**
In parameters	none
Out parameters	ValidationManager instance The instance of the ValidationManager class
Description	The ValidationManager is implemented as a singleton. By calling this method, you will get the current instance of the ValidationManager class.
Exceptions	none

8.2.2.2 validateRelayState()

Name	**validateRelayState**
In parameters	String relayState Contains the RelayState parameter of the incoming request
Out parameters	RelayStateValidationResult result Self-implemented java object to gather step results (RelayState URL, FAR-Type, Validation Results) and occurring errors
Description	Extracts HMAC token and request type from the incoming RelayState string. Validates the HMAC security token to ensure integrity and trustworthiness of the delivered RelayState. Also determine the appropriate, static FAR-file for the current request type.
Exceptions	none

8.2.2.3 validateSAML()

Name	**validateSAML**
In parameters	byte[] samlResponse SAML message as Byte-array FarType farType enumeration type which is linked with the static FAR file, which should be used
Out parameters	SamlValidationResult result Self-implemented java object to gather step results (results of validation steps), required and optional attributes, the assertion id and occurring errors
Description	This method validates the SAML message itself. First the method verifies that the status code of the SAML message is 'urn:oasis:names:tc:SAML:2.0:status:Success'. If this check is passed, the next validation step ensures that all needed data fields are available in the message (issuer, assertion, conditions) and the current time is in the defined validity period. If this step is passed, the credential validation takes place. For that the embedded signature is extracted and validated using OpenSAML functionalities. The next step is the verification of the delivered user attributes. For that the required and optional fields are extracted from the FAR file (according to the given FarType) and matched with the user attributes delivered by the SAML message. If all required attributes are available in the SAML message, this validation step is passed. If the Assertion Cache is activated (via sct-properties file), the ID of the current assertion / SAML message is added to the cache. This enables to prevent 'Replay Attacks' through reusing an assertion / SAML message several times, because the assertion cache throws a SCTException if an assertion ID is already known/present.
Exceptions	SCTException

8.2.2.4 addSessionDataToCache()

Name	**addSessionDataToCache**
In parameters	String assertionID ID of the current assertion String sessionID ID of the created session for the processed assertion
Out parameters	none
Description	Adds the sessionID to the cache entry associated with the processed assertion (identified by its ID)
Exceptions	none

8.2.3 SL Interface

Interface and implementation class to create and delete user data session, through the Session Library (SL) component.

8.2.3.1 getInstance()

Name	**getInstance**
In parameters	none
Out parameters	SLInterface instance

	The instance of the SLInterface class
Description	The SLInterface is implemented as a singleton. By calling this method, you will get the current instance of the SLInterface class.
Exceptions	none

8.2.3.2 createSession()

Name	**createSession**
In parameters	HashMap<String, String> attributes
Out parameters	SLResult result Self-implemented java object to gather step results (SessionData with sessionID, sessionInfo and aesData) and occurring errors
Description	Creates a new user session in the session library and enrich the session data (SessionInfo) with all required and optional attributes from the processed SAML message.
Exceptions	none

8.2.3.3 delSession()

Name	**delSession**
In parameters	String sessionID The ID of the session to be deleted
Out parameters	none
Description	Deletes the session with the given SessionID from the session library. This method will only be executed, if the assertion cache is activated and the cleanCache() method is called by the AssertionCache timer. Only sessions with an expired assertion (checked by expiration date of the assertion) will be deleted.
Exceptions	SessionLibraryException, SessionIdNotFoundException

8.2.4 User Interface / SCT Servlets

To receive incoming HTTP requests and send an appropriated HTTP response we decided to implement a small servlet chain, which consists of the InitSCTServlet and RedirectServlet. The configuration of the web application is done in src/main/webapp/WEB-INF/web.xml (Annex E). Here you can define the name of the web application, its welcome-file and the available servlets including the mapping between an URL-pattern and the associated servlet.

8.2.4.1 InitSCTServlet

This Servlet processes HTTP Post Request from S&E component. HTTP Get Requests are automatically forwarded to the Post-interface. The servlet extracts the given parameters 'RelayState' and 'SAMLResponse' from the request into String variables. Because the SAMLResponse parameter is delivered Base64 encoded, the InitServlet encodes the given String. Afterwards the 'prepared' variables are passed to the processRequest() method of the

SCTManager for further processing. The returning result object (SCTResult) is added as an additional attribute to the original request. After this, the request is forwarded via a RequestDispatcher to the RedirectServlet, which is responsible to create an appropriate response.

8.2.4.2 RedirectServlet

This servlet interprets the SCTResult from the previous processing steps and decides which output is given to the user. If the overall processing result is not 'SCTResultType.Error', this servlet generates an HTTP response with an automatic redirect to the RelayStateURL after 5 seconds. The content of this page includes a list of occurred warnings during processing.

The RelayStateURL which is used to redirect the server, is the incoming RelayState Parameter from the initial request without the HMAC token part. All other URL parameters are unmodified. Only the ID of the session, created during processing of the request, is attached.

In case of an unsuccessful processing of an incoming request, the generated output page (error-page) contains a list of occurred errors and warnings and a link to a configurable error page (see8.3.2). An automatic redirect does not exist in this case.

8.3 SCT development features

8.3.1 AssertionCache

The AssertionCache is a special feature of the SCT to prevent Replay Attacks, where an unauthorized user tries to reuse an intercepted SAML message again. The AssertionCache can be activated via property 'CacheActive' in sct.properties file.

