EUROPEAN MIDDLEWARE INITIATIVE

EUROPEAN MIDDLEWARE INITIATIVE

SECURITY TOKEN SERVICE DESIGN

Document version:	**Xx.Yy.Zz**
EMI Component Version:	**Aa.Bb**
Date:	**October 13, 2011**

This work is co-funded by the EC EMI project under the FP7 Collaborative Projects Grant Agreement Nr. INFSO-RI-261611.

CONTENTS

ABBREVIATIONS

CMP	Certificate Management Protocol
RST	Request Security Token
RSTR	Request Security Token Response
SAML	Security Assertion Markup Language
Shib3	Shibboleth Identity Provider version 3
STS	Security Token Service
WS	Web Service

REFERENCES

[1] K. Lawrence, C. Kaler, A. Nadalin, M. Goodner, M. Gudgin, A. Barbir, H. Granqvist, WS-Trust 1.3, OASIS Standard, http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.html, 2007.

[2] C. La Joie, WS-Trust 1.3 Interoperability Profile (Including Token-Specific Profiles), Working Draft 01, http://www.switch.ch/grid/support/documents, 2008.

[3] Shibboleth Consortium, The Shibboleth Documentation, http://wiki.shibboleth.net

[4] Spring Framework, Spring Web Flow 2, http://www.springsource.org/webflow

[5] C. Adams, S. Farrell, T. Kause, T. Mononen, Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP), RFC 4210, http://www.ietf.org/rfc/rfc4210.txt, 2005.

[6] The Legion of the Bouncy Castle, Java cryptography APIs, http://www.bouncycastle.org

1 INTRODUCTION

1.1 PURPOSE

This document provides the full design for Security Token Service (STS): a partial implementation of the service described in the OASIS WS-Trust specification [1]. At a high level this service is designed to allow its users to transform their existing security tokens from one format into another format. Examples of security tokens include X.509 certificate, SAML assertion and Kerberos ticket. Additionally the STS may renew, validate, or cancel the security tokens for which such functionality is appropriate.

Our STS implementation will be compatible with the WS-Trust Interoperability profile [2], supporting the ISSUE operation for X.509 certificate, SAML assertion and X.509 proxy certificate tokens. In addition to these security token types, also Kerberos ticket will be supported as an incoming security token format for the user authentication.

1.2 DOCUMENT ORGANIZATION

Section 2 provides a narrative description of the components of this service.

Section 3 provides a detailed design of the WS-Trust Profile Handler component.

Section 4 provides a detailed design of the Token Authority component.

Section 5 provides a detailed design of the client toolkit for this service.

Finally a set of appendices follows.

