



## Deliverable No. 5.2

### Security guidelines and initial version of security tools

Grant Agreement No.: 600841  
Deliverable No.: D5.2  
Deliverable Name: Security guidelines and initial version of security tools  
Contractual Submission Date: 30/09/2014  
Actual Submission Date: 30/09/2014

Dissemination Level		
PU	Public	
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	X





COVER AND CONTROL PAGE OF DOCUMENT	
Project Acronym:	CHIC
Project Full Name:	Computational Horizons In Cancer (CHIC): Developing Meta- and Hyper-Multiscale Models and Repositories for In Silico Oncology
Deliverable No.:	D5.2
Document name:	Security guidelines and initial version of security tools
Nature (R, P, D, O) <sup>1</sup>	P
Dissemination Level (PU, PP, RE, CO) <sup>2</sup>	CO
Version:	1.0
Actual Submission Date:	30/09/2014
Editor: Institution: E-Mail:	Elias Neri Custodix elias.neri@custodix.com

#### ABSTRACT:

This document describes the CHIC Security Framework, security interoperability guidelines and integration tutorials. The CHIC security frameworks is made up by lightweight re-usable components dealing with authentication, authorisation and auditing based on widely used industry standards such as SAML, WS-\*, XACML and XDas.

#### KEYWORD LIST:

Security, Authentication, Authorisation, Audit, Auditing, SAML, WS-Security, WS-Federation, WS-Trust, XDas

*The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 600841.*

*The author is solely responsible for its content, it does not represent the opinion of the European Community and the Community is not responsible for any use that might be made of data appearing therein.*

<sup>1</sup> R=Report, P=Prototype, D=Demonstrator, O=Other

<sup>2</sup> PU=Public, PP=Restricted to other programme participants (including the Commission Services), RE=Restricted to a group specified by the consortium (including the Commission Services), CO=Confidential, only for members of the consortium (including the Commission Services)

<b>MODIFICATION CONTROL</b>				
<b>Version</b>	<b>Date</b>	<b>Status</b>	<b>Author</b>	<b>Modification Control</b>
0.1	08/08/2014	Draft	Elias Neri, Custodix	
0.2	10/09/2014	Draft	Elias Neri, Custodix	
0.3	25/09/2014	Draft	Elias Neri, Custodix	
0.4	29/09/2014	Internal Review	Debora Testi, CINECA	
1.0	30/09/2014	Final	Elias Neri, Custodix	
1.1	13/10/2014	Draft	Elias Neri, Custodix	Minor typo's
2.0	23/04/2015	Final	Elias Neri, Custodix	Updated Security Vocabulary Updated Integration Tutorials Added Audit Json Schema

#### List of contributors

- Dawn Walker, University of Sheffield
- Debora Testi, CINECA
- Elias Neri, Custodix
- Wouter Dhaeze, Custodix

## Contents

1	EXECUTIVE SUMMARY.....	6
2	INTRODUCTION .....	7
3	SECURITY FRAMEWORK .....	8
3.1	BROKERED AUTHENTICATION .....	8
3.2	AUTHORISATION.....	16
3.3	AUDITING .....	16
3.4	SECURITY VOCABULARY .....	22
4	SECURITY FRAMEWORK INTEROPERABILITY GUIDELINES.....	23
4.1	TRANSPORT LAYER SECURITY .....	23
4.2	MESSAGE PROTOCOLS AND MESSAGE LAYER SECURITY.....	24
4.3	AUTHENTICATION PROTOCOLS .....	30
5	SECURITY FRAMEWORK INTEGRATION TUTORIALS .....	32
5.1	PASSIVE (BROWSER BASED) AUTHENTICATION .....	32
5.2	ACTIVE AUTHENTICATION.....	51
5.3	INTEGRATION OF SECURITY FRAMEWORK WITH VPH-OP .....	73
5.4	AUDITING .....	75
6	REFERENCES .....	78
	APPENDIX 1 – AUDIT MESSAGE JSON SCHEMA .....	80

## 1 Executive Summary

This document describes the CHIC Security Framework, security interoperability guidelines and integration tutorials.

The CHIC security frameworks is made up by lightweight re-usable components dealing with authentication, authorisation and auditing based on widely used industry standards such as SAML, WS-\*, XACML and XDAS.

A user authenticates himself to CHIC web sites by retrieving an identity assertion (SAML token) from the CHIC Identity Provider (IdP). The IdP will request the user his authentication credentials (e.g. username/password) before a SAML token is issued. Web sites on the other hand integrate with the IdP by publishing SAML metadata that defines which SAML profiles and bindings are supported by the site. The web site is then responsible for consuming and verifying the tokens issued by the IdP.

Non browser clients (e.g. clients calling web services) can make use of the SAML ECP (Enhanced Client or Proxy) profile. To avoid authentication on each web request, the SAML ECP profile assumes session-based security to keep the security context open. As web services are typically stateless and a different security context can be required for each message, message level security is preferred. For SOAP web services CHIC implements WS-\* which handles message level security and brokered authentication. For REST web services CHIC will also make use of the WS-\* security components. More specifically a REST client will also call the WS-Trust Secure Token Service to request a SAML security token and pass it through the HTTP Authorization header to REST services.

A successful auditing system requires high quality information on events with potential hazardous security risks. The provision of this information is primarily the responsibility of the service provider, but well-coordinated cooperation between the developers of service providers, and audit engineers will be necessary to enrich the audit messages with enough data to provide clear and sufficient audit trails in a production environment. The CHIC security architecture provides, through the CHIC audit services, the necessary audit components to assist both developers and audit engineers in implementing high quality audit logs.

After the introduction (Chapter 2), Chapter 3 describes the security framework components, and Chapter 4 describes through interoperability guidelines the minimal requirements to which services should adhere to be allowed to integrate with the CHIC architecture.

The final chapter (Chapter 5) contains integration profiles for various applications and technologies on how they can be integrated into the security architecture.

## 2 Introduction

One of the aims of the CHIC project is to develop cutting edge ICT tools, services and infrastructure to foster the development of elaborate and reusable integrative models (hypermodels) in the field of cancer diagnosis and treatment. During the development of these tools both retrospective and prospective patient data will be used to develop, test and validate these models. The processing of this sensitive data does have legal and ethical requirements and consequences as described in Deliverable D4.1 (submitted at PM7 in September 2013). This resulted in Deliverable D.4.3.1 (submitted at PM12 in May 2014), which provides a concrete data privacy framework, including a security framework, a de-identification framework and a set of agreements for the CHIC partners to sign.

For this reason, access to all CHIC data, tools and services should be covered by the necessary level of security so to avoid un-authorised people to get access to private information.

This deliverable focuses on the pure security aspect: authentication, authorisation and auditing.

### 3 Security Framework

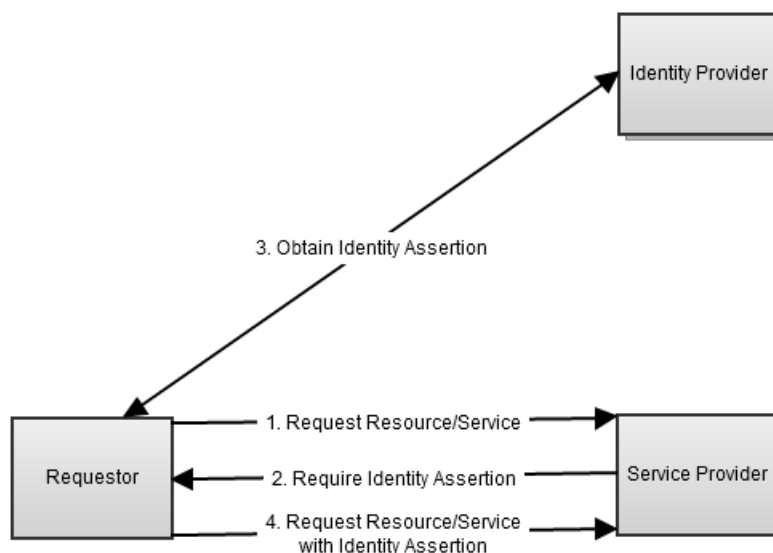
CHIC will not reinvent but instead base its security architecture on the security framework developed in past and ongoing EU projects such as ACGT, P-Medicine, EURECA, and INTEGRATE. This security architecture makes use of modular re-usable components dealing with e.g. brokered authentication, authorisation, auditing and de-identification. These components are based on widely used industry standards such as SAML, WS-\*, XACML.

Typical components amongst others within security architectures are:

- An Identity Provider (IdP) is responsible for authentication and provides identity assertion to other services called service providers.
- A service provider is an accessible service within an architecture that serves protected resources.
- A policy decision point is an entity that takes authorisation decisions.

#### 3.1 Brokered Authentication

Direct authentication of the user on each service (service provider) is not practical or advisable (e.g. authentication would have to be implemented on each service or a user would have to authenticate on each service separately). CHIC will therefore provide brokered authentication in which an authentication broker (such as an IdP) is responsible for authenticating the users and issuing identity tokens to all relevant services. The user can then use such identity tokens to access the project's services. To avoid that the user needs to provide his credentials each time he accesses a different service, the authentication broker can keep the authenticated session open. This results in new identity tokens being issued automatically (Single Sign-on) for each service the user accesses as long as the authentication broker's authenticated (SSO) session is still active.



**Figure 1 Brokered Authentication Flow**

Brokered authentication (Figure 1) is initiated when a user (the requestor) requests a protected resource from a service provider (1). The Service provider replies that he needs an identity assertion (2) to be able to verify whether the user is allowed to access the requested resource. The requestor authenticates on a supported identity provider and requests an identity assertion (3). The requestor



sends the assertion to the service provider as claim who he is and, if he gets access, receives the requested resource (4).

Enhanced clients (Non browser clients) such as desktops and server-side HTTP clients typically have the capability to directly contact identity providers without requiring a redirection from the service provider. Browsers on the other hand do not have this capability and will require a redirection from the service provider to the selected identity provider. This will have impact on the communication flow and thus different communication protocols are available depending on whether the client is a browser or not.

### 3.1.1 Identity Token Format

The format of the framework's assertion is defined by the SAML (Security Assertion Markup Language [1]) standard. SAML is an OASIS standard that defines signed identity assertions in an XML-based format. Such an identity assertion contains the user's authentication and authorisation information. Next to the format, SAML also defines the protocols for browsers and enhanced clients on how to exchange the assertions between identity and service providers. Next to SAML there are also other commonly used standards, such as WS-Federation, that define protocols on how to exchange a SAML assertion.

A SAML 2.0 assertion is structured in three parts:

1. Authentication statements which assert that the user did indeed authenticate with the Identity Provider at a given time using a particular method of authentication.
2. Attribute statements which assert that the user is associated with certain attributes.
3. Authorization decision statements which assert the a user is permitted to perform action on a resource with given evidence.

Example SAML Token

```
<saml2:Assertion ID="..." IssueInstant="..." Version="2.0" xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion">
  <saml2:Issuer>CHIC IdP</saml2:Issuer>
  <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    ...<!--XML Signature-->
  </ds:Signature>
  <saml2:Subject>
    <saml2:NameID>...<!--User's UUID--></saml2:NameID>
  </saml2:Subject>
  <saml2:Conditions NotBefore="..." NotOnOrAfter="...">
    <saml2:AudienceRestriction>
      <saml2:Audience>https://service-chic-test.custodix.com<!--Target service provider for which this token is valid--></saml2:Audience>
    </saml2:AudienceRestriction>
  </saml2:Conditions>
  <saml2:AttributeStatement>
    <saml2:Attribute FriendlyName="mail" Name="urn:oid:0.9.2342.19200300.100.1.3">
      <saml2:AttributeValue><!--User's email address-->y</saml2:AttributeValue>
    </saml2:Attribute>
    <saml2:Attribute FriendlyName="uuid" Name="urn:oid:0.9.2342.19200300.100.1.1">
      <saml2:AttributeValue><!--User's UUID--></saml2:AttributeValue>
    </saml2:Attribute>
    <saml2:Attribute FriendlyName="givenName" Name="urn:oid:2.5.4.42">
      <saml2:AttributeValue>Homer</saml2:AttributeValue>
    </saml2:Attribute>
    ...
  </saml2:AttributeStatement>
</saml2:Assertion>
```

The big advantages of SAML are:

1. Open commonly used & implemented standard
2. Exchange of identity information both on client-server as server-server communication.
3. Support for delegation by allowing the delegation chain to be inserted in the SAML token. A server can thus act through a SAML token in name of another user.

### 3.1.2 Brokered authentication for web services

#### 3.1.2.1 SOAP Web Services

The WS-\* specifications are the most commonly used standards to secure SOAP Web Services. WS-\* is a collective noun for a variety of specifications associated with web services. Together these specifications form the basic framework for web services build on the first-generation standards of SOAP and WSDL.

**WS-Security** [6] & [7] defines how web service messages can be exchanged in a secure way by guarding the integrity, confidentiality and the sender's identity of the messages. To enforce this, WS-Security uses XML signature (for integrity), XML encryption (for confidentiality) and various security token formats, such as SAML, Kerberos, X.509 (for sender authentication), to provide end-to-end security.

**WS-Trust** [11] & [12] is an extension of WS-Security providing methods for issuing, renewing and validating security tokens and providing ways to establish, assess the presence of, and broker trust relationships. Using the extensions defined in WS-Trust, applications can participate in secure communication designed to work within the web service framework. A main concept in WS-Trust is the Security Token Service (STS). This is a special web service that issues security tokens conforming to the WS-Security specification.

**WS-SecureConversation** [13] defines extensions that include session key derivation and security context establishment/sharing.

**WS-SecurePolicy** [16] defines XML based policies that are called security policy assertions. These policies allow web services to express their constraints and requirements. Policies can be used to drive development tools to generate code with certain capabilities, or may be used at runtime to negotiate the security aspects of web service communication. The intent is to provide enough information for compatibility and interoperability to be determined by web service participants.

**WS-Federation** [18] aims to simplify the development of federated services through cross-realm communication and management of federation services by re-using the WS-Trust Security Token Service model and protocol. WS-Federation defines two important profiles:

1. The Active Requestor Profile, which defines how enhanced clients can request identity tokens from an identity provider (called Secure Token Service or STS in WS-\*) and exchange it with a service provider.

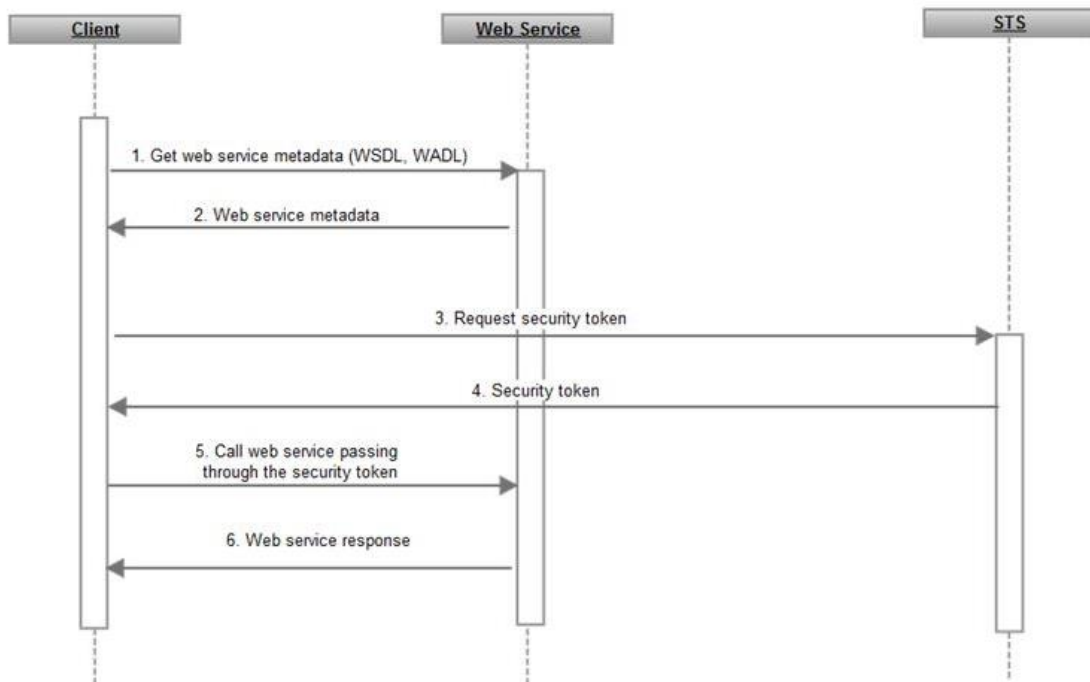


Figure 2 WS-Federation Active Requestor Profile

2. The Passive Requestor Profile, which defines a protocol for browsers to exchange identity token between identity and service providers. As the Passive Requestor Profile is usually not used for web service calls but rather for Web Site (browser) authentication, it is not further described in this chapter.