If a SAML message is completely validated without problems and the AssertionCache is activated, the current assertionID and the expiration date of this assertion is inserted into the cache. If this assertionID is already known (existing in the cache) a processing error occurs and the whole SAML validation fails for this message. So no user session is created and the user gets the 'error-page' from the Redirect-Servlet.

If the validation of the SAML message was successful and the user session was created without problems, the sessionID is also added to the cached data for this assertion. This is done via method 'addSessionDataToCache()' from ValidationManager class.

It is also possible to adjust the time (in milliseconds) between two 'clean runs' of the cache via property 'CacheCleanTimer' in sct.properties file. The cache is implemented to automatically start a clean job after the CacheCleanTimer is exceeded. The clean run checks every entry in the cache, if its expiration date is in the past in the meantime. If this is the case, then the session is deleted from the session library and the entry is discarded from the cache.

8.3.2 sct.properties file

This is a property file to configure several aspects of the SCT. The following table gives an overview about the available properties.

Property	Description	Sample Value
ErrorURL	URL which is displayed on the error-page to route the user to e.g. a manual login page	http\://www.epsos.eu
ValidationTestMode	If SCT runs in TestMode (value = true), validation of IssueDate and Validity Period of the SAML message is disabled. Otherwise testing with a static SAML message is impossible.	true or false
TrustedIssuers	Comma-separated list of trusted issuers. Used to verify that the issuer of the SAML message is a trusted one.	https\://www.skidentity.de/fs/saml
CacheCleanTimer	Time between two 'clean runs' of the Assertion Cache in milliseconds.	3600000
CacheActive	Determines if the Assertion Cache is active or not.	true or false
LoginFAR	Name of the FAR file which should be used to determine the required and optional user attributes for a login request.	FAR-login_ATOS-eLearning_SP-metadata_v2.xml
SignUpFAR	Name of the FAR file which should be used to determine the required and optional user attributes for a registration request.	FAR-signup_ATOS-eLearning_SP-metadata_v2.xml
HMACsecret	Secret string for HMAC validation. Used to calculate the digest for validating the authenticity of the given RelayState parameter.	future.id@jboss.ais
loginType	Value of the 'type' parameter in RelayState parameter of login requests. Used to determine the matching FAR file.	login
registrationType	Value of the 'type' parameter in RelayState parameter of registration requests. Used to determine the matching FAR file.	register

Table 5: Properties of SCT

8.3.3 FAR files

The two FAR files are used to determine the required and optional user attributes for a login or a registration request. These static files must be placed in the resources folder of the application.

8.3.4 Testing via testing.html

The SCT project contains a test file to check correct functionality of the component. If the SCT is started without any problems, the testfile should be available by the URL 'https://localhost:8443/sct/testing.html'. It contains a Base64 encoded SAML message and a sample RelayState as hidden fields. After pressing the submit button, an HTTP Post request is send to the InitSCTServlet. The SCT is processing the request and automatically redirects to the

configured RelayState-URL. For simplicity the RelayState-URL of this test requests points to this testpage again. A successful processing can be noticed by the new URL parameter 'sessionID'.

This file is located in the webapp/WEB-INF folder of the project.

8.3.5 SAML message for testing

A valid SAML message for testing is placed in resources folder of the application, named SAMLResponse-forDev.txt. This message is also used for the testing.html page.

9 Session Library (SL)

The SL is used by both the AF and the SCT. The AF uses it to determine whether a user is known and the SCT sets a user session by calling according services of the SL. SCT also uses SL to set the attributes as session data from the user in order to determine whether the user is known; The SL sets a cookie to include at the user side;

Different alternatives have been investigated in order to choose an easy to use library that covers the basic requirements for controlling the session management. PicketBox framework from JBoss and javax.servlet.http.HttpSession interface were considered. Finally a specific tailor-made set of classes has been defined to use as interface for SL. Figure 13 shows the building blocks of SL. The SLManager acts as the SL interface and the remaining three classes contain the user information during the session. All of them are included into the sessionLibrary.jar package:

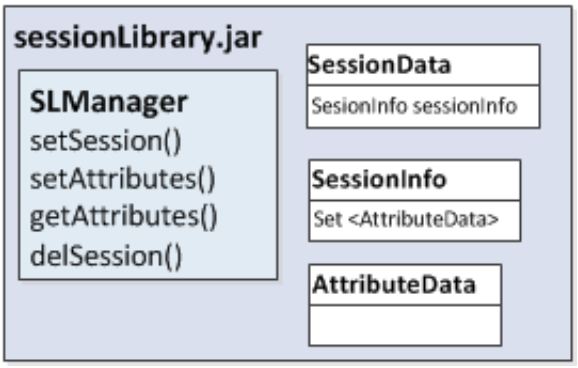

Figure 13: Session Library components

9.1 SLManager

SLManager will be the Interface to manage user data session. The SLManager will use the SessionData class in order to get access to the server session environment. Four methods and three classes will be offered in the “sessionLibrary.jar”. A description for these four methods and classes is provided in the following sections.