2 OVERALL ARCHITECTURE

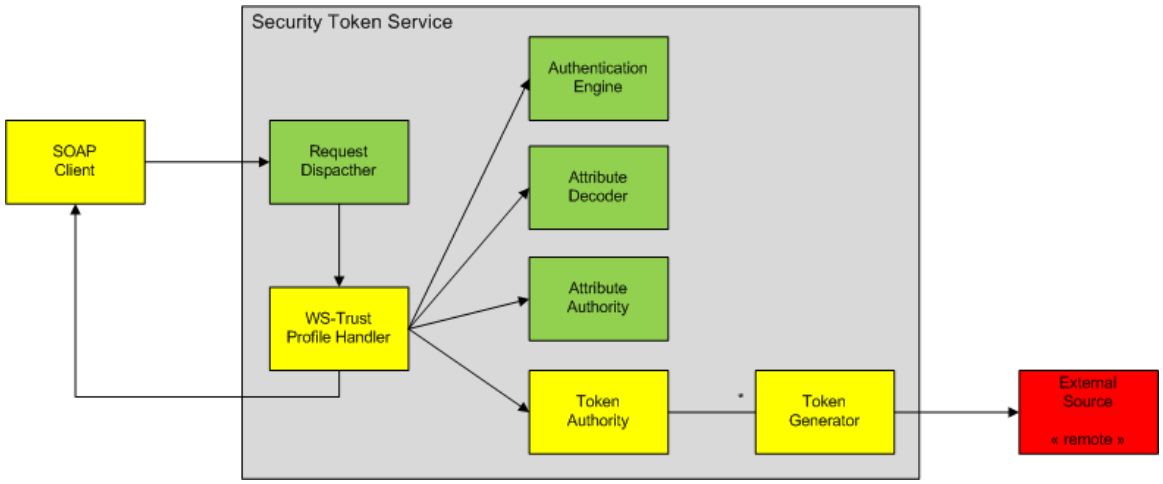

Figure 1: STS architectural overview

Our STS implementation is built on top of the Shibboleth Identity Provider (IdP) v3 (Shib3) and its dependencies, including OpenSAML3 [3]. It is entirely implemented in Java.

Figure 1 describes the overall architecture of the STS implementation and the relationships between its components, clients and remote services. The components described in the green boxes are provided by

Shib3 off the shelf. The components in the yellow boxes are implemented in this project and the red box is a remote source possibly exploited in the security token issuance process.

In principle, the SOAP client can be any Web Service client capable of producing and communicating compatible RST messages, and understanding RTSR messages produced by the STS. The project provides a client toolkit library that facilitates integration with any clients implemented in Java.

The client interacts with the service endpoint provided by the *Request Dispacther*. The dispatcher forwards the appropriate requests to the *WS-Trust Profile Handler* which interacts with the following components of the system before finally returning the response message back to the client:

- *Authentication Engine*: authenticates the user from the incoming security token.
- *Attribute Decoder*: decodes the attributes from the incoming security token.
- *Attribute Authority*: resolves the required attributes for the outgoing security tokens.
- *Token Authority*: issues the requested security tokens by using appropriate *Token Generator*s, possibly connected to external sources.

The components implemented in this project are described in more detail in the next sections. In addition to them, plugins need to be implemented for *Authentication Engine* and *Attribute Decoder* to provide compatibility with our supported incoming security token formats.

3 WS-TRUST PROFILE HANDLER DESIGN

The profile handler is responsible for orchestrating the process for handling a profile request before finally generating the response.

3.1 PROFILE SEQUENCE

The profile sequence starts when the *Request Dispatcher* has forwarded a SOAP message to this profile handler. At first, the handler creates a *request context* that is exploited and/or updated in the following tasks:

- *Decode the request*: decode and optionally decrypt the RST message.
- *Validate the request*: signature, SSL/TLS, replay and timestamp validation, and policy conformancy verification.
- *Validate the claims*: incoming request may contain multiple claims that need to be extracted and validated.
- *Resolve attributes*: the resolution and filtering of the attributes for the requested security token.
- *Issuance of the security tokens*: use the collected information to issue the requested security tokens.
- *Create the response*: encode the requested security token into a RSTR message, possibly signed and/or encrypted.

3.2 SOFTWARE DESIGN

Similarly to the other profile handlers in Shib3, WS-Trust Profile Handler uses *ProfileRequestContext* as the base container of all the data needed in the profile sequence. Figure 2 illustrates its dependencies to the other components. As a *SubcontextContainer* it can also contain other, more specific request contexts, for instance for resolving attributes.