A SOAP client will thus request identity tokens and authenticate on a Secure Token Service (STS) as defined in WS-Trust. An STS is a SOAP Web Service with the following operations (non-exclusive):

1. **Issue:** this operation issues a new security token based on the credential provided or proven in the request. The operation accepts a request security token (RST) and returns a request security token response (RSTR).

```

<wst:RequestSecurityToken Context="..." xmlns:wst="...">
  <wst:TokenType>...</wst:TokenType>
  <wst:RequestType>...</wst:RequestType>
  <wst:SecondaryParameters>...</wst:SecondaryParameters> ... <wsp:AppliesTo>...</wsp:AppliesTo>
  <wst:Claims Dialect="...">...</wst:Claims>
  <wst:Entropy>
    <wst:BinarySecret>...</wst:BinarySecret>
  </wst:Entropy>
  <wst:Lifetime>
    <wsu:Created>...</wsu:Created>
    <wsu:Expires>...</wsu:Expires>
  </wst:Lifetime>
</wst:RequestSecurityToken>

```

```

<wst:RequestSecurityTokenResponse Context="..." xmlns:wst="...">
  <wst:TokenType>...</wst:TokenType>
  <wst:RequestedSecurityToken>...</wst:RequestedSecurityToken>
</wst:RequestSecurityTokenResponse>

```

2. **Cancel:** this operation cancels a token so that it cannot be used anymore when it is no longer needed. After cancellation the STS will not renew or validate the token anymore.

```
<wst:RequestSecurityToken>
  <wst:RequestType> http://docs.oasis-open.org/ws-sx/ws-trust/200512/Cancel </wst:RequestType>
  <wst:CancelTarget> ... </wst:CancelTarget>
</wst:RequestSecurityToken>
```

```
<wst:RequestSecurityTokenResponse>
  <wst:RequestedTokenCancelled/>
</wst:RequestSecurityTokenResponse>
```

3. **Renew:** a previously issued, possibly expired token, is presented and the same token is returned with new expiration semantics. The requestor must either prove authorised use of the token or be trusted by STS to issue third-party renewal requests.

```
<wst:RequestSecurityToken xmlns:wst="...">
  <wst:TokenType>...</wst:TokenType>
  <wst:RequestType> http://docs.oasis-open.org/ws-sx/ws-trust/200512/Renew </wst:RequestType>
  <wst:RenewTarget> ... reference to previously issued token ... </wst:RenewTarget>
  <wst:AllowPostdating/>
  <wst:Renewing Allow="..." OK="..."/>
</wst:RequestSecurityToken>
```

```
<wst:RequestSecurityTokenResponse xmlns:wst="...">
  <wst:TokenType> ... </wst:TokenType>
  <wst:RequestedSecurityToken>...</wst:RequestedSecurityToken>
  <wst:Lifetime>...</wst:Lifetime>
</wst:RequestSecurityTokenResponse>
```

4. **Validate:** this operation evaluates the specified token. The result can be a status, a new token or both.

```
<wst:RequestSecurityToken xmlns:wst="...">
  <wst:TokenType>...</wst:TokenType>
  <wst:RequestType>...</wst:RequestType>
  <wst:ValidateTarget>... </wst:ValidateTarget>
</wst:RequestSecurityToken>
```

```
<wst:RequestSecurityTokenResponse xmlns:wst="..." >
  <wst:TokenType>...</wst:TokenType>
  <wst:RequestedSecurityToken>...</wst:RequestedSecurityToken>
  <wst:Status>
    <wst:Code>...</wst:Code>
    <wst:Reason>...</wst:Reason>
  </wst:Status>
</wst:RequestSecurityTokenResponse>
```

WS-Trust does not specify or limit the format of the issued token. Within CHIC SAML 2.0 tokens will be issued by the STS. The WS-Security profile "SAML Token Profile 1.1." defines how a SAML token can be passed to SOAP web services through SOAP headers.

```
<!--SAML token issued by STS for "End User" targetted to "Service A"-->
<saml2:Assertion xmlns:saml2="..."
ID="uuid-ec80226b-0a2e-43bd-a98b-54b071407edd" IssueInstant="2012-05-09T09:43:00.292Z" Version="2.0">
  <saml2:Issuer>STS</saml2:Issuer>
  <saml2:Subject>
    <saml2:NameID>End User</saml2:NameID>
  </saml2:Subject>
  <saml2:Conditions NotBefore="..." NotOnOrAfter="...">
    <saml2:AudienceRestriction>
      <saml2:Audience>http://serviceA.com</saml2:Audience>
    </saml2:AudienceRestriction>
  </saml2:Conditions>
  <saml2:AttributeStatement>
    ...
  </saml2:AttributeStatement>
</saml2:Assertion>
```

### 3.1.2.2 REST Web Services

As REST Web Services are also web services, a natural solution to secure REST Web Services would be to look at the WS-\* specifications. A Secure Token Service (STS) can then be used by REST clients to request a security token.

Before calling the REST service a client sends a SOAP request containing an RST (RequestSecurityToken) to the STS. The STS then returns the identity assertion as a SAML token, embedded in a RSTR (RequestSecurityTokenResponse). In SOAP this SAML token is passed to the calling service through a SOAP header. This is not possible in REST though as there are no SOAP headers. In REST, the HTTP headers and request line are part of the message: an HTTP header is therefore the REST equivalent of a SOAP header. A SAML token can then be passed to a REST service through the HTTP authorization header. The authorization header value should be formatted as follows: "SAML auth=<Base 64 encoded compressed SAML token>". The SAML token is hereby compressed with the DEFLATE (zlib) [26] compression algorithm. This ensures that the token fits in the typical 4kb header size limit.

To secure a REST service the same SOAP STS service is used as with SOAP services as defined in WS-Trust. REST clients are responsible for retrieving a SAML token from the STS and then passing it through a HTTP authorization header to the REST service. The client somehow needs to find out what the REST service's security policies are (i.e. what STS to call). In WS-\* a web service publishes its security policies through WS-Policy annotations on the service's WSDL metadata. Such a WSDL file annotated with security policies (by using the WSDL HTTP binding) could actually also be used for REST services. As REST is resource based while SOAP and the WSDL metadata format are action based, this is not an exact match. There is a metadata format, called WADL, which is specifically designed to describe a REST service. No known specification that defines how to add security policies to a WADL metadata file, is available yet. Therefore research should be done on whether WS-Policy metadata annotations can be used to add security annotations to a WADL metadata file.

A drawback of using a SOAP STS is that clients are forced to use SOAP to request security tokens. Therefore the secure token service could be better integrated into REST services by REST-ifying it.

A next step is to guarantee the integrity and confidentiality of a REST message. Signing and possibly encrypting the message body is not sufficient as the HTTP request line and headers are also part of a REST message. Although no standards exist yet that define how to do message level security for REST services, big service providers such as Amazon do implement it.

### 3.1.3 Web Site (browser) Authentication

There are three big standards that define SSO and brokered authentication on the web.

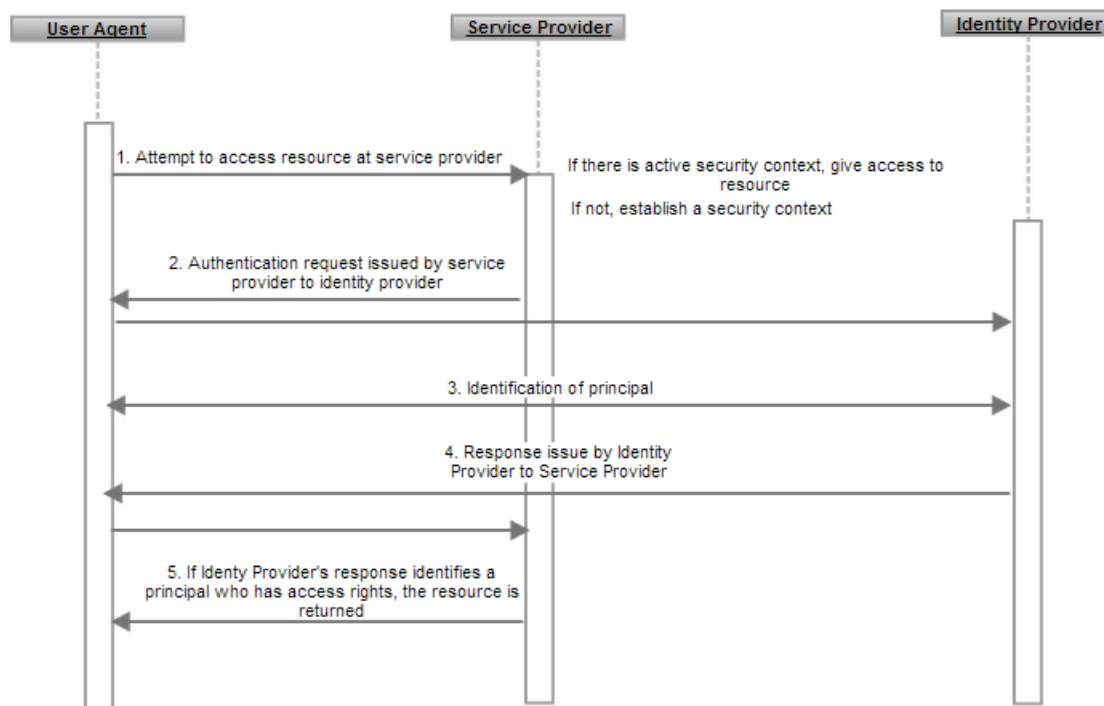
1. OpenID
  1. The SAML Web Browser SSO Profile
  2. The WS-Federation Passive Requestor Profile

**OpenID** [2] is a lightweight HTTP-based profile for Single Sign-On and attribute exchange. OpenID is mainly oriented towards browsers, not really towards REST and SOAP web services called by enhanced clients such as server-side HTTP clients. A record standardisation effort "OpenID Connect" tries to solve this problem in OpenID Connect 1.0 (2014) [3]. OpenID Connect came too late though to be considered for the CHIC Security Framework. Within the course of CHIC it should be watched closely whether it gains any track and could as a result later on in the project be included in the framework. OpenID does not use SAML as token format. As SAML is the token format used to

protect Web Services, it would be preferable stay with a SAML based standard for protecting web sites.

The **WS-Federation Passive Requestor Profile** [18] defines how passive requestors such as a browser can obtain a security token, this in contrast to the WS-Federation Active Requestor Profile as explained in 3.1.2.1. As CHIC already uses WS-Federation for active requestors it could also be a good choice for Web Browser Authentication. The WS-Federation Passive Requestor Profile is mainly implemented by Microsoft though and not as much in other frameworks.

The **SAML Web Browser SSO Profile** is part of the SAML (Security Assertion Markup Language) 2.0 standard [1]. SAML, as explained in 3.1.1, is an XML-based protocol making it possible to exchange authentication and authorization data within or between security domains. The exchange is done by using signed assertions containing identity information. The SAML standard mainly focuses on how to do this exchange through a browser (e.g. the SAML Web Browser Single Sign-On Profile).



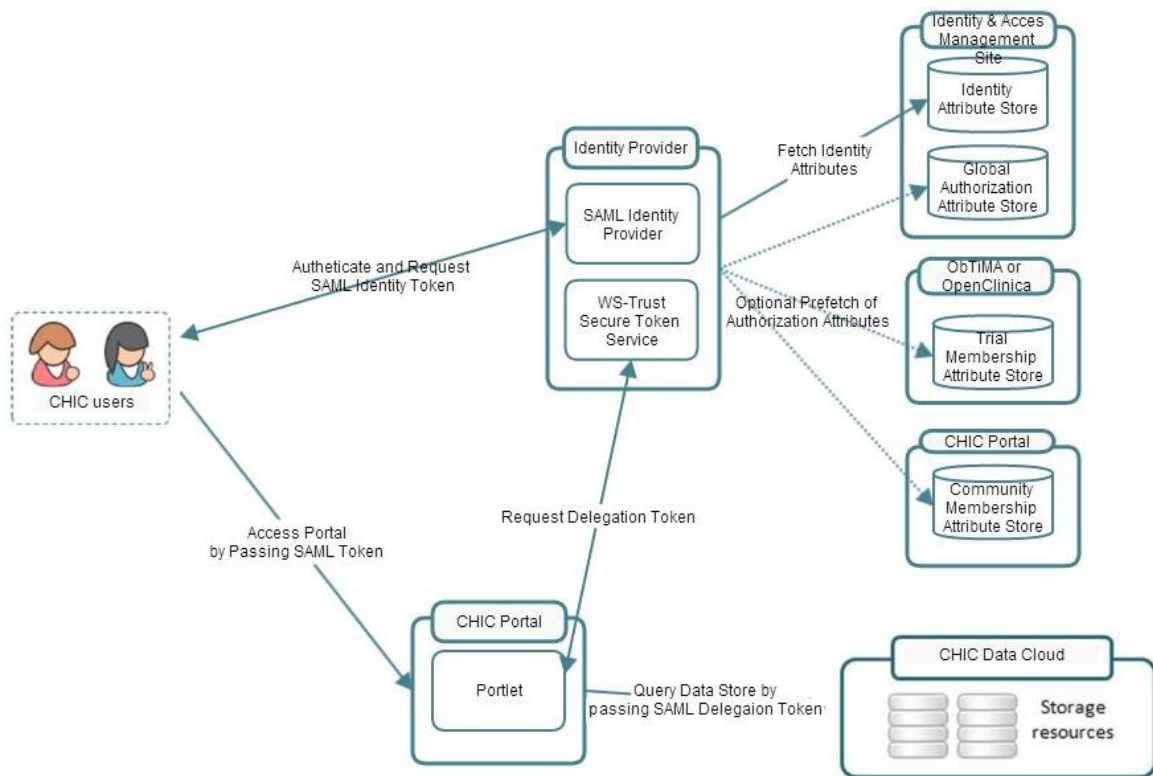
**Figure 3 SAML Browser SSO Profile**

The Web Browser SSO Profile is initiated by an end user who visits a protected web site, also called a relying party or Service Provider (SP). The SP redirects the user to the assertion provider (also called Identity Provider (IdP)) passing through an authentication request. The IdP will request the user to authenticate and upon successful authentication the IdP will issue an identity assertion for the user containing all information on the user needed by the SP to authenticate and authorise him. The assertion is then sent back to the SP that will use it to determine whether the user is allowed to access the requested resource.

CHIC makes use of the SAML Web Browser SSO Profile for browser-based authentication.