9.1.1 setSession

Name	**setSession**
In parameters	none
Out parameters	SessionData sessionData: Object sessionData will include the sessionID and the last accessed time
Description	Sets a new authenticated session for the current user. Returns: The created sessionData object
Exceptions	SessionLibraryException

9.1.2 setAttributes

Name	**setAttributes**
In parameters	SessionData sessionData: A sessionData object which contains the sessionID and the attributes that should be set
Out parameters	SessionData sessionData: sessionData object that will contain the session information: sessionID (it will return this data if it wasn't included in input parameters), last accessed time and attribute list (updated)
Description	If session is null it creates a new session and includes the attributes passed as parameter. If the session passed as parameter is a valid session sets the new attributes passed as parameters. Returns: - A sessionData object with attributes - A SessionLibraryException if the session is not valid
Exceptions	SessionLibraryException

9.1.3 getAttributes

Name	**getAttributes**
In parameters	String: sessionIdentifier: sessionIdentifier: Identifier of an established session.
Out parameters	SessionData sessionData sessionData object that will contain the session information: sessionID, last accessed time and attribute list
Description	Access to the server environment identity attributes for the current user from the application A. Returns an object sessionData including the session attributes
Exceptions	SessionLibraryException, SessionIdNotFoundException

9.1.4 delSession

Name	**delSession**
In parameters	String: sessionIdentifier sessionIdentifier: Identifier of an established session.
Out parameters	none
Description	Delete the session for the sessionIdentifier Returns: void
Exceptions	SessionLibraryException, SessionIdNotFoundException

9.1.5 SessionData Class

This class is defined to contain the complete session information. It uses the class SessionInfo which contains the session information and is described in subsection 9.1.6.

SessionData
String: sessionID SessionInfo: sessionInfo AesData: aesData
setSessionID() getSessionID() setSessionInfo() getSessionInfo() setAesData() getAesData()

9.1.6 SessionInfo Class

This class contains the information related to a concrete session, which is the last accessed time that the session was required and the attributes list

SessionInfo
long: lastAccessedTime Set <AttributeData>: attributeList
setlastAccessedTime () getlastAccessedTime () setAttributeList () getAttributeList()

9.2 AttributeData Class:

This class contains information for every needed attribute; this information is provided by the FAR xml file and by the SAMLAuthenticationResponse.

AttributeData
String: attributeName String: value String: description String: reason boolean: mandatory
set() methods get() methods

9.3 Error and Exception management

The SessionLibraryException and the SessionIdNotFoundException class have been defined.

9.4 Session Library installation

Session Library uses a custom made REST (Representational State Transfer) web services to manage the session information. The communication with the web services is secured by two ways: encryption and using the algorithm “Mutual authentication based in challenge response”. The configurations that have to be made are described in Annex B.

The configuration for logs files used by the REST services is detailed in Annex C.

Annex D contains the instructions to install the Session Library.

10 Development and deployment environment

Table 6 summarizes the development and deployment environment for the AIS components::

Feature	Version
Java version	jdk 1.7.0_55+
REST services	Jersey library version 1.8
Communication protocol	https, using protocol TLSv1
Server	jboss-as-7.1.0.Final

Table 6: Development and deployment environment features

11 Conclusions

The JBoss Application Integration Service has been implemented using Java technologies. There are two major roles covered by the AIS in the FutureID architecture:

- Intercepting unknown users and requesting the FutureID infrastructure to authenticate them;
- Receiving and validating credentials in order to set up authenticated sessions for users.

With this aim the former is covered by the implemented Access Filter component and the latest is covered by the Simple Credential Transformer implementation. AF includes the FutureID client detection. Both components AF and SCT are sharing the SL component which manages the user session.

Annex A: FAR Structure

```
<?xml version="1.0" encoding="UTF-8"?>
<saml2p:AuthnRequest
        xmlns:saml2p="urn:oasis:names:tc:SAML:2.0:protocol"
        xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
        xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion"
        xmlns:futureid="urn:eu:futureid:names:tc:FutureID:1.0:far"
        AssertionConsumerServiceURL="https://url_of_AIS/ReturnPage"
        Consent="urn:oasis:names:tc:SAML:2.0:consent:unspecified"
        Destination="https://destination_of_far_message/path"
        ForceAuthn="true"
        ID="_943b7edc5ba2b869a08b72e2aceb9e36"
        IsPassive="false"
        IssueInstant="2014-10-14T12:19:12.569Z"
        ProtocolBinding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST"
        ProviderName="FutureID AIS on name_of_SP" Version="2.0">
<saml2:Issuer>https://url_if_AIS/metadata</saml2:Issuer>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <ds:SignedInfo>
                <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                <ds:SignatureMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256"/>
                <ds:Reference URI="#_943b7edc5ba2b869a08b72e2aceb9e36">
                        <ds:Transforms>
                                <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-
signature"/>
```

```
                    <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                </ds:Transforms>
                <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                <ds:DigestValue>...digest value using digest method defined before...</ds:DigestValue>
            </ds:Reference>
        </ds:SignedInfo>
        <ds:SignatureValue>...signature of this SAML message using algorithms referenced
before...</ds:SignatureValue>
        <ds:KeyInfo>
            <ds:X509Data>
                <ds:X509Certificate>...digital certificate associated to the private key used for the
signature...</ds:X509Certificate>
            </ds:X509Data>
        </ds:KeyInfo>
</ds:Signature>
<saml2p:Extensions>
<futureid:FAR>
        <futureid:name>name_of_SP</futureid:name>
        <futureid:attribute>
            <futureid:attrName>eIdentifier</futureid:attrName>
            <futureid:attrDescription>User electronic identification number</futureid:attrDescription>
            <futureid:attrReason>This attribute is needed to open the right user profile in the
                application</futureid:attrReason>
            <futureid:attrMandatory>true</futureid:attrMandatory>
        </futureid:attribute>
        <futureid:credential>
```

```
            <futureid:credName>Cross-border authentication with STORK</futureid:credName>
            <futureid:credIssuer>STORK</futureid:credIssuer>
            <futureid:credFormat>SAML.bearer</futureid:credFormat>
        </futureid:credential>
        <futureid:credential>
            <futureid:credName> ABC national eID card</futureid:credName>
            <futureid:credIssuer>ABC national certification authority</futureid:credIssuer>
            <futureid:credFormat>x509</futureid:credFormat>
        </futureid:credential>
        <futureid:SCT>
            <SCTname>Example SAML bearer SCT</SCTname>
            <SCTAcceptedFormat>SAML.bearer</SCTAcceptedFormat>
            <SCTAcceptedIssuers>Example Broker Service 1</SCTAcceptedIssuers>
        </futureid:SCT>
</futureid:FAR>
</saml2p:Extensions>
</saml2p:AuthnRequest>
```