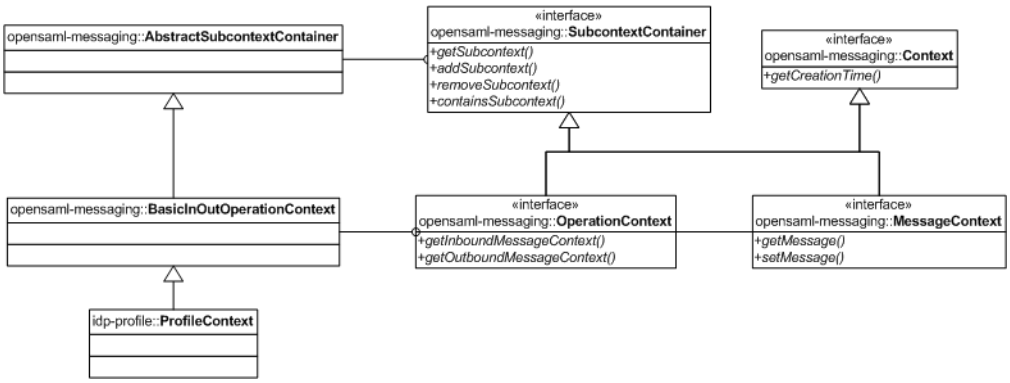

Figure 2: Class Dependencies for the *ProfileRequestContext*

The profile sequence is composed of a set of states, whose flow is defined with the Spring WebFlow [4]. In practise this means that the profile sequence is divided into multiple *actions* containing the logic of the flow. As the STS is accessed with a request-response protocol, the states cannot contain user interaction, i.e. view-states. Some of the actions may invoke a subflow, for instance for validating a specific type of security token.

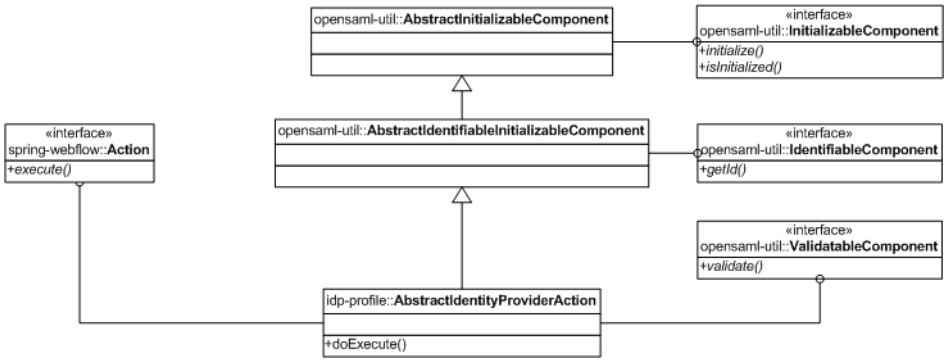

Figure 3: Class Dependencies for the *AbstractIdentityProviderAction*

Shib3 provides an abstract class for the actions (see Figure 3) that is extended in our action implementations. The detailed flow of the actions can be seen in Section 6.1 (TODO).

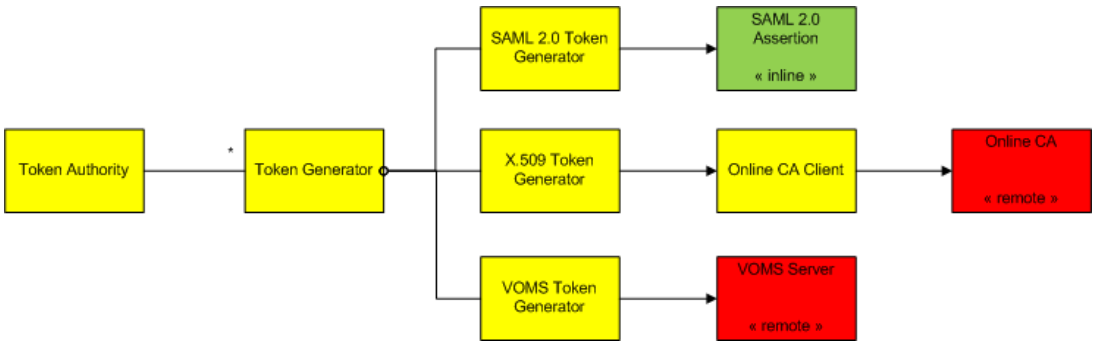

Figure 4: Token Authority Overview

4 TOKEN AUTHORITY DESIGN

4.1 OVERVIEW

Figure 4 illustrates the Token Authority design. The plugin mechanism allows support for different kinds of token generators. In the current EMI workplan, our STS project is supposed to support three outgoing security tokens: SAML assertion, X.509 certificate and X.509 proxy certificate.