### 3.1.4 Component Overview



**Figure 4 CHIC Security Components (Authentication)**

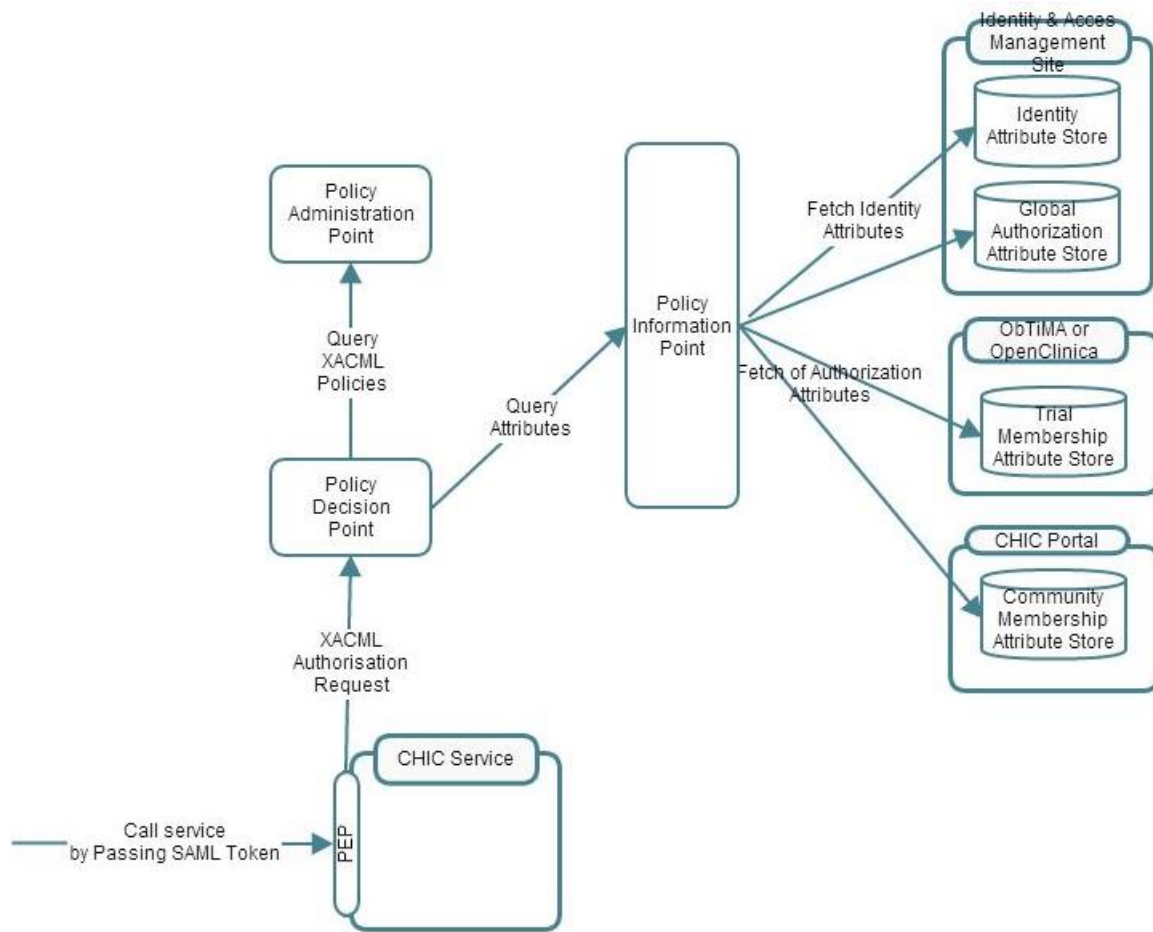
Figure 4 shows the interaction between the different security components.

A user accesses CHIC by browsing to the CHIC portal. Through the CHIC portal the user can initiate the login procedure which redirects the user to the CHIC Identity Provider (IdP) passing through a SAML authentication request. The IdP then requests the user to authenticate. Upon successful authentication, the IdP will query all backend attribute providers (also called PIPs, Policy Information Points in the authorization context) for information on the authenticated user. The IdP issues a SAML identity assertion, including the queried information, and redirects the user back to the portal where the assertion is validated.

A rendered portlet is required to obtain a token from the CHIC Secure Token Service (STS) when it needs to call a backend service. Such a token should identify the original end user performing the action and the portlet or portal as intermediate service. This is also called a delegation token. A portlet obtains a delegation token by calling the STS passing through the end user's identity assertion (issued by IdP as explained above). Upon successful authentication and after validation that the given portlet is indeed allowed to delegate in name of the user, a delegation token is issued to the portlet. By using the delegation token the portlet can call the backend service.

## 3.2 Authorisation

### 3.2.1 Component Overview



**Figure 5 CHIC Security Components (Authorisation)**

Figure 5 shows the interaction between the different authorization components.

A Policy Enforcement Point (PEP) integrates a CHIC service in the authorization framework. When a service needs an authorization decision, the PEP will create an authorization request, authenticate on the CHIC Policy Decision Point (PDP) and will parse the returned authorization decision. The PEP will populate the authorization request with all information available within the service that might be relevant concerning the subject performing the action, the resource on which the action is performed and the actual action.

Upon receiving an authorization request the PDP will query the Policy Administration Point (PAP) for relevant policies to be evaluated. During policy evaluation the PDP might need some additional information on the subject or resource not available in the authorization request. Such additional information is obtained by querying the CHIC Policy Information Points such as the Identity Attribute Store, the CHIC portal and clinical trial software (e.g. ObTiMA or OpenClinica).

## 3.3 Auditing

A successful auditing system requires high quality information on events with potential hazardous security risks. The provision of this information is primarily the responsibility of the service provider,



but well-coordinated cooperation between the developers of service providers, and audit engineers will be necessary to enrich the audit messages with enough data to provide clear and sufficient audit trails in a production environment.

The preferred audit data model is based on XDASv2 [24].

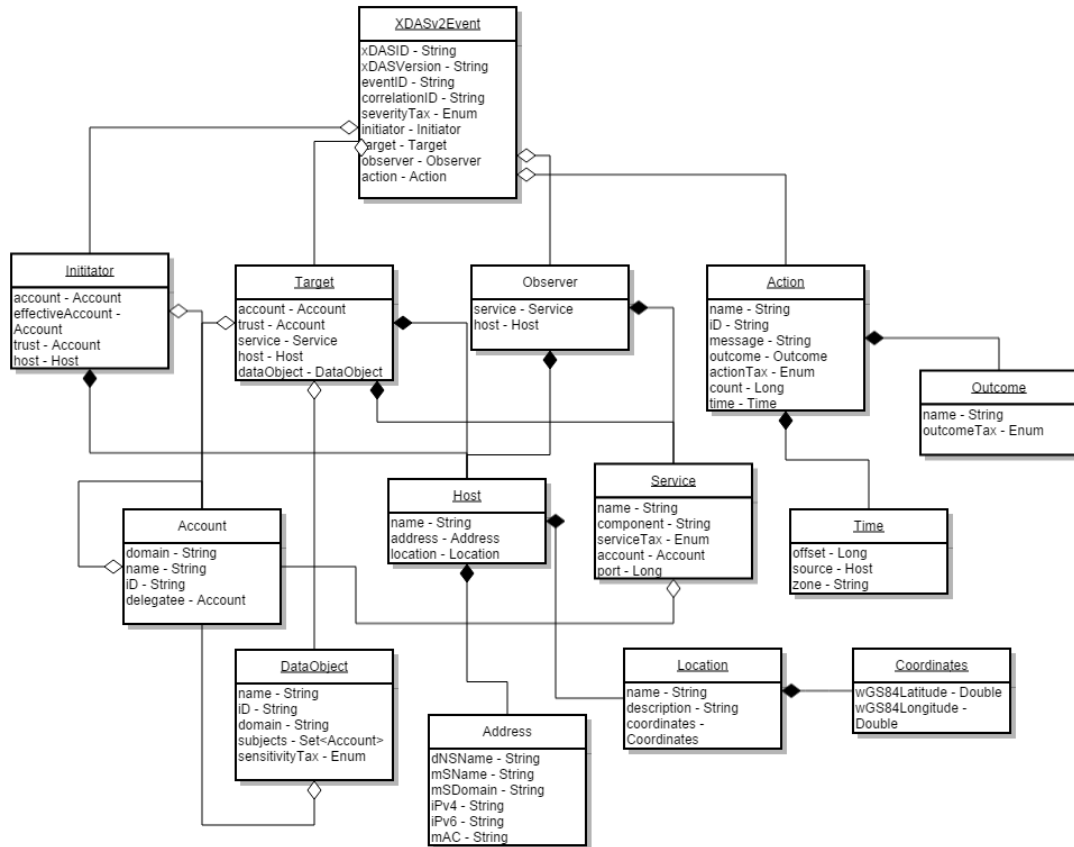


Figure 6 Audit Data Model

### 3.3.1 Data Model

Below follows a description of the main elements of an audit message. The elements that are not described below are self-explanatory.

#### 3.3.1.1 XDASv2Event

At the heart of the model is the XDASv2Event object. This object contains an id (eventID) and might contain a reference to another event (correlationID). It also contains the main actors of an event: the initiator, the target and the observer of the event, along with the action undertaken, and a term indicating the severity of the event.

#### 3.3.1.2 Initiator

The initiator indicates the entity that requests an action to be performed. It contains the following elements:

- **Account:** An account representing the principal (person or service) performing the request. This account might contain another delegation account or chain of accounts to represent previous requests made at other services that lead directly up to the event.
- **EffectiveAccount:** The account that actually performs an action.
- **Host:** Represents the host of the initiator.

#### **3.3.1.3 Target**

The target represents the entity that the initiator wants to get access to as an immediate consequence of the event.

- **Account/dataobject:** This is the principal/object that the initiator tries to get access to/perform an action on
- **Service:** This is the service that provides the requested access to the target.
- **Host:** Represents the domain at which the target service is running

#### **3.3.1.4 Observer**

- **Service:** This is the service at which the event takes place.
- **Host:** Represents the domain at which the events take place.

#### **3.3.1.5 Action**

The action object represents what action was actually taken at this event. It contains the action taxonomy to classify the action, an outcome object to describe the outcome of the action, the time at which the action takes place, and a message, which the service that emitted the event can provide to add more context to the event.

#### **3.3.1.6 Account**

An account represents a principal. A principal can be a person or a service, or even a temporary entity with case-dependent rights. An account might contain a reference to another account, to represent a delegation chain (see Account in the Initiator object).

#### **3.3.1.7 Service**

The service object represents an application that is relevant to the event taking place. In the Observer object, it performs a certain action; in the Target object it provides access to a data object or principal. It contains a classification of the service, the account of the service and the name of the component of the infrastructure it belongs to.

#### **3.3.1.8 Host**

The host object provides the address through which the host can be reached, and the physical location of the server at which the host runs, if applicable.

### 3.3.1.9 DataObject

This object represents information stored somewhere in a database and if present, will only be found in the Target object. If the data is related to principals, this information can be stored in the Subjects component. The DataObject also shows the sensitivity of the target information.

## 3.3.2 Component Overview

Auditing is managed centrally in CHIC. All services within the CHIC domain may send audit log messages to the CIAM auditing service. The CIAM audit system (Figure 7) consists of four major components:

1. **Message Collector and Parser:** This component publishes endpoints for service providers to send messages to.
2. **Message Queue:** There are two instances of the message queue. A first one that queues message coming from the different source interfaces and a second one that gathers parsed audit messages coming from different instances of the collector and parser.
3. **Audit Message Repository:** This is the object repository containing all audit trails.
4. **Message Viewer:** An auditor can use this web-based viewer to view audit information

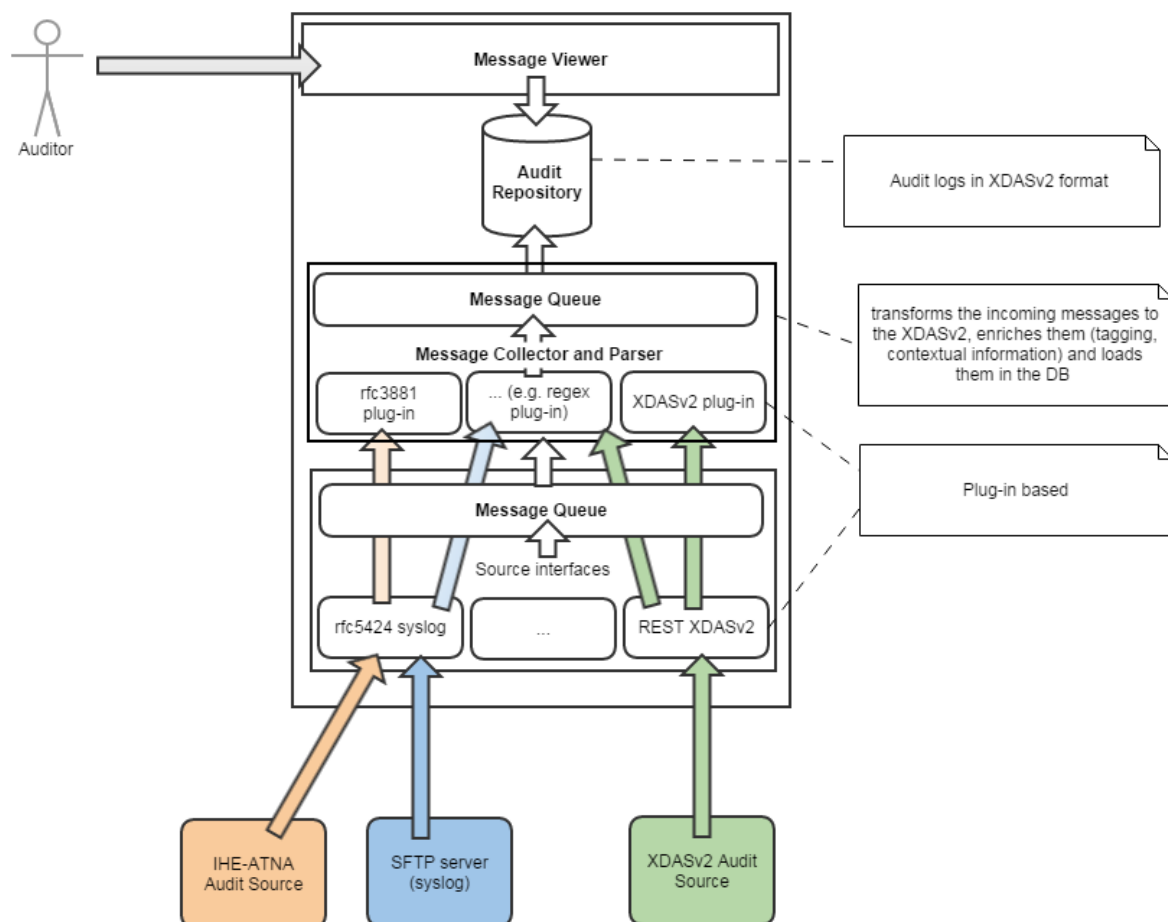


Figure 7 Audit Conceptual Architecture

### 3.3.2.1 Message Collector and Parser

The goal of this component is threefold:

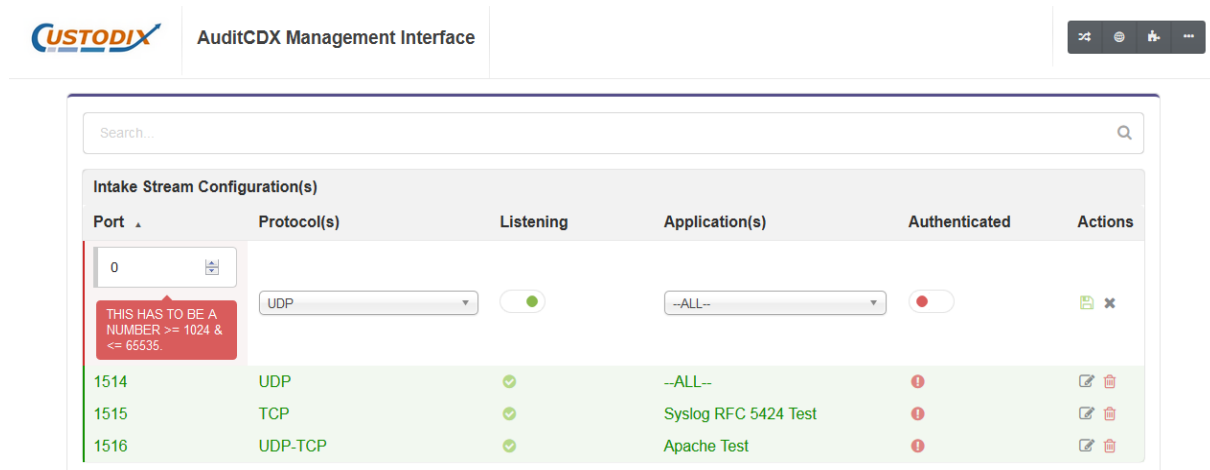
1. Collecting messages from service providers:

This component publishes endpoints. Service providers can connect to these end points to send audit log messages. The figure below (Figure 8) depicts how these endpoints can be configured in the CHIC platform. Following fields can be set:

- Port: On what port is the endpoint listening
- Protocol: What's the transport level protocol
- Listening: To enable/disable the endpoint
- Applications: This field is the 'name' of the endpoint. This name can be used to query audit trails in the audit viewer
- Authenticated: If true the end-point is secured using ws-security. A security header must be set in request, containing a SAML token.