Annex B: Security for the REST services in Session Library

1. Introduction

Session Library uses a custom made REST web services to manage the session information. The web services have been implemented taking into account the security and the following are the ideas that have been implemented:

- The communication between the client (Session Library) and the web service should be encrypted. So HTTPS will be used;
- Authentication: SL and Web services uses Mutual authentication based in Challenge response.

2. HTTPS

2.1 Server configuration

2.1.1 STEP 1: installing keystore

Take the file containing the keystore provided in the installation (futureid.keystore) and place it in jboss installation (\\jboss-as-7.1.0.Final\standalone\configuration)

2.1.2 STEP2: JBoss configuration

SL REST Web Services are running in jboss (JBoss-as-7.1.0) and the following code should be included in C:\jboss-as-7.1.0.Final\standalone\configuration\standalone.xml:

```
<subsystem xmlns="urn:jboss:domain:web:1.1" default-virtual-
server="default-host" native="false">

   <connector name="http" protocol="HTTP/1.1" scheme="http" socket-
binding="http"/>

   <connector name="https" protocol="HTTP/1.1" scheme="https" socket-
binding="https" secure="true">

      <ssl name="futureid-ssl" key-alias="futureid" password="20150119"
certificate-key-file="/home/futureid/jboss-as-
7.1.1.Final/standalone/configuration/futureid.keystore"
protocol="TLSv1"/>

   </connector>
```

```
...
</subsystem>
```

We must now stop and start JBoss AS

And now accessing in the browser to https://localhost:9443/ it asks about confirming security exception, before confirming the security exception, we must store the public certificate futureidec.cer.

2.1.3 STEP3: import the certificate in cacerts

Open a cmd (with root permissions) and do the following to import the certificate in cacerts (futureidec.crt is also provided with the installation):

```
[certificates stored path]> keytool -import -alias localhost -keystore
"C:\Program Files\Java\jdk1.7.0_55\jre\lib\security\cacerts" -file
"futureidec.crt"
```

```
[certificates stored path]> keytool -import -alias localhost -keystore
"C:\Program Files\Java\jre7\lib\security\cacerts" -file
"futureidec.crt"
```

If we have problems because alias “localhost” exists, we can delete it as follows:

```
keytool -delete -keystore "C:\Program
Files\Java\jdk1.7.0_55\jre\lib\security\cacerts" -alias localhost
```

```
keytool -delete -keystore "C:\Program
Files\Java\jre7\lib\security\cacerts" -alias localhost
```

And to know if the certificate was imported into cacerts:

```
C:\Program Files\Java\jdk1.7.0_55\jre\lib\security\> keytool -list -v -
keystore cacerts -alias localhost
```

```
C:\Program Files\Java\jre7\lib\security\> keytool -list -v -keystore
cacerts -alias localhost
```

3. Mutual authentication based in Challenge response

Session Library and Custom Web services developed for Session Library uses the algorithm Mutual Authentication based in Challenge response to ensure that the communication between them are trustworthy.

The steps that has been implemented:

- The client generates a 128bits random number called CC (client challenge).
- The client sends CC to the server (using a web service).
- The server generates a 128bits random number called SC (server challenge)
- The server calculates CR=AESEncryption128(SC+CC, K). CR is client response. That means that is encrypting using 128 bits AES algorithm de 128 bits, the SC and CC together (256 bits data), the key used to encrypt is K.
- The server answers with CR and SC
- The client takes CR, decrypts it (SC+CC = decryptAES128(CR, K) and tests if CC is the same as the original and if SC is the same as the received one. If the tests are ok, the client can trust the server.
- The client calculates SR=encryptionAES128(CC+SC,K) SR means Server response,
- The client call the web service with the session parameters and includes SR
- The server takes SR, it decrypts (CC+SC = decryptAES128(SR, K) and tests if the SC is the same as the original one and if CC is the same as the one received at the beginning. If these tests are fulfilled, the server can trust the client and it can do the necessary operations for the session parameters.

Annex C: Logging configuration for WS for the Session Library

The Web Services developed for the Session Library, are running in JBoss Application Server 7. In order to log the requests and responses to the Web Services, JBoss uses log4j and the configuration has been made to custom these logs.

The configuration file `$JBOSS_PATH\standalone\configuration\standalone.xml` contains the handlers and the loggers that must be configured

The handlers define the minimum tracing level that could be shown and the loggers define the tracing level that will finally show. That is, if the handler fixes level "INFO" and the logger defines "DEBUG" the logs will only show the INFO traces.