4.2 SOFTWARE DESIGN

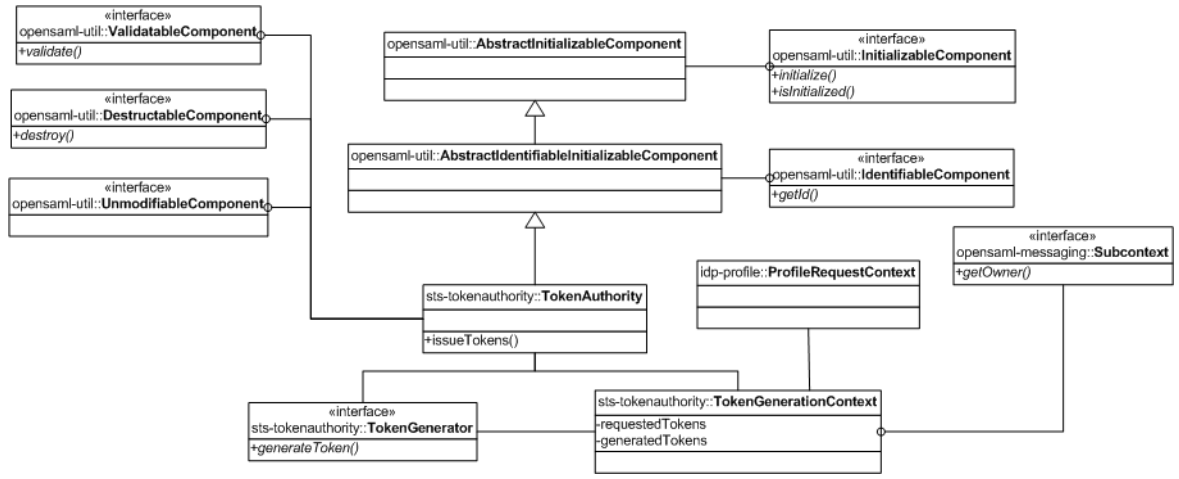

Figure 5: UML Class Diagram for the Token Authority

Figure 5 describes the class diagram for *Token Authority* and its dependencies. It is accessed by the method *issueTokens* that requires *TokenGenerationContext* as a parameter. Depending on the contents of the context, one or more security tokens will be generated using the appropriate token generators. The generated tokens can be accessed from the generation context.

4.3 TOKEN GENERATORS

4.3.1 SAML TOKEN GENERATOR

SAML Token Generator is implemented using the Spring WebFlow, by exploiting some of the same actions that are already implemented in Shib3 for the SAML 2.0 profile handlers.

The flow definition file is described in Section 6.2 (TODO).