2. *Parse the message and convert to XDASv2 format.*

3. *Put the parse message on the central queue.*



**Figure 8 Audit Management Interface**

### 3.3.2.2 Message queue

The Message queue is provided by employing RabbitMQ [27] as a message broker. RabbitMQ is an open source message broker that implements the Advanced Message Queuing Protocol (AMQP). RabbitMQ manages messages coming from message producing processes in one or more queues, and sends them to appropriate message consumers. RabbitMQ is highly configurable, and provides a web based user interface to manage the different queues.

### 3.3.2.3 Audit Repository

As Audit Repository the open source NoSQL database solution, MongoDB [28], is used, working with JSON (-like) documents. It does not require a predefined structure of the objects to be stored, which provides some flexibility for the format of the messages. However it is still advised to keep to the proposed XDASv2 model (see 3.3.1) as well as possible to make standardised manipulation of the information possible in the web viewer.

MongoDB is highly scalable, which might prove to be important as the amount of logged data increases. Big data operations such as MapReduce are also supported which could be a powerful tool in the future to analyse the enormous amounts of incoming messages.

### 3.3.2.4 Web Viewer

The web viewer provides the frontend through which an authorised user can query stored messages. This will usually be a CDP (Centre for Data Protection) operative. The user is able to filter messages by relevant parameters:

- Initiator (user or service)
- Observer (service)
- Target (server, object or user)
- Action type
- Outcome type
- Time of occurrence by selecting a time window

The user will be able to select individual messages and review their contents. This allows also a more granular display of events, which helps auditors to pinpoint the source of a security breach and intent of the perpetrator. The web viewer provides the means to review an audit trail and precisely track the series of events that led up to a potential security breach. This adds a powerful tool to not only analyse security breaches after the events have taken place, but also to proactively seal potential breaches before they are discovered by users with malicious intents. A thorough testing by investigators of the auditing system with use cases that resemble (potential) security breaches will uncover holes in the auditing system, such as events that have been overlooked, or relevant information that has been omitted in the event messages.

The screenshot shows the 'AuditCDX Web Viewer' interface. At the top, there's a header with the 'USTODIX' logo and the title 'AuditCDX Web Viewer'. Below the header is a search form with several input fields: 'Initiator', 'Action Type', 'Observer', 'Target', 'Time from', 'Time to', 'Outcome', 'Severity', and 'Sensitivity'. A 'FILTER' button is visible next to the 'Outcome' field. Below the search form is a table displaying a list of audit events. The table has columns for 'Time', 'Action', 'Initiator', 'Observer', 'Target', 'Outcome', 'Severity', and 'Sensitivity'. The 'Outcome' column is currently expanded, showing a list of possible outcomes: DENY\_DENIAL, DENY\_INSUFFICIENT\_AUTHC, DENY\_INVALID\_CREDENTIAL, DENY\_INVALID\_IDENTITY, FAIL\_ALREADY\_ENABLED, FAIL\_BUSY, FAIL\_DISABLED, FAIL\_ENTITY\_NON\_EXISTENT, FAIL\_FAILURE, and FAIL\_HARDWARE\_FAILURE. The table shows various events such as 'Modify Process Context', 'Configure Audit Service', 'Query Data Item Association', 'Federate Identity', 'Terminate Session', 'Modify Account Security Token', 'Shutdown System', 'Query Role', 'Disable Role', 'Remove Service', 'Modify Service Configuration', 'Delete Role', 'Invoke Service', and 'Query Data Item Association'. The 'Outcome' column contains icons representing the result of each action, such as a green checkmark for success or a red 'X' for failure. The 'Severity' column contains icons representing the severity of the event, such as a green circle for low severity or a red exclamation mark for high severity. The 'Sensitivity' column contains icons representing the sensitivity of the event, such as a green circle for low sensitivity or a red exclamation mark for high sensitivity. At the bottom of the table, there are navigation controls including a page number '1' and arrows for navigating between pages.

Figure 9 Audit Web Viewer

### 3.4 Security Vocabulary

This section provides a preliminary definition of the security vocabulary. It will be further extended in the next security and architectural deliverables.

Attribute Name	URN	Description
UUID	urn:custodix:ciam:1.0:principal:uuid	The federated ID of the user
Alternative UUID	urn:custodix:ciam:1.0:principal:uuid:alias	The alternative federated ID's of the user (after account merging)
Uid (userid)	urn:oid:0.9.2342.19200300.100.1.1	Contains a human readable identifier: the username of the user.
Domain uuid	urn:custodix:ciam:1.0:domain:uuid	The security domains uuid this user is member of.
Domain name (unique)	urn:custodix:ciam:1.0:domain:name	The security domains name this user is member of.
Email	urn:custodix:ciam:1.0:principal:email urn:oid:0.9.2342.19200300.100.1.3	Registered email address
Title	urn:oid:2.5.4.12	Title of the user
Given name	urn:oid:2.5.4.42	First name of the user
Name	urn:oid:2.5.4.4	Last name of the user
Organisation name	urn:oid:2.5.4.10	Name of the organisation the user is part of
Role	urn:oid:2.5.6.8	CHIC role
annexCContractSigned	urn:chic:1.0:principal:contracts:annexCSigned	Whether a user has signed the required contracts for data access.

## 4 Security Framework Interoperability Guidelines

### 4.1 Transport Layer Security

All web service communication within CHIC MUST go over a secured HTTP connection by using the HTTP SSL/TLS protocol as specified in [1],[20],[21],[22].

- SSL version 1.0 or 2.0 MUST NOT be used.
- TLS 1.1/1.2 are RECOMMENDED
- SSL version 3.0 or TLS 1.0/1.1/1.2 MUST be used.

*TLS and SSL are cryptographic protocols aimed to provide communication security over the Internet. They use X.509 certificates (asymmetric cryptography) to assure the counterparty whom they are communicating with, and to exchange a symmetric key. This allows data/message confidentiality and message authentication codes for message integrity and as a by-product message authentication. In CHIC we limit the use of these protocols to SSL version 3.0 or TLS version 1.0/1.1/1.2. SSL version 1.0 was never made public and SSL version 2.0 is considered insecure (vulnerable to attacks). Transport Layer Security (TLS) must be enabled to protect web service endpoints and secure communications between CHIC service providers. SSL/TLS only provide protection between adjacent HTTP Nodes. It therefore does not protect HTTP SOAP or REST messages when the message path contains intermediaries. SSL/TLS should thus always be in conjunction with message level security (e.g. encryption of SOAP and REST messages) when the message path contains intermediaries or when these security functions are required to be performed independently of the connection.*

Administrators MUST follow the NHS Approved Cryptographic Algorithms Good Practice Guideline [23] for choosing the SSL/TLS cryptographic algorithms.

*In SSL and TLS, choices of algorithms are expressed as ciphersuites. The Approved Cryptographic Algorithms Good Practice Guideline specifies the ciphersuites that are required, recommended, discouraged and prohibited. The ciphersuites defined in the SSL and TLS specifications that use anonymous Diffie-Hellman (i.e. those that have DH\_anon in their symbolic name) are vulnerable to man-in-the-middle attacks and MUST NOT be used. It is also RECOMMENDED that ciphersuites that include MD5 (i.e. those that have MD5 in their symbolic name) be avoided, due to known security weaknesses of the MD5 algorithm. It is RECOMMENDED that ciphersuites that use 40 or 56 bit keys be avoided, due to their relative ease of compromise through brute-force attack.*

Following Ciphersuites are RECOMMENDED for secure communication between CHIC web components:

- TLS\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA
- SSL\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA
- TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA
- SSL\_RSA\_WITH\_AES\_128\_CBC\_SHA

*The specified algorithm suites are considered to be widely-implemented, secure and interoperable.*

The TLS/SSL authentication keys **MUST** be signed by a trusted certificate authority.

- Authentication keys **MUST** be equal or higher than 2048 bit

*Self-signed keys are not permitted in the CHIC production environment. They should be signed by a Central Authority (CA) like Verisign, Comodo Group, Go Daddy, GlobalSign, etc. keys less than 2048 bit are not considered secure anymore.*

## **4.2 Message Protocols and Message Layer Security**

The two most commonly used message protocols, SOAP and REST, should be used for CHIC web services.

In addition to encryption on the transport layer, all sensitive messages within CHIC **SHOULD** be encrypted.

### **4.2.1 SOAP**

CHIC SOAP Web Services **MUST** use either SOAP version 1.1 or 1.2. CHIC SOAP Web Services **SHOULD** use version 1.2.

*SOAP (Simple Object Access Protocol), is a protocol specification for exchanging structured information in the implementation of Web Services in computer networks. It relies on XML Information Set for its message format, and usually relies on other Application Layer (HTTP/SMTP/...) protocols. The SOAP version can be 1.1 or 1.2. We leave the decision to the implementers because most web services implementations still use the 1.1 version as standard and 1.1 does not limit us in other guidelines<sup>3</sup>.*

CHIC SOAP Web Services **MUST** use the WSDL protocol as specified in [4] for describing network services.

- WSDL version 1.0/1.1 **MUST** be used.
- WSDL version 1.2 or 2.0 **MUST NOT** be used.

*WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. WSDL version 1.1 is the formalization of version 1.0 which contains no major changes. WSDL version 2.0 (original drafted as version 1.2) was designed to solve the interoperability issues found in WSDL 1.0/1.1. But very few vendors support WSDL 2.0 today since it is still not completely interoperable. This is why we demand the use of version 1.0/1.1.*

The Types element of each WSDL defined in CHIC **MUST** contain XSD as the canonical type system. Other systems **MUST NOT** be used.

---

<sup>3</sup> Differences between 1.1 and 1.2 can be found here:

[http://pic.dhe.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=%2Fcom.ibm.websphere.wsfeplmu%2Fplatform.doc%2Finfo%2Fae%2Fae%2Ffwbs\\_soapverdiffs.html](http://pic.dhe.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=%2Fcom.ibm.websphere.wsfeplmu%2Fplatform.doc%2Finfo%2Fae%2Fae%2Ffwbs_soapverdiffs.html).



*WSDL does not restrict the Types element to the commonly used XSD specification, other type systems can be added via extensibility elements. In CHIC we restrict this flexibility to XSD, as XSD is well supported in software.*

CHIC SOAP web services MUST use the SOAP 1.1 [see section 3 in [4]] or SOAP 1.2 [5] WSDL bindings. Other possible bindings MUST NOT be used.

Bindings in CHIC WSDLs SHOULD use the document/literal wrapped pattern.

*document/literal is the most commonly used binding pattern and should be used as much as possible in CHIC. Exceptions are the cases described in <http://www.ibm.com/developerworks/webservices/library/ws-whichwsdl/>*

#### **4.2.1.1 WS-Policy**

CHIC SOAP web services MUST use the WS-Policy protocol as specified in [8] to describe the policies of entities in the web services.

WS-Policy version 1.5 MUST be used.

*The Web Services Policy Framework provides a general purpose model and corresponding syntax to describe the policies of entities in a Web services-based system. Web Services Policy Framework defines a base set of constructs that can be used and extended by other Web services specifications to describe a broad range of service requirements and capabilities. In CHIC we use the version 1.5.*

#### **4.2.1.2 WS-Addressing**

CHIC SOAP web services MUST use the WS-Addressing protocol as specified in [9] and [10] to address web services and messages.

WS-Addressing version 1.0 MUST be used

*WS-Addressing provides transport-neutral mechanisms to address Web services and messages. Specifically, this specification defines XML elements to identify Web service endpoints and to secure end-to-end endpoint identification in messages. Version 1.0 is used in CHIC.*

CHIC SOAP web services MUST use the SOAP 1.1 or SOAP 1.2 [10] WS-Addressing bindings. Other possible bindings MUST NOT be used.

*In CHIC WS-Addressing is required, so we do not allow the optional parameter to be set to true. A WS-Policy containing this assertion will be included in the WSDL of each web service. Anonymous responses are allowed in CHIC.*

#### **4.2.1.3 WS-Security**

CHIC web service requesters/providers MUST use the WS-Security protocol as specified in [6] & [7] for exchanging security tokens between other CHIC web service requesters/providers.

- WS-Security version 1.0 or 1.1 MUST be used.
- WS-Security version 1.1 SHOULD be used.

*WS-Security is a specification for signing (assure integrity), encrypting (assure confidentiality) and attaching security tokens (identity provision) to CHIC SOAP messages. Version 1.0 does not differ that much from version 1.1 (1.1 is backwards compatible with 1.0).*

WS-Security headers MUST use SAML Assertion Tokens [6] & [1] as security tokens.

- User Name Tokens, Binary Security Tokens, XML Tokens, EncryptedData Tokens and SecurityTokenReference Tokens as specified in [6] & [7] MUST not be used.

*SAML Assertion Tokens are core to the CHIC security framework. They are both used in web as web service based authentication.*

WS-Security headers MUST include the /wsse:Security/@S11:mustUnderstand or /wsse:Security/@S12:mustUnderstand attribute as specified in [6]. This attribute MUST be set true or 1.

*The mustUnderstand attribute must be set to true because we require that the security header is mandatory for the recipient to process (if the recipient does not support security headers an exception will be thrown).*

WS-Security headers MUST include a Security Timestamp. The Security Timestamp MUST contain a creation time and an expiration time. Recipients MUST discard (ignore) any message whose security semantics have passed their expiration.

*The inclusion of both a creation and expiration timestamps reduced the overall risk of replay attacks by third-parties.*

#### 4.2.1.4 WS-Trust

CHIC web service requesters/providers MUST use the WS-Trust protocol as specified in [11] & [12] to construct trusted SOAP message exchanges between other CHIC web service requesters/providers.

- WS-Trust version 1.4 including 1.3 MUST be used.

*WS-Trust 1.4 provides extensions to WS-Security for enabling applications to construct trusted SOAP message exchanges. This trust is represented through the exchange and brokering of security tokens. This specification provides a protocol agnostic way to issue, renew, and validate these security tokens. In CHIC WS-Trust version 1.4 is required because this version supports act-as tokens, which is a security requirement in CHIC.*

##### 4.2.1.4.1 Request Token

The <wst:RequestSecurityToken> element (RST) is used to request a security token (for any purpose). This element SHOULD be signed by the requestor, using the tokens contained/referenced in the request, which are relevant to the request. If using a signed request, the requestor MUST prove any required claims to the satisfaction of the security token service.

If a parameter is specified in a request that the recipient doesn't understand, the recipient SHOULD fault. The syntax for this element is as follows:

```
<wst:RequestSecurityToken Context="..." xmlns:wst="...">
  <wst:TokenType>...</wst:TokenType>
```

```
<wst:RequestType>...</wst:RequestType>
<wst:SecondaryParameters>...</wst:SecondaryParameters>
<wsp:AppliesTo>...</wsp:AppliesTo>
<wst:Claims Dialect="...">...</wst:Claims>
<wst:Entropy>
  <wst:BinarySecret>...</wst:BinarySecret>
</wst:Entropy>
<wst:Lifetime>
  <wsu:Created>...</wsu:Created>
  <wsu:Expires>...</wsu:Expires>
</wst:Lifetime>
</wst:RequestSecurityToken>
```

wst:TokenType is an OPTIONAL element. If it is not specified in an issue request, it is RECOMMENDED that the OPTIONAL element <wsp:AppliesTo> be used to indicate the target where this token will be used.