For production environment the level should be “INFO” and for development “DEBUG”. The following lines show the part of the configuration file for the handlers for development environment:

```
        <subsystem xmlns="urn:jboss:domain:logging:1.1">
            <console-handler name="CONSOLE">
                <level name="DEBUG"/>
                <formatter>
                    <pattern-formatter pattern="%d{HH:mm:ss,SSS} %-5p [%c] (%t)
%s%E%n"/>
                </formatter>
            </console-handler>
            <periodic-rotating-file-handler name="FILE">
                <formatter>
                    <pattern-formatter pattern="%d{HH:mm:ss,SSS} %-5p [%c] (%t)
%s%E%n"/>
                </formatter>
                <file relative-to="jboss.server.log.dir" path="server.log"/>
                <suffix value=".yyyy-MM-dd"/>
                <append value="true"/>
            </periodic-rotating-file-handler>
```

A new logger for FutureID should be included as follows (development environment)

```
<logger category="eu.futureid.atos" use-parent-handlers="false">
    <level name="DEBUG"/>
    <handlers>
        <handler name="CONSOLE"/>
        <handler name="FILE"/>
    </handlers>
</logger>
```

Annex D: Session Library installation guide

Annex B and Annex C explained important information about the Session Library and showed several steps for its installation and configuration.

In this annex a detailed set of steps to install it will be described.

Step 1: Files to install

- SessionLibrary.jar: This file will be used by the application, which will call the methods for creating session deleting session, setting attributes and getting the attributes
- SLSessioWS.war: This file must be deployed in JBoss. It contains the Web Services that performs the storing of session parameters.

Step 2: Deploying the SLSessionWS.war

SLSessionWS has been developed, tested and deployed using jboss-as-7.1.0. and JDK 1.7.0.55.

- Generate a public certificate as explained in 2.1.1
- Configure JBoss as explained in 2.1.2 (Annex B)
- Import the certificate as explained in 2.1.3 (Annex B)
- Configure the logs as explained in Annex C. Note that this is the case of develop environment (using DEBUG level) but for production environments “INFO” level should be used.

After having doing the above steps, you should have the file `$JBOSS_PATH/standalone/configuration/standalone.xml` similar to the file below.

And then JBoss can be started using standalone.bat

And after that SLSessionWS.war can be deployed.