4.3.2 X.509 TOKEN GENERATOR

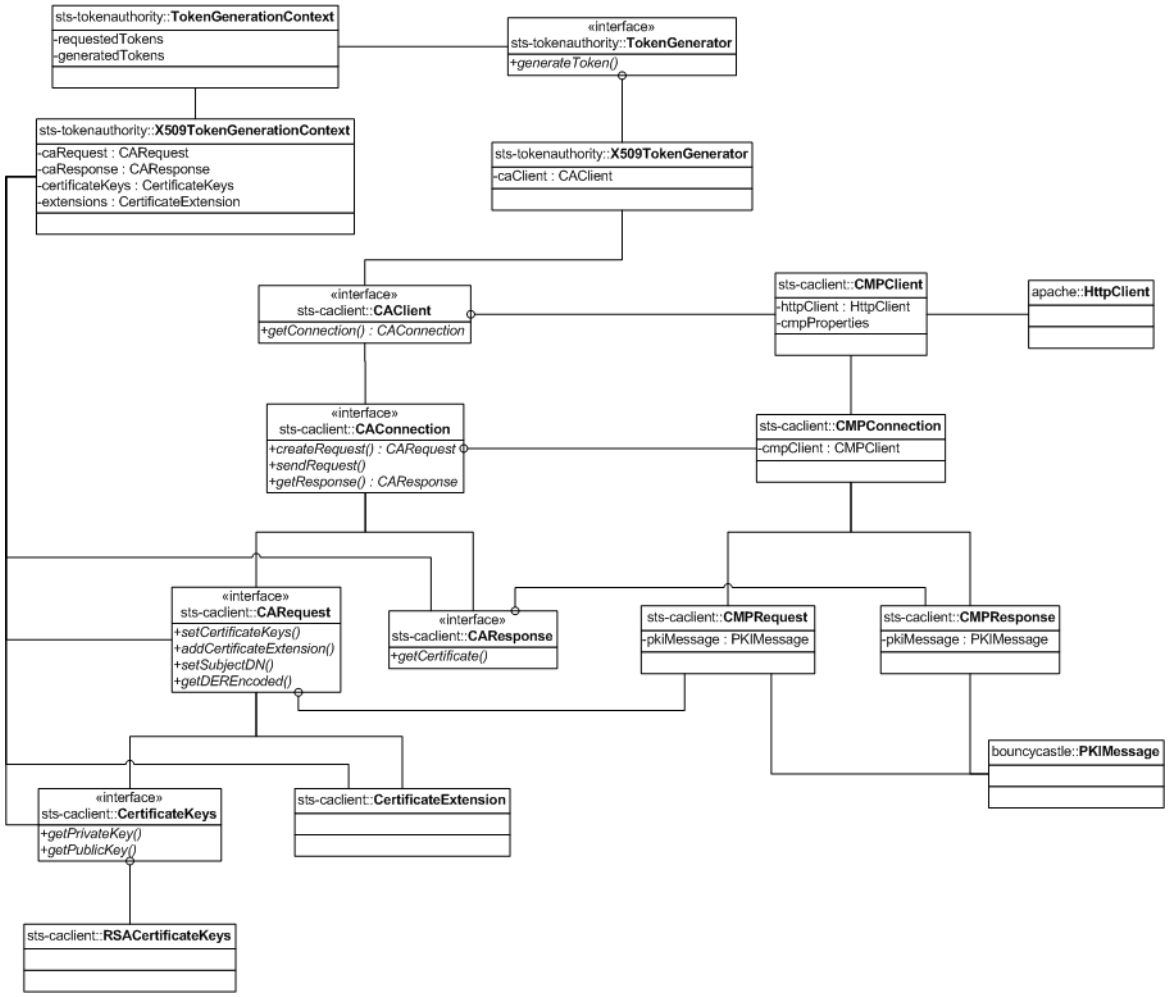

Figure 6: UML Class Diagram for the X.509 Token Generator

Figure 6 illustrates the design of the X.509 Token Generator. Initially, the Certificate Management Protocol (CMP) [5] will be supported for the communication with the online CA: the implementation will be based on the Bouncy Castle Java library [6].

Another protocols, like MyProxy, may be supported by implementing the corresponding interfaces: *CAClient*, *CAConnection*, *CARequest* and *CAResponse*.

4.3.3 X.509 PROXY TOKEN GENERATOR

TODO

5 CLIENT TOOLKIT

TODO

Toolkit can also be utilized for obtaining the desired token to be attached in the RST message, e.g. obtaining a SAML assertion token using ECP profile.