#### 4.2.1.4.2 Return Token

When returning a security token, the following OPTIONAL elements MAY be included in the response. Security tokens are returned embedded in a wst:RequestSecurityTokenResponse element. The syntax for these elements is as follows:

```
<wst:RequestSecurityTokenResponse xmlns:wst="...">
  <wst:TokenType>...</wst:TokenType>
  <wst:RequestedSecurityToken>...</wst:RequestedSecurityToken>
  ...
  <wsp:AppliesTo xmlns:wsp="...">...</wsp:AppliesTo>
  <wst:RequestedAttachedReference>
  ...
</wst:RequestedAttachedReference>
  <wst:RequestedUnattachedReference>
  ...
</wst:RequestedUnattachedReference>
  <wst:RequestedProofToken>...</wst:RequestedProofToken>
  <wst:Entropy>
    <wst:BinarySecret>...</wst:BinarySecret>
  </wst:Entropy>
  <wst:Lifetime>...</wst:Lifetime>
</wst:RequestSecurityTokenResponse>
```

<wst:RequestedSecurityToken> is an OPTIONAL element that is used to return the requested security token. This element is OPTIONAL, but it is REQUIRED that at least one of <wst:RequestedSecurityToken> or <wst:RequestedProofToken> is returned unless there is an error or part of an on-going message exchange (e.g. negotiation). WS-Trust MUST use SAML 2.0 token.

#### 4.2.1.4.3 wsp:AppliesTo

Both the requestor and the issuer can specify a scope for the issued token using the <wsp:AppliesTo> element. If a token issuer cannot provide a token with a scope that is at least as broad as that requested by the requestor then it SHOULD generate a fault. This section defines some rules for interpreting the various combinations of provided scope:

- If neither the requestor nor the issuer specifies a scope, then the scope of the issued token is implied.
- If the requestor specifies a scope and the issuer does not, then the scope of the token is assumed to be that specified by the requestor.

- If the requestor does not specify a scope and the issuer does specify a scope, then the scope of the token is as defined by the issuer's scope.
- If both requestor and issuer specify a scope, then there are two possible outcomes:
  - If both the issuer and requestor specify the same scope, then the issued token has that scope.
  - If the issuer specifies a wider scope than the requestor, then the issued token has the scope specified by the issuer.
- The requestor and issuer **MUST** agree on the version of [14] used to specify the scope of the issued token. The Trust13 assertion in [16] provides a mechanism to communicate which version of [14] is to be used.

#### 4.2.1.5 WS-SecurityPolicy

CHIC web service providers **MUST** use the WS-SecurityPolicy protocol as specified in [16] to define security policy assertions.

- WS-SecurityPolicy version 1.2/1.3 **MUST** be used.

WS-SecurityPolicy defines a set of security policy assertions for use with the [14] framework with respect to security features provided in WSS: SOAP Message Security [6], [7] & [12]. In CHIC we use the latest version of the standard.

CHIC web service providers **MUST** include the Trust13 Assertion as defined in [16] through the [15] in their WSDL. Following sub-assertions **MUST** be included:

- MustSupportIssuedTokens Assertion
- RequireClientEntropy Assertion
- RequireServerEntropy Assertion

This MustSupportIssuedTokens Assertion indicates that the wst:IssuedTokens header is supported as described in WS-Trust. The RequireClientEntropy Assertion indicates that client entropy is **REQUIRED** to be used as key material for a requested proof token. The RequireServerEntropy Assertion indicates that server entropy is **REQUIRED** to be used as key material for a requested proof token.

CHIC web service providers **MUST** include the Wss11 Assertion as defined in [16] through the [15] in their WSDL. Following sub-assertions **MUST** be included:

- MustSupportRefIssuerSerial Assertion
- MustSupportRefThumbprint Assertion
- MustSupportRefEncryptedKey Assertion

The MustSupportRefIssuerSerial Assertion indicates that the initiator and recipient **MUST** be able to process references using the issuer and token serial number. The MustSupportRefThumbprint Assertion indicates that the initiator and recipient **MUST** be able to process references using token thumbprints. The MustSupportRefEncryptedKey Assertion indicates that the initiator and recipient **MUST** be able to process references using EncryptedKey references.

CHIC web service providers **MUST** include the TransportBinding Assertion as defined in [16] through the [15] in their WSDL. Following sub-assertions **MUST** be included:

- IncludeTimestamp Assertion
- TransportToken Assertion containing following sub-assertions:

- HttpsToken Assertion with parameter “RequireClientCertificate” set to “false”
- AlgorithmSuite Assertion containing following sub-assertions:
  - Basic128 Assertion
- Layout Assertion containing following sub-assertions:
  - Lax Assertion

The IncludeTimestamp Assertion indicates that a wsu:Timestamp element MUST present in the wsse:Security header, this is required to avoid replay attacks. TransportToken Assertion indicates that communication MUST go over HTTPS, this will assure confidentiality and integrity in the messages. The AlgorithmSuite Assertion specifies that the Basic128 algorithm suite (see [WS-SecurityPolicy13]) is REQUIRED for performing cryptographic operations with symmetric or asymmetric key based security tokens. The Layout Assertion indicates that Items are added to the security header in any order that conforms to WSS: SOAP Message Security.

CHIC web service providers MUST include the SignedSupportingTokens Assertion as defined in [16] through the [15] in their WSDL. Following sub-assertions MUST be included:

- IssuedToken Assertion with parameter “IncludeToken” set to “<http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/IncludeToken/AlwaysToRecipient>” containing following elements/assertions:
  - Issuer element containing the URL to the CHIC STS WSDL
  - RequestSecurityTokenTemplate element containing following elements:
    - TokenType element with value “urn:oasis:names:tc:SAML:2.0:assertion”
    - KeyType element with value “<http://docs.oasis-open.org/ws-sx/ws-trust/200512/Bearer>”
  - RequireInternalReference Assertion

The IssuedToken Assertion indicates that the security token MUST be included in all messages sent from initiator to the recipient. This token MUST NOT be included in messages sent from the recipient to the initiator. It also defines that an internal reference is REQUIRED when referencing to the security token. The RequestSecurityTokenTemplate element REQUIRES that the TokenType (<urn:oasis:names:tc:SAML:2.0:assertion>) and KeyType (<http://docs.oasis-open.org/ws-sx/ws-trust/200512/Bearer>) elements MUST be copied into the wst:SecondaryParameters of the RST request sent to the specified issuer. The Issuer element contains a reference to the issuer (CHIC STS WSDL) for the issued token.

#### 4.2.1.6 WS-SecureConversation

CHIC web service providers MUST use the WS-SecurityConversation protocol as specified in [17] to allow security context establishment and sharing, and session key derivation.

- WS-SecurityPolicy version 1.4 MUST be used.

*WS-SecureConversation defines extensions to WS-Security to allow security context establishment and sharing, and session key derivation. This allows contexts to be established and potentially more efficient keys or new key material to be exchanged, thereby increasing the overall performance and security of the subsequent exchanges. In CHIC we use the latest version of the standard.*

#### 4.2.1.7 WS-Federation

CHIC web service requesters/providers MUST use the WS-Federation protocol as specified in [18] to allow different security realms to federate.

- WS-Federation version 1.2 MUST be used.

*WS-Federation 1.1 defines mechanisms for allowing disparate security realms to broker information on identities, identity attributes and authentication.*

CHIC web service requesters/providers MUST use the WS-Federation Active Requestor Profile for requesting identity tokens from an STS and exchange these tokens with other CHIC web service providers.

*The WS-Federation Active Requestor Profile defines how enhanced clients can request identity tokens from an identity provider (called Security Token Service or STS in WS-\*) and exchange it with a service provider.*

CHIC web pages MUST NOT use the WS-Federation Passive Requestor Profile for requesting identity tokens from an STS and exchange these tokens with other CHIC web service providers.

*The WS-Federation Passive Requestor Profile defines a protocol for browsers to exchange identity token between identity and service providers.*

#### 4.2.2 REST

CHIC does allow the use of REST web services. These Security guidelines currently do not define REST protocols, encryption and signing guidelines. A next iteration of this deliverable will be appended with REST security guidelines. Other sections such as the integration tutorials do explain though how REST services can be protected and make use of the CHIC security services and components.

### 4.3 Authentication Protocols

#### 4.3.1 Passive Protocol for Web Browsers

Web sites (service providers) MUST allow integration with CHIC security at minimal with the following subset of the SAML 2.0 standard:

1. The service provider MUST support the following bindings of the Web Browser SSO Profile:
  - a. for <AuthnRequest> from SP to IdP:
    - i. HTTP redirect
    - ii. HTTP POST
  - b. for IdP <Response> to SP:
    - i. HTTP POST
    - ii. HTTP artefact
2. The website must support at least the following bindings of the Single Logout Profile:

- a. for IdP-initiated single logout the HTTP redirect and SOAP binding must be supported.
  - b. for SP-initiated single logout the HTTP redirect and SOAP binding must be supported.
3. The website must publish its SAML metadata.

```
<!-- Sample metadata of a compatible service provider -->
<?xml version="1.0" encoding="UTF-8"?>
<md:EntityDescriptor entityID="..."
  xmlns:md="urn:oasis:names:tc:SAML:2.0:metadata">
  <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    ...
  </ds:Signature>
  <md:SPSSODescriptor AuthnRequestsSigned="true"
    WantAssertionsSigned="true"
    protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol">
    <md:KeyDescriptor use="signing">
      <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <ds:X509Data>
          <ds:X509Certificate>...</ds:X509Certificate>
        </ds:X509Data>
      </ds:KeyInfo>
    </md:KeyDescriptor>
    <md:KeyDescriptor use="encryption">
      <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <ds:X509Data>
          <ds:X509Certificate>...</ds:X509Certificate>
        </ds:X509Data>
      </ds:KeyInfo>
    </md:KeyDescriptor>
    <md:SingleLogoutService
      Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect"
      Location="..." />
    <md:SingleLogoutService
      Binding="urn:oasis:names:tc:SAML:2.0:bindings:SOAP"
      Location="..." />
    <md:NameIDFormat>urn:oasis:names:tc:SAML:2.0:nameid-
format:persistent</md:NameIDFormat>
    <md:AssertionConsumerService
      Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST"
      Location="..."
      index="0" isDefault="true" />
    <md:AssertionConsumerService
      Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Artifact"
      Location="http..."
      index="1" isDefault="false" />
  </md:SPSSODescriptor>
</md:EntityDescriptor>
```

Figure 10 Example SAML metadata

## 4.3.2 Active Protocol for REST Web Services

## 4.3.3 Active Protocol for SOAP Web Services

## 5 Security Framework Integration Tutorials

### 5.1 *Passive (Browser based) Authentication*

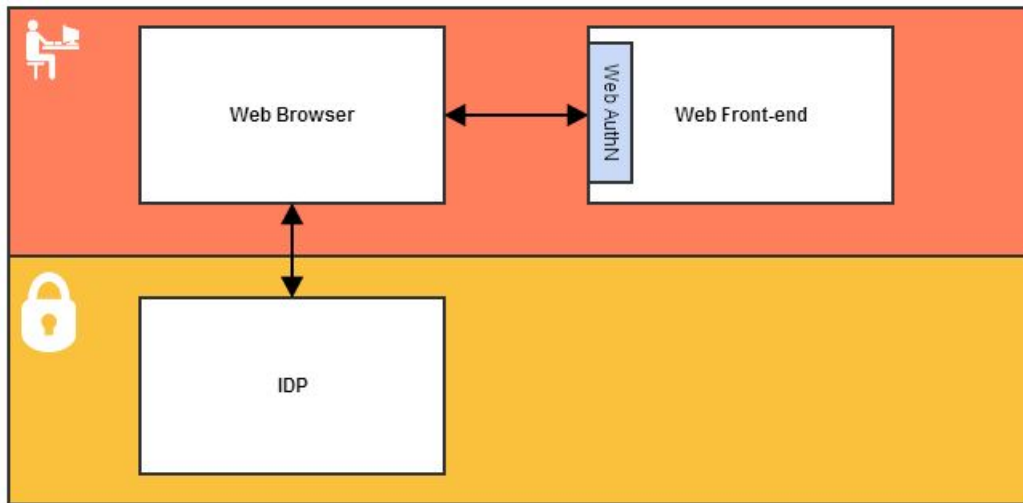


Figure 11 Passive Authentication Profile

#### 5.1.1 Client Side (Browsers)

The SAML Web Browser SSO Profile makes use of standard HTTP Get redirects or POSTs. It will thus by default work with any HTTP compatible browsers and no further integration on the side of the browser is required.

#### 5.1.2 Java Based Service Providers through Spring Security

##### 5.1.2.1 Preconditions

These instructions describe how to configure a Service Provider secured by spring security with local identities as a SAML Service Provider linking the central identity with the local identity. It assumes that the Service Provider is already configured to be using Spring Security and that there is a bean called "userService" which implements "UserDetailsService". This class has a method "loadUserByUsername" that accepts a String (username) and returns a "UserDetails" object.

- Following libraries are required on class path<sup>4</sup>:
  - bcprov-jdk15-144.jar
  - commons-httpclient-3.1.jar
  - commons-lang-2.4.jar
  - joda-time-1.6.2.jar
  - not-yet-commons-ssl-0.3.9.jar
  - opensaml2-2.4.1.jar
  - openws-1.4.1.jar
  - serializer-2.7.0.jar

<sup>4</sup> The libraries can be downloaded from the CHIC wiki by CHIC partners. External users should be able to fetch all these libraries from a public maven repository.



- spring-security-saml2-core
- velocity-1.5.jar
- xalan-2.7.0.jar
- xmlsec-1.4.4.jar
- xmltooling-1.3.1.jar
- Definitions:
  - Example requesting user:
    - username: “elias-chic-test”
    - password: “El1a\$Ner1”
  - IdP Metadata: <https://ciam-dev-chic.custodix.com/idp/shibboleth> (further referred to as “YOUR\_IDP\_METADATA\_URL”)
  - Example web site (service provider) to be integrated: <https://portal-test-chic.custodix.com> (further referred to as “YOUR\_SP\_URL”)

### 5.1.2.2 Tutorial

- 1) Create the file “security.properties” in the WEB-INF folder of your application, with following content

```
#URL to redirect to after succesful authentication
security.saml.successRedirectHandler.defaultTargetUrl=/
#URL to redirect to after succesful logout
security.saml.successLogoutHandler.defaultTargetUrl=/
#Suffix of the login filter, saml authentication is initiated when user browses to this url
security.saml.entryPoint.filterProcessesUrl=/saml/login
#SAML Binding to be used for above entry point url.
security.saml.entryPoint.binding=urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST
#Suffix of the Service Provider's metadata, this url needs to be configured on IDP
security.saml.metadata.filterSuffix=/saml/metadata
#Id of the spring security's authentication manager
security.saml.authenticationManager=authenticationManager
#Whether sessions should be invalidated after logout
security.saml.logout.invalidateHttpSession=true
#Id of the spring security user service that should be called to fetch users.
security.saml.userService=userService
#Alias of the Service Provider
security.saml.sp.alias=CHICTestPortal
#ID of the Service Provider
security.saml.sp.id=chic
#Alias of the Service Provider's signing key
security.saml.sp.signingKeyAlias=YOUR_SERVICE_ALIAS
#URL of the service provider
security.saml.sp.url= YOUR_SP_URL
#URL to the IDP's metadata
security.saml.idp.metadataURL= YOUR_IDP_METADATA_URL
#Alias of the Service Provider's tls key
security.saml.sp.tlsKey=chic_key
#Alias of the Service Provider's encryption key
security.saml.sp.encryptionKeyAlias= chic_key
#path to keystore which contains keys used by the Service Provider
```

```
security.saml.keystore=file\:<PathTOKeyStore>
#keystore's storepass
security.saml.keystore.password=changeit
#keystore's default key
security.saml.keystore.defaultKey= chic_key
#Alias of a key in the keystore
security.saml.keystore.key.alias= chic_key
#Password of that the key with above alias in the keystore
security.saml.keystore.key.password=changeit
#Note that it's only possible to configure one key alias through property file
#If different keys are used for encryption, signing, ...
#the applicationContext-spring-security-saml needs to be updated.
```

Note that it is only possible to configure one key alias through the property file. If different keys are to be used for encryption, signing ... the “applicationContext-spring-security-saml.xml” file, further defined below, needs to be updated.