FILE: `$JBOSS_PATH/standalone/configuration/standalone.xml`

```
<?xml version='1.0' encoding='UTF-8'?>


<server xmlns="urn:jboss:domain:1.1">


    <extensions>
```

```
        <extension module="org.jboss.as.clustering.infinispan"/>
        <extension module="org.jboss.as.configadmin"/>
        <extension module="org.jboss.as.connector"/>
        <extension module="org.jboss.as.deployment-scanner"/>
        <extension module="org.jboss.as.ee"/>
        <extension module="org.jboss.as.ejb3"/>
        <extension module="org.jboss.as.jdr"/>
        <extension module="org.jboss.as.jmx"/>
        <extension module="org.jboss.as.jpa"/>
        <extension module="org.jboss.as.logging"/>
        <extension module="org.jboss.as.mail"/>
        <extension module="org.jboss.as.naming"/>
        <extension module="org.jboss.as.osgi"/>
        <extension module="org.jboss.as.pojo"/>
        <extension module="org.jboss.as.remoting"/>
        <extension module="org.jboss.as.sar"/>
        <extension module="org.jboss.as.security"/>
        <extension module="org.jboss.as.threads"/>
        <extension module="org.jboss.as.transactions"/>
        <extension module="org.jboss.as.web"/>
        <extension module="org.jboss.as.webservices"/>
        <extension module="org.jboss.as.weld"/>
    </extensions>


    <management>
        <security-realms>
            <security-realm name="ManagementRealm">
                <authentication>
```

```
                    <properties path="mgmt-users.properties" relative-
to="jboss.server.config.dir"/>

                </authentication>

            </security-realm>

            <security-realm name="ApplicationRealm">

                <authentication>

                    <properties path="application-users.properties" relative-
to="jboss.server.config.dir"/>

                </authentication>

            </security-realm>

        </security-realms>

        <management-interfaces>

            <native-interface security-realm="ManagementRealm">

                <socket-binding native="management-native"/>

            </native-interface>

            <http-interface security-realm="ManagementRealm">

                <socket-binding http="management-http"/>

            </http-interface>

        </management-interfaces>

    </management>


    <profile>

        <subsystem xmlns="urn:jboss:domain:logging:1.1">

            <console-handler name="CONSOLE">

                <level name="DEBUG"/>

                <formatter>

                    <pattern-formatter pattern="%d{HH:mm:ss,SSS} %-5p [%c] (%t)
%s%E%n"/>

                </formatter>
```

```
            </console-handler>
            <periodic-rotating-file-handler name="FILE">
                <formatter>
                    <pattern-formatter pattern="%d{HH:mm:ss,SSS} %-5p [%c] (%t)
%s%E%n"/>
                </formatter>
                <file relative-to="jboss.server.log.dir" path="server.log"/>
                <suffix value=".yyyy-MM-dd"/>
                <append value="true"/>
            </periodic-rotating-file-handler>
            <logger category="com.arjuna">
                <level name="WARN"/>
            </logger>
            <logger category="org.apache.tomcat.util.modeler">
                <level name="WARN"/>
            </logger>
            <logger category="sun.rmi">
                <level name="WARN"/>
            </logger>
            <logger category="jacorb">
                <level name="WARN"/>
            </logger>
            <logger category="jacorb.config">
                <level name="ERROR"/>
            </logger>
            <logger category="eu.futureid.atos" use-parent-handlers="false">
                <level name="DEBUG"/>
                <handlers>
```

```
                    <handler name="CONSOLE"/>
                    <handler name="FILE"/>
                </handlers>
            </logger>
            <root-logger>
                <level name="INFO"/>
                <handlers>
                    <handler name="CONSOLE"/>
                    <handler name="FILE"/>
                </handlers>
            </root-logger>
        </subsystem>
        <subsystem xmlns="urn:jboss:domain:configadmin:1.0"/>
        <subsystem xmlns="urn:jboss:domain:datasources:1.0">
            <datasources>
                <datasource jndi-name="java:jboss/datasources/ExampleDS" pool-
name="ExampleDS" enabled="true" use-java-context="true">
                    <connection-url>jdbc:h2:mem:test;DB_CLOSE_DELAY=-1</connection-url>
                    <driver>h2</driver>
                    <security>
                        <user-name>sa</user-name>
                        <password>sa</password>
                    </security>
                </datasource>
                <drivers>
                    <driver name="h2" module="com.h2database.h2">
                        <xa-datasource-class>org.h2.jdbcx.JdbcDataSource</xa-
datasource-class>
```

```
                    </driver>

                </drivers>

            </datasources>

        </subsystem>

        <subsystem xmlns="urn:jboss:domain:deployment-scanner:1.1">

            <deployment-scanner path="deployments" relative-to="jboss.server.base.dir"
scan-interval="5000"/>

        </subsystem>

        <subsystem xmlns="urn:jboss:domain:ee:1.0"/>

        <subsystem xmlns="urn:jboss:domain:ejb3:1.2">

            <session-bean>

                <stateless>

                    <bean-instance-pool-ref pool-name="slsb-strict-max-pool"/>

                </stateless>

                <stateful default-access-timeout="5000" cache-ref="simple"/>

                <singleton default-access-timeout="5000"/>

            </session-bean>

            <mdb>

                <resource-adapter-ref resource-adapter-name="hornetq-ra"/>

                <bean-instance-pool-ref pool-name="mdb-strict-max-pool"/>

            </mdb>

            <pools>

                <bean-instance-pools>

                    <strict-max-pool name="slsb-strict-max-pool" max-pool-size="20"
instance-acquisition-timeout="5" instance-acquisition-timeout-unit="MINUTES"/>

                    <strict-max-pool name="mdb-strict-max-pool" max-pool-size="20"
instance-acquisition-timeout="5" instance-acquisition-timeout-unit="MINUTES"/>

                </bean-instance-pools>

            </pools>
```

```
            <caches>

                <cache name="simple" aliases="NoPassivationCache"/>

                <cache name="passivating" passivation-store-ref="file"
aliases="SimpleStatefulCache"/>

            </caches>

            <passivation-stores>

                <file-passivation-store name="file"/>

            </passivation-stores>

            <async thread-pool-name="default"/>

            <timer-service thread-pool-name="default">

                <data-store path="timer-service-data" relative-
to="jboss.server.data.dir"/>

            </timer-service>

            <remote connector-ref="remoting-connector" thread-pool-name="default"/>

            <thread-pools>

                <thread-pool name="default">

                    <max-threads count="10"/>

                    <keepalive-time time="100" unit="milliseconds"/>

                </thread-pool>

            </thread-pools>

        </subsystem>

        <subsystem xmlns="urn:jboss:domain:infinispan:1.1" default-cache-
container="hibernate">

            <cache-container name="hibernate" default-cache="local-query">

                <local-cache name="entity">

                    <transaction mode="NON_XA"/>

                    <eviction strategy="LRU" max-entries="10000"/>

                    <expiration max-idle="100000"/>

                </local-cache>
```

```
                <local-cache name="local-query">
                    <transaction mode="NONE"/>
                    <eviction strategy="LRU" max-entries="10000"/>
                    <expiration max-idle="100000"/>
                </local-cache>
                <local-cache name="timestamps">
                    <transaction mode="NONE"/>
                    <eviction strategy="NONE"/>