6 APPENDICES

6.1 WS-TRUST PROFILE HANDLER FLOW DEFINITION

```
<flow xmlns="http://www.springframework.org/schema/webflow"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <action-state id="InitializeProfileRequestContext">
    <transition on="proceed" to="DecodeRequestSecurityTokenMessage"/>
  </action-state>

  <action-state id="DecodeRequestSecurityTokenMessage">
    <transition on="proceed" to="ValidateRequestSecurityTokenMessage"/>
  </action-state>

  <action-state id="ValidateRequestSecurityTokenMessage">
    <transition on="proceed" to="ExtractClaims"/>
    <transition on="error" to="GenerateErrorResponse"/>
  </action-state>

  <action-state id="ExtractClaims">
    <transition on="proceed" to="ValidateClaims"/>
  </action-state>

  <!-- This will programatically invoke subflow(s) for validating the claims -->
  <action-state id="ValidateClaims">
    <transition on="proceed" to="DecodeAttributesFromClaims"/>
    <transition on="error" to="GenerateErrorResponse"/>
  </action-state>

  <action-state id="DecodeAttributesFromClaims">
    <transition on="proceed" to="AuthorizeUser"/>
  </action-state>

  <action-state id="AuthorizeUser">
    <transition on="proceed" to="ResolveAttributes"/>
    <transition on="error" to="GenerateErrorResponse"/>
  </action-state>
```

```
  <action-state id="ResolveAttributes">
    <transition on="proceed" to="FilterAttributes"/>
    <transition on="error" to="GenerateErrorResponse" />
  </action-state>

  <action-state id="FilterAttributes">
    <transition on="proceed" to="IssueSecurityTokens"/>
    <transition on="error" to="GenerateErrorResponse" />
  </action-state>

  <!-- This will programatically invoke subflow(s) for generating the security tokens -->
  <action-state id="IssueSecurityTokens">
    <transition on="proceed" to="GenerateResponse" />
    <transition on="error" to="GenerateErrorResponse" />
  </action-state>

  <action-state id="GenerateResponse">
    <transition on="proceed" to="AttachSecurityTokens"/>
  </action-state>

  <action-state id="AttachSecurityTokens">
    <transition on="proceed" to="EncryptResponse"/>
  </action-state>

  <action-state id="EncryptResponse">
    <transition on="proceed" to="SignResponse"/>
    <transition on="error" to="GenerateErrorResponse" />
  </action-state>

  <action-state id="SignResponse">
    <transition on="proceed" to="EncodeResponse"/>
    <transition on="error" to="GenerateErrorResponse" />
  </action-state>

  <action-state id="GenerateErrorResponse">
    <transition on="proceed" to="EncodeResponse"/>
  </action-state>

  <action-state id="EncodeResponse">
    <transition on="proceed" to="done"/>
  </action-state>

  <end-state id="done" />

</flow>
```

Listing 1: The Spring WebFlow definition for the WS-Trust Profile Handler

6.2 SAML TOKEN GENERATOR FLOW DEFINITION

```
<flow xmlns="http://www.springframework.org/schema/webflow"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
```

```
  <action-state id="GenerateAssertion">
    <transition on="proceed" to="AddRecipientToAssertion"/>
  </action-state>

  <action-state id="AddRecipientToAssertion">
    <transition on="proceed" to="AddAuthenticationStatementToAssertion"/>
  </action-state>

  <action-state id="AddAuthenticationStatementToAssertion">
    <transition on="proceed" to="AddAttributeStatementToAssertion"/>
  </action-state>

  <action-state id="AddAttributeStatementToAssertion">
    <transition on="proceed" to="AddSubjectToStatements"/>
  </action-state>

  <action-state id="AddSubjectToStatements">
    <transition on="proceed" to="AddNotBeforeConditionToAssertion"/>
  </action-state>

  <action-state id="AddNotBeforeConditionToAssertion">
    <transition on="proceed" to="AddNotOnOrAfterConditionToAssertion"/>
  </action-state>

  <action-state id="AddNotOnOrAfterConditionToAssertion">
    <transition on="proceed" to="AddAudienceRestrictionsToAssertion"/>
  </action-state>

  <action-state id="AddAudienceRestrictionsToAssertion">
    <transition on="proceed" to="SignAssertion"/>
  </action-state>

  <action-state id="SignAssertion">
    <transition on="proceed" to="done"/>
  </action-state>

  <action-state id="GenerateError">
    <transition on="proceed" to="done"/>
  </action-state>

  <global-transitions>
    <transition on="error" to="GenerateError"/>
  </global-transitions>

  <end-state id="done"/>

</flow>
```

Listing 2: The Spring WebFlow definition for the SAML Token Generator