- 2) Modify this “security.properties” for your Service Provider. See in detail description below (SAML Security Properties Explained).
- 3) Create a spring configuration file “applicationContext-spring-security-saml.xml” in your “WEB-INF” folder.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:security="http://www.springframework.org/schema/security"
       xmlns:beans="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
                           http://www.springframework.org/schema/security
                           http://www.springframework.org/schema/security/spring-security-3.1.xsd">

    <bean id="propertyConfigurer"
          class="org.springframework.web.context.support.ServletContextPropertyPlaceholderConfigurer">
        <property name="locations">
            <list>
                <value>/WEB-INF/security.properties</value>
            </list>
        </property>
        <property name="contextOverride" value="true"/>
        <property name="fileEncoding" value="UTF-8"/>
    </bean>

    <!-- Handler deciding where to redirect user after successful login -->
    <bean id="samlSuccessRedirectHandler"
          class="org.springframework.security.web.authentication.SavedRequestAwareAuthenticationSuccessHandler">
        <property name="defaultTargetUrl" value="\${security.saml.successRedirectHandler.defaultTargetUrl}"/>
    </bean>

    <!-- Handler for successful logout -->
    <bean id="samlSuccessLogoutHandler"
          class="org.springframework.security.web.authentication.logout.SimpleUrlLogoutSuccessHandler">
        <property name="defaultTargetUrl" value="\${security.saml.successLogoutHandler.defaultTargetUrl}"/>
    </bean>
```

```

<!-- Entry point to initialise authentication, default values taken from properties file -->
<bean id="samlEntryPoint" class="org.springframework.security.saml.SAMLEntryPoint">
  <property name="filterProcessesUrl" value="${security.saml.entryPoint.filterProcessesUrl}"/>
  <!-- OPTIONAL property: In case idpSelectionPath property is not set the user will be redirected to the
default IDP -->
  <!--property name="idpSelectionPath" value="/WEB-INF/security/idpSelection.jsp"/-->
  <property name="defaultProfileOptions">
    <bean class="org.springframework.security.saml.websso.WebSSOProfileOptions">
      <property name="includeScoping" value="false"/>
      <property name="binding" value="${security.saml.entryPoint.binding}"/>
    </bean>
  </property>
</bean>

<!-- SAML Authentication Provider responsible for validating of received SAML messages -->
<bean id="samlAuthenticationProvider"
class="org.springframework.security.saml.SAMLAuthenticationProvider">
  <property name="userDetails" ref="userService"/>
</bean>

    <bean id="userService" class="${security.saml.userService}"/>

<!-- OPTIONAL bean: The filter is waiting for connections on URL suffixed with filterSuffix and presents SP
metatdata there -->
<bean id="samlMetadataFilter"
class="org.springframework.security.saml.metadata.MetadataDisplayFilter"/>

<!-- Provider of default SAML Context -->
<bean id="contextProvider" class="org.springframework.security.saml.context.SAMLContextProviderImpl"/>
<!-- Override default authentication processing filter with the one processing SAML messages -->
<bean id="samlProcessingFilter" class="org.springframework.security.saml.SAMLProcessingFilter">
  <property name="authenticationManager" ref="${security.saml.authenticationManager}"/>
  <property name="authenticationSuccessHandler" ref="samlSuccessRedirectHandler"/>
</bean>

<!-- Logout handler terminating local session -->
<bean id="samlLogoutHandler"
  class="org.springframework.security.web.authentication.logout.SecurityContextLogoutHandler">
  <property name="invalidateHttpSession" value="${security.saml.logout.invalidateHttpSession}"/>
</bean>

<!-- Override default logout processing filter with the one processing SAML messages -->
<bean id="samlLogoutFilter" class="org.springframework.security.saml.SAMLLogoutFilter">
  <constructor-arg ref="samlSuccessLogoutHandler"/>
  <constructor-arg ref="samlLogoutHandler"/>
  <constructor-arg ref="samlLogoutHandler"/>
</bean>

<!-- Filter processing incoming logout messages -->
<!-- First argument determines URL user will be redirected to after successful global logout -->
<bean id="samlLogoutProcessingFilter"
class="org.springframework.security.saml.SAMLLogoutProcessingFilter">
  <constructor-arg ref="samlSuccessLogoutHandler"/>
  <constructor-arg ref="samlLogoutHandler"/>
</bean>

```

```

<!-- Class loading incoming SAML messages from httpRequest stream -->
<bean id="processor" class="org.springframework.security.saml.processor.SAMLProcessorImpl">
  <constructor-arg>
    <list>
      <ref bean="redirectBinding"/>
      <ref bean="postBinding"/>
      <ref bean="artifactBinding"/>
      <ref bean="soapBinding"/>
      <ref bean="paosBinding"/>
    </list>
  </constructor-arg>
</bean>

<!-- Logger for SAML messages and events -->
<bean id="samlLogger" class="org.springframework.security.saml.log.SAMLDefaultLogger"/>

<!-- Central storage of cryptographic keys -->
<bean id="keyManager" class="org.springframework.security.saml.key.JKSKeyManager">
  <constructor-arg value="{security.saml.keystore}"/>
  <constructor-arg type="java.lang.String" value="{security.saml.keystore.password}"/>
  <constructor-arg>
    <map>
      <entry key="{security.saml.keystore.key.alias}" value="{security.saml.keystore.key.password}"/>
    </map>
  </constructor-arg>
  <constructor-arg type="java.lang.String" value="{security.saml.keystore.defaultKey}"/>
</bean>

<!-- SAML 2.0 Assertion Consumer -->
<bean id="webSSOprofileConsumer"
class="org.springframework.security.saml.websso.WebSSOProfileConsumerImpl"/>

<!-- SAML 2.0 Web SSO profile -->
<bean id="webSSOprofile" class="org.springframework.security.saml.websso.WebSSOProfileImpl"/>

<!-- SAML 2.0 ECP profile -->
<bean id="ecpprofile" class="org.springframework.security.saml.websso.WebSSOProfileECImpl"/>

<!-- SAML 2.0 Logout Profile -->
<bean id="logoutprofile" class="org.springframework.security.saml.websso.SingleLogoutProfileImpl"/>

<!-- Bindings, encoders and decoders used for creating and parsing messages -->
<bean id="postBinding" class="org.springframework.security.saml.processor.HTTPPostBinding">
  <constructor-arg ref="parserPool"/>
  <constructor-arg ref="velocityEngine"/>
</bean>

<bean id="redirectBinding"
class="org.springframework.security.saml.processor.HTTPRedirectDeflateBinding">
  <constructor-arg ref="parserPool"/>
</bean>

<bean id="artifactBinding" class="org.springframework.security.saml.processor.HTTPArtifactBinding">
  <constructor-arg ref="parserPool"/>
  <constructor-arg ref="velocityEngine"/>

```

```

<constructor-arg>
  <bean class="org.springframework.security.saml.websso.ArtifactResolutionProfileImpl">
    <constructor-arg>
      <bean class="org.apache.commons.httpclient.HttpClient"/>
    </constructor-arg>
    <property name="processor">
      <bean id="soapProcessor"
class="org.springframework.security.saml.processor.SAMLProcessorImpl">
        <constructor-arg ref="soapBinding"/>
      </bean>
    </property>
  </bean>
</constructor-arg>
</bean>

<bean id="soapBinding" class="org.springframework.security.saml.processor.HTTPSOAP11Binding">
  <constructor-arg ref="parserPool"/>
</bean>

<bean id="paosBinding" class="org.springframework.security.saml.processor.HTTPPAOS11Binding">
  <constructor-arg ref="parserPool"/>
</bean>

<!-- Initialization of OpenSAML library-->
<bean class="org.springframework.security.saml.SAMLBootstrap"/>

<!-- Initialization of the velocity engine -->
<bean id="velocityEngine" class="org.springframework.security.saml.util.VelocityFactory"
  factory-method="getEngine"/>

<!-- XML parser pool needed for OpenSAML parsing -->
<!-- <bean id="parserPool" class="org.opensaml.xml.parse.StaticBasicParserPool" scope="singleton" init-
method="initialize" /> -->
<bean id="parserPool" class="org.opensaml.xml.parse.BasicParserPool" scope="singleton"/>

<!-- IDP Metadata configuration - paths to metadata of IDPs in circle of trust is here -->
<!-- Do no forget to call initalize method on providers -->
<bean id="metadata" class="org.springframework.security.saml.metadata.CachingMetadataManager">
  <constructor-arg>
    <null/>
  </constructor-arg>
  <property name="providers">
    <list>
      <bean class="org.opensaml.saml2.metadata.provider.HTTPMetadataProvider">
        <!-- URL containing the metadata -->
        <constructor-arg>
          <value type="java.lang.String">${security.saml.idp.metadataURL}</value>
        </constructor-arg>
        <!-- Timeout for metadata loading in ms -->
        <constructor-arg>
          <value type="int">5000</value>
        </constructor-arg>
        <property name="parserPool" ref="parserPool"/>
      </bean>
      <bean class="org.springframework.security.saml.metadata.ExtendedMetadataDelegate">
        <constructor-arg>

```

```

<bean class="org.springframework.security.saml.metadata.MetadataMemoryProvider">
  <constructor-arg>
    <bean factory-bean="metadataGenerator" factory-method="generateMetadata"/>
  </constructor-arg>
</bean>
</constructor-arg>
<constructor-arg>
  <bean class="org.springframework.security.saml.metadata.ExtendedMetadata">
    <property name="local" value="true"/>
    <property name="alias" value="{security.saml.sp.alias}"/>
    <property name="securityProfile" value="metaiop"/>
    <property name="signingKey" value="{security.saml.sp.signingKeyAlias}"/>
    <property name="encryptionKey" value="{security.saml.sp.encryptionKeyAlias}"/>
    <property name="tlsKey" value="{security.saml.sp.tlsKey}"/>
    <property name="requireArtifactResolveSigned" value="true"/>
    <property name="requireLogoutRequestSigned" value="true"/>
    <property name="requireLogoutResponseSigned" value="true"/>
  </bean>
</constructor-arg>
</bean>
</list>
</property>
</bean>

<bean class="org.springframework.security.saml.metadata.MetadataGenerator"
  id="metadataGenerator">
  <property name="entityId" value="{security.saml.sp.id}"/>
  <property name="entityBaseURL" value="{security.saml.sp.url}"/>
  <property name="entityAlias" value="{security.saml.sp.alias}"/>
  <property name="signingKey" value="{security.saml.sp.signingKeyAlias}"/>
  <property name="encryptionKey" value="{security.saml.sp.encryptionKeyAlias}"/>
  <property name="tlsKey" value="{security.saml.sp.tlsKey}"/>
  <property name="requestSigned" value="true"/>
  <property name="signMetadata" value="true"/>
  <property name="wantAssertionSigned" value="true"/>
  <property name="includeDiscovery" value="true"/>
</bean>
</beans>

```

- 4) Import “applicationContext-spring-security-saml.xml” into your spring context by either configuring it in web.xml (see below) or importing it from another spring configuration file.

```

<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>
    WEB-INF/applicationContext-security-saml.xml
  </param-value>
</context-param>

```

- 5) Add to your spring security configuration file the following filters as children of your existing <http> tag.

```

<security:custom-filter before="PRE_AUTH_FILTER" ref="samlMetadataFilter"/>
<security:custom-filter position="PRE_AUTH_FILTER" ref="samlEntryPoint"/>
<security:custom-filter after="BASIC_AUTH_FILTER" ref="samlProcessingFilter"/>

```

Which should result in for example:

```
<http>
  <intercept-url pattern="/pages/**" access="IS_AUTHENTICATED_FULLY" requires-channel="https"/>
  <form-login login-page="/" login-processing-url="/login" always-use-default-target="true"/>
  <logout logout-url="/logout"/>
  <security:custom-filter before="PRE_AUTH_FILTER" ref="samlMetadataFilter"/>
  <security:custom-filter position="PRE_AUTH_FILTER" ref="samlEntryPoint"/>
  <security:custom-filter after="BASIC_AUTH_FILTER" ref="samlProcessingFilter"/>
</http>
```

Add to your spring security file the saml authentication provider

```
<authentication-provider ref="samlAuthenticationProvider"/>
```

as child of an <authentication-manager> tag where possibly other authentication provider might be configured for other means of authentication i.e.

```
<authentication-manager alias="authenticationManager">
  <authentication-provider user-service-ref="userService">
    <password-encoder ref="passwordEncoder"/>
  </authentication-provider>
  <authentication-provider ref="samlAuthenticationProvider"/>
</authentication-manager>
```

6) Generate a new public/private key, used for signing and encryption. Open your operation system console and execute following command (replacing the words in capitals):

```
keytool -genkeypair -alias YOUR_SERVICE_ALIAS -keystore samlKeystore.jks -storepass changeit -validity 3650 -keyalg RSA -keysize 2048
```

```
First last name=YOUR_SERVICE_ALIAS
Organization unit=Primary
Organisation=YOUR_COMPANY
City=YOUR_CITY
State=YOUR_STATE
Country code=YOUR_COUNTRY_CODE
```

Yes

enter

Move this keystore to your project and configure it in the file “security.properties” as explained in 5.1.2.3.

### 5.1.2.3 SAML Security Properties Explained

```
security.saml.successRedirectHandler.defaultTargetUrl=
```

When SAML authentication is successful, the client will be redirected to the above URL.

```
security.saml.successLogoutHandler.defaultTargetUrl=
```

When logout is successful, the client will be redirected to the above URL.

```
security.saml.entryPoint.filterProcessesUrl=/saml/login
security.saml.entryPoint.binding=urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST
security.saml.idp.metadataURL=http://ciam-dev-chic.custodix.com/idp/shibboleth
security.saml.userService=<Implementation of your user details service>
```



When an end-user surfs to the path configured as “security.saml.entryPoint.filterProcessesUrl” (i.e. by being redirected, or by clicking on a button/link) SAML authentication is triggered. The SAML entry point will create a SAML authentication request and send it to the configured IDP through the configured SAML Binding

```
security.saml.authenticationManager=authenticationManager
```

Bean ID of the spring security's authentication manager to be used. Unless you have given the authentication manager bean another id, this should not be changed.

```
security.saml.logout.invalidateHttpSession=true
```

Whether the http session should be invalidated after logout

```
security.saml.sp.alias=CHIC
```

```
security.saml.sp.id=chic
```

```
security.saml.metadata.filterSuffix=/saml/metadata
```

```
security.saml.sp.url=YOUR_SP_URL
```

Alias, ID, URL and metadata suffix of the Service Provider. The metadata URL is actually configured as a suffix as it is a relative URL (relative towards the service provider's URL). This metadata URL can be configured on the IdP (see IdP configuration for more explanation), so that the IdP knows where to fetch the Service Provider's metadata.