                </local-cache>
            </cache-container>
        </subsystem>
        <subsystem xmlns="urn:jboss:domain:jca:1.1">
            <archive-validation enabled="true" fail-on-error="true" fail-on-
warn="false"/>
            <bean-validation enabled="false"/>
            <default-workmanager>
                <short-running-threads>
                    <core-threads count="50"/>
                    <queue-length count="50"/>
                    <max-threads count="50"/>
                    <keepalive-time time="10" unit="seconds"/>
                </short-running-threads>
                <long-running-threads>
                    <core-threads count="50"/>
                    <queue-length count="50"/>
                    <max-threads count="50"/>
                    <keepalive-time time="10" unit="seconds"/>
                </long-running-threads>
```

```
        </default-workmanager>
    </subsystem>
    <subsystem xmlns="urn:jboss:domain:jdr:1.0"/>
    <subsystem xmlns="urn:jboss:domain:jmx:1.1">
        <show-model value="true"/>
        <remoting-connector/>
    </subsystem>
    <subsystem xmlns="urn:jboss:domain:jpa:1.0">
        <jpa default-datasource=""/>
    </subsystem>
    <subsystem xmlns="urn:jboss:domain:mail:1.0">
        <mail-session jndi-name="java:jboss/mail/Default">
            <smtp-server outbound-socket-binding-ref="mail-smtp"/>
        </mail-session>
    </subsystem>
    <subsystem xmlns="urn:jboss:domain:naming:1.1"/>
    <subsystem xmlns="urn:jboss:domain:osgi:1.2" activation="lazy">
        <properties>
            <property name="org.osgi.framework.startlevel.beginning">
                1
            </property>
        </properties>
        <capabilities>
            <capability name="javax.servlet.api:v25"/>
            <capability name="javax.transaction.api"/>
            <capability name="org.apache.felix.log" startlevel="1"/>
            <capability name="org.jboss.osgi.logging" startlevel="1"/>
            <capability name="org.apache.felix.configadmin" startlevel="1"/>
```

```
                <capability name="org.jboss.as.osgi.configadmin" startlevel="1"/>

            </capabilities>

        </subsystem>

        <subsystem xmlns="urn:jboss:domain:pojo:1.0"/>

        <subsystem xmlns="urn:jboss:domain:remoting:1.1">

            <connector name="remoting-connector" socket-binding="remoting" security-
realm="ApplicationRealm"/>

        </subsystem>

        <subsystem xmlns="urn:jboss:domain:resource-adapters:1.0"/>

        <subsystem xmlns="urn:jboss:domain:sar:1.0"/>

        <subsystem xmlns="urn:jboss:domain:security:1.1">

            <security-domains>

                <security-domain name="other" cache-type="default">

                    <authentication>

                        <login-module code="Remoting" flag="optional">

                            <module-option name="password-stacking"
value="useFirstPass"/>

                        </login-module>

                        <login-module code="RealmUsersRoles" flag="required">

                            <module-option name="usersProperties"
value="${jboss.server.config.dir}/application-users.properties"/>

                            <module-option name="rolesProperties"
value="${jboss.server.config.dir}/application-roles.properties"/>

                            <module-option name="realm" value="ApplicationRealm"/>

                            <module-option name="password-stacking"
value="useFirstPass"/>

                        </login-module>

                    </authentication>

                </security-domain>

                <security-domain name="jboss-web-policy" cache-type="default">
```

```
                    <authorization>
                        <policy-module code="Delegating" flag="required"/>
                    </authorization>
                </security-domain>
                <security-domain name="jboss-ejb-policy" cache-type="default">
                    <authorization>
                        <policy-module code="Delegating" flag="required"/>
                    </authorization>
                </security-domain>
            </security-domains>
        </subsystem>
        <subsystem xmlns="urn:jboss:domain:threads:1.1"/>
        <subsystem xmlns="urn:jboss:domain:transactions:1.1">
            <core-environment>
                <process-id>
                    <uuid/>
                </process-id>
            </core-environment>
            <recovery-environment socket-binding="txn-recovery-environment" status-
socket-binding="txn-status-manager"/>
            <coordinator-environment default-timeout="300"/>
        </subsystem>
        <subsystem xmlns="urn:jboss:domain:web:1.1" native="false" default-virtual-
server="default-host">
            <connector name="http" protocol="HTTP/1.1" scheme="http" socket-
binding="http"/>
            <connector name="https" protocol="HTTP/1.1" scheme="https" socket-
binding="https" enable-lookups="false" secure="true">
                <ssl name="foo-ssl" key-alias="foo" password="20141201" certificate-
key-file="../standalone/configuration/foo.keystore" protocol="TLSv1"/>
```

```
            </connector>

            <virtual-server name="default-host" enable-welcome-root="true">

                <alias name="localhost"/>

                <alias name="example.com"/>

            </virtual-server>

        </subsystem>

        <subsystem xmlns="urn:jboss:domain:webservices:1.1">

            <modify-wsdl-address>true</modify-wsdl-address>

            <wsdl-host>${jboss.bind.address:127.0.0.1}</wsdl-host>

            <endpoint-config name="Standard-Endpoint-Config"/>

            <endpoint-config name="Recording-Endpoint-Config">

                <pre-handler-chain name="recording-handlers" protocol-
bindings="##SOAP11_HTTP ##SOAP11_HTTP_MTOM ##SOAP12_HTTP ##SOAP12_HTTP_MTOM">

                    <handler name="RecordingHandler"
class="org.jboss.ws.common.invocation.RecordingServerHandler"/>

                </pre-handler-chain>

            </endpoint-config>

        </subsystem>

        <subsystem xmlns="urn:jboss:domain:weld:1.0"/>

    </profile>


    <interfaces>

        <interface name="management">

            <inet-address value="${jboss.bind.address.management:127.0.0.1}"/>

        </interface>

        <interface name="public">

            <inet-address value="${jboss.bind.address:127.0.0.1}"/>

        </interface>
```

```
    </interfaces>


    <socket-binding-group name="standard-sockets" default-interface="public" port-
offset="${jboss.socket.binding.port-offset:0}">

        <socket-binding name="http" port="8080"/>

        <socket-binding name="https" port="8443"/>

        <socket-binding name="management-native" interface="management"
port="${jboss.management.native.port:9999}"/>

        <socket-binding name="management-http" interface="management"
port="${jboss.management.http.port:9990}"/>

        <socket-binding name="management-https" interface="management"
port="${jboss.management.https.port:9443}"/>

        <socket-binding name="osgi-http" interface="management" port="8090"/>

        <socket-binding name="remoting" port="4447"/>

        <socket-binding name="txn-recovery-environment" port="4712"/>

        <socket-binding name="txn-status-manager" port="4713"/>

        <outbound-socket-binding name="mail-smtp">

            <remote-destination host="localhost" port="25"/>

     </outbound-socket-binding>

    </socket-binding-group>

    <deployments>

        <deployment name="ProofOfConcept.war" runtime-name="ProofOfConcept.war">

            <content sha1="76dd2b592ca25eecf6b39bbb730a3e025939f1d2"/>

        </deployment>

        <deployment name="ProofOfConcept2.war" runtime-name="ProofOfConcept2.war">

            <content sha1="66ebab91654265a2612c91aeec4eb51b6d6695d8"/>

        </deployment>

        <deployment name="SLSessionWS.war" runtime-name="SLSessionWS.war">

            <content sha1="a6e5cff8622fea6640241824ddc971b0ed4e9ab1"/>
```

```
        </deployment>
    </deployments>
</server>
```

Annex E: Installation Guide for SCT

SCT is intend to run on the same environment like the SL and AF components, therefore a JBoss AS 7.1.0 Application Server is the technical runtime platform.

The Simple Credential Transformer has been developed, tested and deployed using jboss-as-7.1.0. and JDK 1.7.0.67. The implementation is provided as a war-file (Web Application Archive) which supports an easy deployment and usage of the application.

Installation steps needed for running the SCT:

1. Installation of Java and JBoss
2. Configuration of JBoss Security and Logging via standalone.xml
3. Possibly adjust application configuration via web.xml
4. Possibly adjust SCT properties via sct.properties
5. Deploy SCT via JBoss Administration Console

1. Installation of Java and JBoss

Download and install Java SE 7 in accordance to the instructions given by Oracle. No specific configurations are needed.

Download JBoss AS 7.1 and unzip the downloaded archive file in a folder of your choice.

2. Configuration of JBoss Security and Logging

Configuration of JBoss is done via standalone.xml (<unzip-folder>\jboss-as-7.1.0.Final\standalone\configuration\standalone.xml). It is the central configuration file for a standalone JBoss instance.

For configure usage of HTTPS connections and an automatic redirect, adjust the configuration file as follows (Line 266ff):

```
<subsystem xmlns="urn:jboss:domain:web:1.1" native="false" default-virtual-server="default-host">
        <connector name="http" protocol="HTTP/1.1" scheme="http" socket-binding="http" redirect-port="8443"/>
        <connector name="https" protocol="HTTP/1.1" scheme="https" socket-binding="https" enable-
lookups="false" secure="true">
                <ssl name="foo-ssl" key-alias="foo" password="20141201" certificate-key-
        file="../standalone/configuration/foo.keystore" protocol="TLSv1"/>
        </connector>
        <virtual-server name="default-host" enable-welcome-root="true">
                <alias name="localhost"/>
```

```
        </virtual-server>
</subsystem>
```

For configure a logger for the SCT component with a particular configuration (log-level, handler) add and adjust the following lines to the configuration file:

```
<logger category="eu.futureid.aisjbos" use-parent-handlers="false">
        <level name="DEBUG"/>
        <handlers>
                <handler name="CONSOLE"/>
                <handler name="FILE"/>
        </handlers>
</logger>
```

3. Adjust application configuration via web.xml

If necessary you can adjust the application configuration via web.xml. For example you can set an alternative URL-pattern for the servlets or change the name of the application itself. You can find the web.xml file in the WEB-INF folder of the war-file.

The following listing presents the current definition of the SCT web application via web.xml file:

```
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns="http://java.sun.com/xml/ns/javaee"
        xmlns:web="http://java.sun.com/xml/ns/javaee"
        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
        http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="SCT_0.1" version="2.5">
        <display-name>Simple Credential Transformer</display-name>
        <display-name>sct</display-name>
        <welcome-file-list>
                <welcome-file>index.html</welcome-file>
        </welcome-file-list>
        <servlet>
                <description>Initialize Simple Credential Transformer and receive SAML Auth
        Response</description>
                <display-name>InitSCTServlet</display-name>
                <servlet-name>InitSCTServlet</servlet-name>
                <servlet-class>eu.futureid.aisjbos.sct.servlet.InitSCTServlet</servlet-class>
        </servlet>
        <servlet-mapping>
                <servlet-name>InitSCTServlet</servlet-name>
                <url-pattern>/InitSCT</url-pattern>
        </servlet-mapping>
        <servlet>
                <description></description>
                <display-name>RedirectServlet</display-name>
                <servlet-name>RedirectServlet</servlet-name>
                <servlet-class>eu.futureid.aisjbos.sct.servlet.RedirectServlet</servlet-class>
        </servlet>
        <servlet-mapping>
                <servlet-name>RedirectServlet</servlet-name>
                <url-pattern>/Redirect</url-pattern>
        </servlet-mapping>
```

```
</web-app>
```

4. Adjust sct.properties file

Within this file you can configure internal processing parameters for the SCT. The file is located in the WEB-INF\classes folder of the war-file. Purpose and usage of the parameters are described in 8.3.2.

5. Deploy SCT via JBoss Administration Console

Start JBoss in standalone configuration using bin/standalone.bat. Open https://localhost:8443 to get JBoss welcome page. Open Administrator Console, maybe you need to create a dedicated management user for the JBoss first. In the Management Console, go to Manage Deployments and add the SCT war-file by pressing Add Content button.