```
security.saml.sp.signingKeyAlias=chic_key
```

```
security.saml.sp.tlsKey= chic_key
```

```
security.saml.sp.encryptionKeyAlias= chic_key
```

```
security.saml.keystore.defaultKey= chic_key
```

```
security.saml.keystore=file\:/<pathToKeystore>
```

```
security.saml.keystore.password=changeit
```

```
security.saml.keystore.key.alias= chic_key
```

```
security.saml.keystore.key.password=changeit
```

Above properties should contain the aliases, store path and passwords of the Service Provider's signing, encryption, tls and default key. Only one key can be configured with the current sample spring configuration. To be able to assign different aliases to the different key types, the spring configuration should be updated!

## 5.1.3 PHP Based Service Providers through simplesamlphp

### 5.1.3.1 Preconditions

- A web server capable of executing PHP scripts (e.g. Apache)
- PHP version >= 5.2.0.
- Support for the following PHP extensions:
  - Always required: date, dom, hash, libxml, openssl, pcre, SPL, zlib, mcrypt
- When using database:
  - Always: PDO
  - Database driver: (mysql, pgsql, ...)



- Definitions:
  - Example requesting user:
    - username: “elias-chic-test”
    - password: “El1a\$Ner1”
  - IdP Metadata: <https://ciam-dev-chic.custodix.com/idp/shibboleth> (further referred to as “YOUR\_IDP\_METADATA\_URL”)
  - Example web site (service provider) to be integrated: <https://sp-chic-dev.custodix.com>

### 5.1.3.2 Assumptions

Paths can defer on other systems

- /var/www a private folder not publicly accessible where scripts can be installed
- /var/www/html the public web folder

### 5.1.3.3 Install simplesamlphp

- 1) Download the latest version of simplesamlphp from <http://code.google.com/p/simplesamlphp/downloads/list>
- 2) Untar and move contents to /var/www/simplesamlphp
- 3) Move/Copy the contents of /var/www/simplesamlphp/www to /var/www/html/simplesaml
  - a) Configure Apache
  - i) Edit /etc/httpd/conf.d/virtualhosts.conf

```
<Directory "/var/www/html/simplesaml" >
    Order Deny,Allow
    Allow from all
</Directory>
<VirtualHost *>
    ServerName sp-chic-dev.custodix.com
    DocumentRoot "/var/www/html"
    Alias /simplesaml "/var/www/html/simplesaml"
</VirtualHost>
```

- ii) Replace sp-chic-dev.custodix.com with your own hostname.

### 5.1.3.4 Configure the service provider

- 7) Create a self-signed certificate & key in var/www/simplesamlphp/cert/

```
openssl req -newkey rsa:2048 -new -x509 -days 3652 -nodes -out saml.crt -keyout saml.pem
```

- 8) Replace /var/www/simplesamlphp/config/authsources.php

```
<?php
$config = array(
    // This is a authentication source which handles admin authentication.
    //Disable once configuration successful
    'admin' => array(
        'core:AdminPassword',
    ),
    'chicDevSP' => array(
        'saml:SP',
        'NameIDPolicy' => NULL,
```

```
'entityID' => 'https://sp-chic-dev.custodix.com',
'idp' => 'https://ciam-chic-dev.custodix.com/idp/shibboleth',
'discoURL' => NULL,
// Private and public key of this SP
'privatekey' => 'saml.pem',
'certificate' => 'saml.crt',
),
);
```

Replace sp-chic-dev.custodix.com with your own hostname.

#### 9) Replace /var/www/simplesamlphp/config/config.php

```
<?php
$config = array (
    'baseurlpath'      => 'https://sp-chic-dev.custodix.com/simplesaml/',
    'certdir'          => 'cert/',
    'loggingdir'        => 'log/',
    'datadir'           => 'data/',
    'tempdir'           => '/tmp/simplesaml',
    'debug'             => FALSE,
    'showerrors'        => TRUE,
    'debug.validatexml' => FALSE,
    'auth.adminpassword' => 'YxDRKuk0IORL',
    'admin.protectindexpage' => false,
    'admin.protectmetadata' => false,
    'secretsalt'        => '6co4emiwgai745zzfcsz8oiqvr8ra0ys',
    'technicalcontact_name' => 'Administrator',
    'technicalcontact_email' => 'elias.neri@custodix.com',
    'timezone'          => NULL,
    'logging.level'      => SimpleSAML_Logger::NOTICE,
    'logging.handler'     => 'file',
    'logging.facility'    => defined('LOG_LOCAL5') ? constant('LOG_LOCAL5') : LOG_USER,
    'logging.processname' => 'simplesamlphp',
    'logging.logfile'     => 'simplesamlphp.log',
    'statistics.out'     => array(
    ),
    'enable.shib13-idp'   => false,
    'enable.adfs-idp'     => false,
    'enable.wsfed-sp'     => false,
    'enable.authmemcookie' => false,
    'session.duration'    => 8 * (60*60), // 8 hours.
    'session.datastore.timeout' => (4*60*60), // 4 hours
    'session.state.timeout' => (60*60), // 1 hour
    'session.cookie.name' => 'SimpleSAMLSessionID',
    'session.cookie.lifetime' => 0,
    'session.cookie.path' => '/',
    'session.cookie.domain' => NULL,
    'session.cookie.secure' => FALSE,
    'session.disable_fallback' => FALSE,
    'enable.http_post'    => FALSE,
    'session.phpsession.cookie.name' => null,
    'session.phpsession.savepath' => null,
    'session.phpsession.httponly' => FALSE,
    'session.authtoken.cookie.name' => 'SimpleSAMLAuthToken',
    'language.available'  => array('en', 'no', 'nn', 'se', 'da', 'de', 'sv', 'fi', 'es', 'fr', 'it', 'nl', 'lb',
    'cs', 'sl', 'lt', 'hr', 'hu', 'pl', 'pt', 'pt-br', 'tr', 'ja', 'zh', 'zh-tw', 'ru', 'et', 'he', 'id', 'sr', 'lv'),
```

```
'language.rtl' => array('ar','dv','fa','ur','he'),
'language.default' => 'en',
'language.parameter.name' => 'language',
'language.parameter.setcookie' => TRUE,
'language.cookie.name' => 'language',
'language.cookie.domain' => NULL,
'language.cookie.path' => '/',
'language.cookie.lifetime' => (60*60*24*900),
'attributes.extradictionary' => NULL,
'theme.use' => 'default',

'default-wsfed-idp' => 'urn:federation:pingfederate:localhost',
'idpdisco.enableremember' => TRUE,
'idpdisco.rememberchecked' => TRUE,
'idpdisco.validate' => TRUE,
'idpdisco.extDiscoveryStorage' => NULL,
'idpdisco.layout' => 'dropdown',
'shib13.signresponse' => TRUE,
'authproc.idp' => array(
    30 => 'core:LanguageAdaptor',
    45 => array(
        'class' => 'core:StatisticsWithAttribute',
        'attributename' => 'realm',
        'type' => 'saml20-idp-SSO',
    ),
    50 => 'core:AttributeLimit',
    99 => 'core:LanguageAdaptor',
),
'authproc.sp' => array(
    60 => array('class' => 'core:GenerateGroups', 'eduPersonAffiliation'),
    61 => array('class' => 'core:AttributeAdd', 'groups' => array('users', 'members')),
    90 => 'core:LanguageAdaptor',
),
'metadata.sources' => array(
    array('type' => 'flatfile'),
),
'store.type' => 'sql',
'store.sql.dsn' => 'mysql:host=localhost;dbname=saml',
'store.sql.username' => 'php',
'store.sql.password' => 'php',
'store.sql.prefix' => 'simpleSAMLphp',

'metadata.sign.enable' => FALSE,
'metadata.sign.privatekey' => NULL,
'metadata.sign.privatekey_pass' => NULL,
'metadata.sign.certificate' => NULL,

'proxy' => NULL,

'redirect.trustedsites' => NULL,
);
```

Update 'store.sql.dsn', 'store.sql.username', 'store.sql.password'.  
Please also update 'auth.adminpassword', 'secretsalt', 'technicalcontact\_name',  
'technicalcontact\_email'.

10) Update /var/www/simplesamlphp/metadata/saml20-idp-remote.php to configure the CHIC idp.

11) Update /var/www/html/simplesaml/\_include.php

a) Replace

```
require_once(dirname(dirname(__FILE__)) . '/lib/_autoload.php');
```

With

```
require_once('/var/www/simplesamlphp/lib/_autoload.php');
```

Note this depends on the path where simplesamlphp was installed

b) Replace

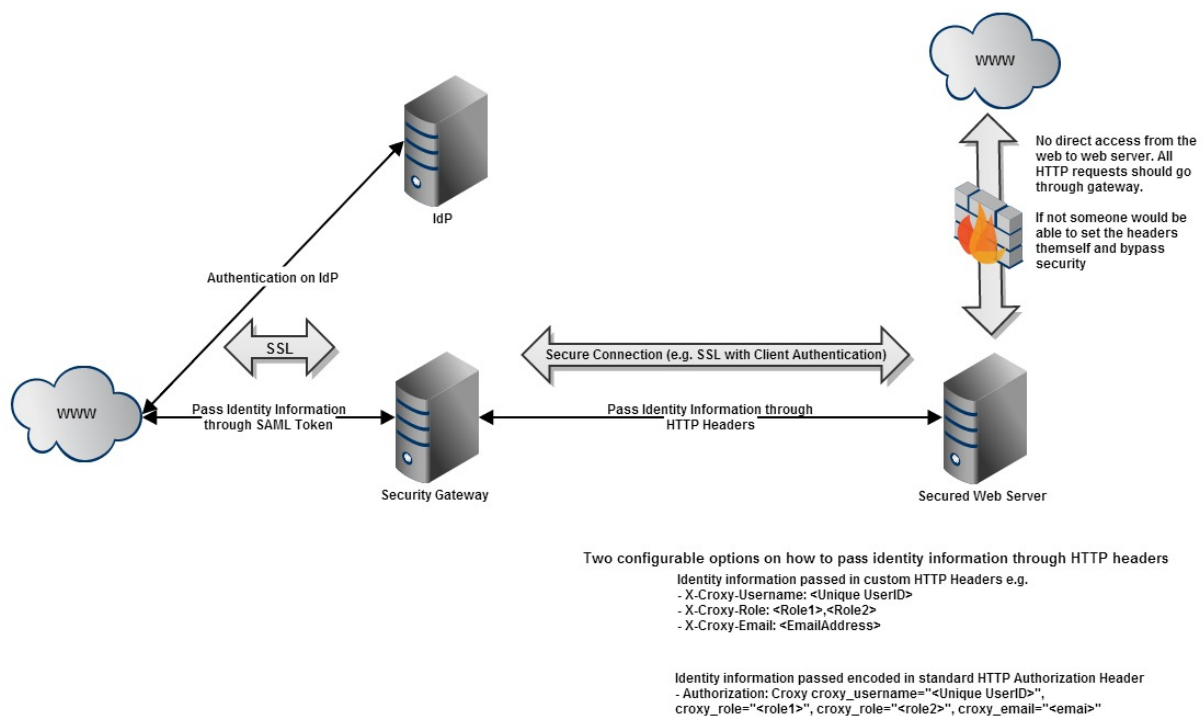
```
$configdir = dirname(dirname(__FILE__)) . '/config';
```

With

```
$configdir = '/var/www/simplesamlphp/config';
```

Note: this depends on the path where simplesamlphp was installed.

### 5.1.4 Integrate Service Providers through a Security Gateway



**Figure 12 Security Gateway Architecture**

For legacy applications or for applications where it is not feasible to integrate the SAML protocol, CHIC provides a Security Gateway. The security gateway is deployed as proxy in front of the web application to be integrated and it will handle all security (SAML) protocols. Upon successful authentication on the gateway, it will forward identity information to the backend application through HTTP headers<sup>5</sup>.

<sup>5</sup> Note that for this to be secure it is important that the backend application is shielded from the outside world and can only be accessed through the security gateway. Otherwise HTTP headers can easily be faked and injected in HTTP requests resulting in security breaches.

## 5.1.5 Liferay Portal Integration

As portal technology CHIC has chosen Liferay. Therefore this section has been included which explains how Liferay can be integrated in CHIC and make use of the CHIC security services.

### 5.1.5.1 Preconditions

*Jdk1.6.0\_45*

Liferay 6.1.1 uses the last version of Java6 JDK<sup>6</sup>. It is not Java7 compatible. Be sure that there is a system variable called JAVA\_HOME which refers to the jdk1.6.0\_45 folder, and add %JAVA\_HOME%\bin to the PATH system variable (this might slightly defer from the operating system used).

*liferay-portal-6.1.1-ce-ga2*

You can download a clean Liferay portal<sup>7</sup> together with a preconfigured tomcat (see next requirement).

*tomcat-7.0.27*

You can download a Liferay portal<sup>8</sup> together with a preconfigured tomcat. The tomcat-7.0.27 directory will be called **your\_tomcat** from now on in this manual.

*liferay-plugins-sdk-6.1.1-ce-ga2*

Download from here<sup>9</sup> the liferay-plugins-sdk-6.1.1-ce-ga2. This will from now on in this manual be called **your\_liferay\_plugins\_sdk**.

*apache-ant-1.8.4*

Have a system variable called ANT\_HOME which refers to your ant folder and add %ANT\_HOME%\bin to your PATH system variable (this might slightly defer from the operating system used).

### 5.1.5.2 Configure Liferay

Before running the tomcat server or deploying the hook/extension, it is advised to remove any existing directories in **your\_tomcat/webapps** except for the ROOT directory. These folders contain plugins that come bundled together with the liferay portal version. They could slow down the server.

Also make a backup of the your\_tomcat folder, so if anything goes wrong you can start from scratch again.

### 5.1.5.3 Creating a Liferay Portal

Start the tomcat server from your\_tomcat/bin directory:

```
startup.bat
```

<sup>6</sup> <http://www.oracle.com/technetwork/java/javase/downloads/java-archive-downloads-javase6-419409.html>

<sup>7</sup> <http://sourceforge.net/projects/lportal/files/Liferay%20Portal/6.1.1%20GA2/liferay-portal-tomcat-6.1.1-ce-ga2-20120731132656558.zip/download>

<sup>8</sup> <http://sourceforge.net/projects/lportal/files/Liferay%20Portal/6.1.1%20GA2/liferay-portal-tomcat-6.1.1-ce-ga2-20120731132656558.zip/download>

<sup>9</sup> <http://sourceforge.net/projects/lportal/files/Liferay%20Portal/6.1.1%20GA2/liferay-plugins-sdk-6.1.1-ce-ga2-20120731132656558.zip/download>

If everything is correctly configured the Liferay Basic Configuration page should open when the server has fully started. On this page you can register an administrator with liferay and also choose your database setup. Follow through with the administration registration. Please write down the information as you will be required to log in to enable the security extension options.

Now stop the tomcat server.

#### 5.1.5.4 Configure the Liferay Plugins sdk

Create your own build.properties file in **your\_liferay\_plugins\_sdk**, and name it "build.YOUR\_USERNAME.properties". The properties that you set in this file will overwrite the properties in the original "build.properties" file.

The "build.YOUR\_USERNAME.properties" file should only contain the following lines:

```
app.server.type = tomcat
app.server.dir = your_tomcat
app.server.lib.global.dir = ${app.server.dir}\\lib\\ext
app.server.deploy.dir = ${app.server.dir}\\webapps
app.server.portal.dir = ${app.server.deploy.dir}\\ROOT
```

Download the CHIC security extension from the CHIC wiki, and unzip the file. Place the samlAuthentication-ext folder in **your\_liferay\_plugins\_sdk/ext** (**your\_liferay\_plugins\_sdk/ext/samlAuthentication-ext** now named **samlAuthentication\_ext**) folder.

Download the CHIC security hook from the CHIC wiki, and unzip the file. Place the samlAuthentication-hook folder in the **your\_liferay\_plugins\_sdk/hooks** (**your\_liferay\_plugins\_sdk/hooks/samlAuthentication-hook** now named **samlAuthentication\_hook**) folder.

#### 5.1.5.5 Create a keystore with key

To create a keystore with key, execute the following command in the place you want to create the keystore. The fields marked with xxx can be filled in as you wish. Keytool [25] is provided together with any standard JDK and is found in the jdk.xxx/bin folder, which should (this should already be available through your PATH system variable).

```
keytool -genkey -dname "cn=xxx, ou=xxx, o=xxx, c=xxx" -alias YOUR_KEY_ALIAS -keypass YOUR_PASSWORD -
keystore ./YOUR_KEYSTORE_FILE.jks -storepass YOUR_PASSWORD -keyalg RSA -keysize 2048 -validity
KEY_VALIDITY_IN_DAYS
```

Place the generated keystore file "YOUR\_KEYSTORE\_FILE.jks" inside **samlAuthentication\_ext/docroot/WEB-INF/ext-impl/src/resources** (now named **extension\_resources**).

#### 5.1.5.6 Configure the extension

To enable the SAML authentication you first need to download the shibboleth metadata xml file from the CHIC IdP (<https://ciam-dev-chic.custodix.com/idp/shibboleth>). Save this file as an .xml file (and name it for example "IDP\_SHIBBOLETHMETADATA.xml"). Now place it in the folder **extension\_resources** and configure the "security.properties" file located in **samlAuthentication\_ext/docroot/WEB-INF/ext-web/docroot/WEB-INF** accordingly (further named **extension\_webinf**).

```
security.saml.sp.alias=YOUR_SERVICE_PROVIDERS_ALIAS
security.saml.sp.id=YOUR_SERVICE_PROVIDERS_ID
security.saml.sp.url=YOUR_LIFERAY_URL
security.saml.idp.metadataURL=classpath\:/resources/IDP_SHIBBOLETHMETADATA.xml
security.saml.keystore=classpath\:/resources/YOUR_KEYSTORE_FILE.jks
security.saml.keystore.password=YOUR_PASSWORD
```

```
security.saml.keystore.defaultKey=YOUR_KEY_ALIAS
security.saml.keystore.key.alias=YOUR_KEY_ALIAS
security.saml.keystore.key.password=YOUR_PASSWORD
```

Replace `security.saml.sp.alias/idp/url` with your own values. When the website is running, these values will be used to generate a metadata file that will have to be provided to the IdP in order to allow authentication through the IdP. The values for the keystore are the ones that you used for the creation of the keystore.

#### 5.1.5.7 Deploy the extension and hook

Go to **samlAuthentication\_ext** and deploy the extension:

```
ant deploy
```

You should find a 'BUILD SUCCESSFUL' message. Start the tomcat server again (`startup.bat`). The extension will be deployed on the server. Finally you will be prompted to reboot the server. Stop the tomcat server again.

Now go into **samlAuthentication\_hook** and recompile and deploy the `samlAuthentication-hook`:

```
ant clean
ant deploy
```

You should find a "BUILD SUCCESSFUL" message. Start the tomcat server again (`startup.bat`). The hook will be deployed on the server. If the hook is deployment is successful, you should find: 'Hook for `samlAuthentication-hook` is available for use'.

The `samlAuthentication-hook` is hot deployable, which means any time the 'ant deploy' command has been executed in **samlAuthentication\_hook** the server will automatically redeploy the hook if the server is still running. Changes should be visible the next time the user refreshes the window.

*Beware: every time the extension (i.e. **samlAuthentication-ext**) has been changed, your tomcat should be removed and replaced by a clean installation of tomcat (a version that has no deployed extension). The deployment of an extension changes the contents of the tomcat server irreversibly.*

#### 5.1.5.8 Check if the security extension is correctly deployed

Surf to `YOUR_LIFERAY_URL` and sign in with the information you used to register as an admin, in Liferay's own sign in form. If everything is correctly installed, you should find the SAML configuration options in the control panel as described below. You can enable SAML, and the link to the authentication form for SSO. But users will not be able to sign in through the IdP before you have provided the IdP with the site's metadata.

#### 5.1.5.9 Provide the IdP with your metadata

Make sure that the server is running and surf to **your\_liferay\_url/saml/metadata**, save the document and give it a name, like 'YOUR\_SHIBBOLETHMETADATA.xml'. This file needs to be sent to `elias.neri@custodix.com`.

#### 5.1.5.10 Configuration for the portal

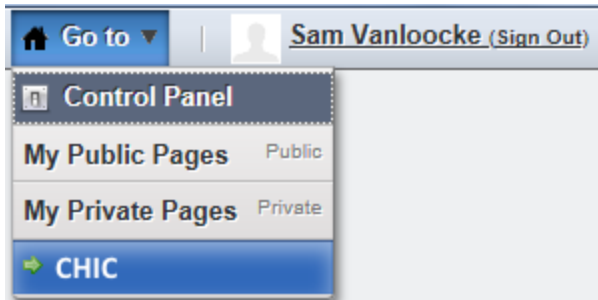
It is possible to modify the default values of configuration of the security extension. As soon as these values are modified and saved in the control panel of the website, these new values are stored in the database, and take precedence over the defaults. They are discussed below.

#### 5.1.5.11 Working with the web interface

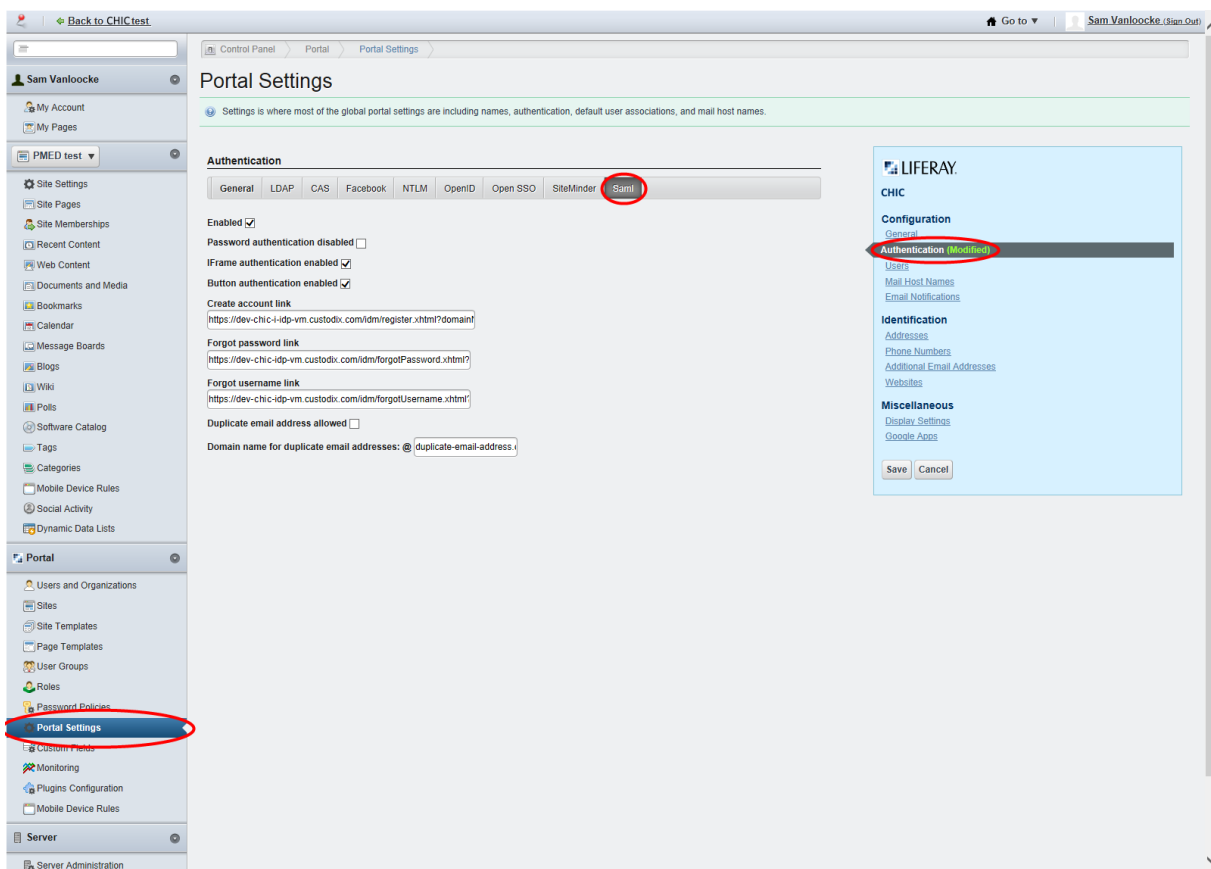
The addition of the security extension and hook opens up new options for users to safely and easily log in using a SSO service provided by a running Custodix IdP instance. These options are editable for an administrator with the necessary rights in the web portal.

### 5.1.5.11.1 Security extension enabled configurations

To access the settings that have been added by the security extension, first log in as an admin.



The picture above shows how to access the Liferay control panel. In the upper right corner of the screen, find the *Go to* tab and press *Control Panel*.



As the picture above shows, first click *Portal Settings* on the left, then *Authentication* on the right, and at last *Saml* at the top of the page. Here you find the new options that the extension enables. The fields are associated with properties (field property) found in the "portal.properties" file in `samlAuthentication_hook\docroot\WEB-INF\src`. The text (text property) associated with each fields is configurable, and is found in the "Language-en.properties" file in `samlAuthentication_hook\docroot\WEB-INF\src\content`.

field property	text property	default text
	saml	Saml



saml.auth.enabled	not available	Enabled
password.auth.disabled	passwordAuthDisabled	Password authentication disabled
button.authentication.enabled	buttonAuthenticationEnabled	Button authentication enabled
iFrame.authentication.enabled	iFrameAuthenticationEnabled	IFrame authentication enabled
allow.duplicate.emailaddress.registration	allowDuplicateEmailaddressRegistration	Duplicate email address allowed
email.domain.duplicates	emailDomainDuplicates	Domain name for duplicate email addresses: @
create.account.link	createAccountLink	Create account link
forgot.password.link	forgotPasswordLink	Forgot password link
forgot.userName.link	forgotUserNameLink	Forgot username link
	chicSSO	Login with CHIC account
	forgot-username	Forgot Username
	chic-login-link	Login with CHIC

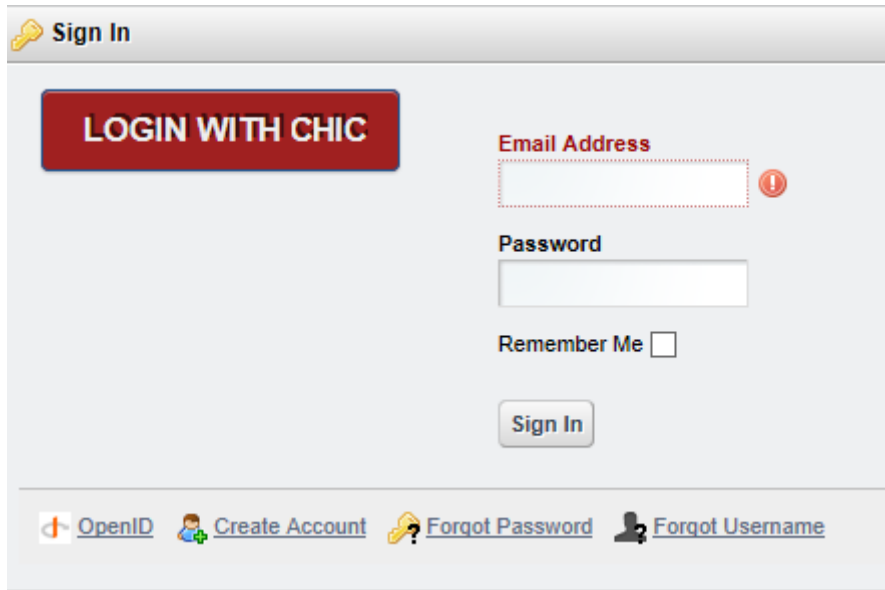
The admin is free to change the default values of the field properties as he desires. The field descriptions are found below:

field property	description
----------------	-------------

saml.auth.enabled	Enables/disables the CHIC security plugin
password.auth.disabled	Enables SAML authentication as a mode to log in
button.authentication.enabled	Provides a link to the authentication form.
iFrame.authentication.enabled	Provides a direct frame to the authentication form allowing the user to stay on the page when authenticating, this is disabled due to browser incompatibility
allow.duplicate.emailaddress.registration	Allows users to register with an email address that already exist in the database.
email.domain.duplicates	When duplicate email addresses are allowed, the user that tries to register with an already registered address gets appointed a new email address with the value as domain name.
create.account.link	The address a user is directed to when pressing the 'create account' link in the login page.
forgot.password.link	The address a user is directed to when pressing the 'forgot password' link in the login page.
forgot.userName.link	The address a user is directed to when pressing the 'forgot username' link in the login page.

#### 5.1.5.11.2 The sign in page

The security extension mainly focuses on the ways existing users log in and new users register. When visiting the website, any user that has not already been logged in through the IdP, will be able to sign in from the Liferay portal. The sign in page with password authentication not disabled and button authentication enabled (see above), should look something like this:



On the left the button "login with CHIC" links to a form in the domain where users can log in through a SSO service. This is enabled by checking 'Button authentication enabled' in the portal settings for SAML.

On the right Liferay's own login form can be used to gain access the site. However users logging in using this option are not authenticated by the IdP, and thus do not enjoy the SSO service advantages. This is enabled by unchecking 'Password authentication disabled' in the portal settings for SAML.

On the bottom the 'Create Account' link leads to the page defined under 'Create account link' in the portal settings for SAML.

The 'Forgot password' link leads to the page defined under 'Forgot password link' in the portal settings for SAML.

The 'Forgot username' link leads to the page defined under 'Forgot username link' in the portal settings for SAML.

## 5.2 Active Authentication

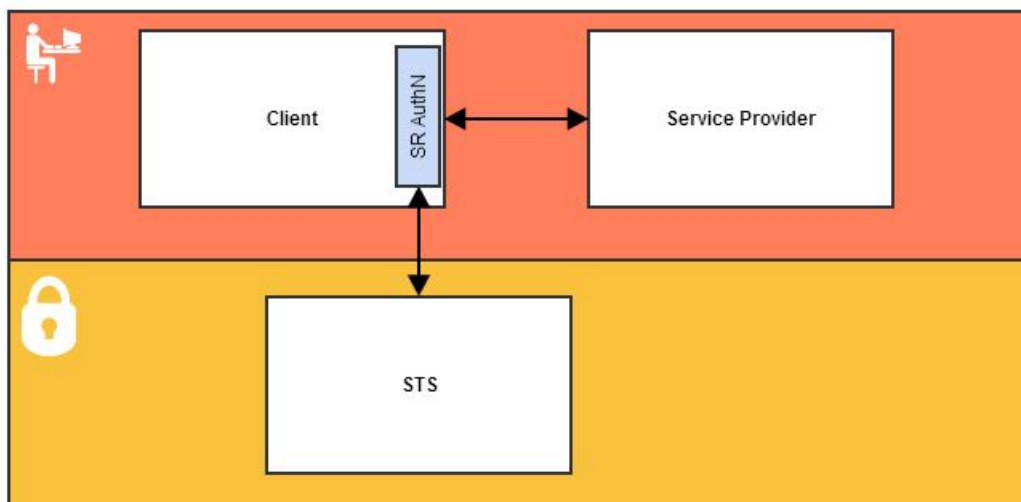


Figure 13 Activate Authentication Profile

To call a web service a client needs to obtain a SAML token. The Client obtains a SAML token by sending an authentication request and a security token request to the CHIC STS. If Authentication is

successful, the STS returns a security token. This token contains embedded the SAML Token that identifies the authenticated user. This SAML token should then be passed through the web service call to the service provider.

## 5.2.1 XML Based REST or SOAP Client

### 5.2.1.1 Preconditions and Assumptions

- Definitions:
  - Example requesting user:
    - username: "elias-chic-test"
    - password: "El1a\$Ner1"
  - STS: <https://ciam-dev-chic.custodix.com/sts/services/STS>
  - Example web site (service provider) to be integrated: <https://service-chic-test.custodix.com>

### 5.2.1.2 Tutorial

#### 5.2.1.2.1 Step 1: Call and authenticate on STS to fetch a SAML authentication token

Call the STS by sending the following XML (containing authentication credentials and detail of the kind of security token requested). All text in bold should be updated for each request.

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <Action xmlns="http://www.w3.org/2005/08/addressing">http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/Issue</Action>
    <MessageID xmlns="http://www.w3.org/2005/08/addressing">urn:uuid:566fed28-bb9c-467e-8bfc-e25f729d4fb0</MessageID>
    <To xmlns="http://www.w3.org/2005/08/addressing">https://ciam-dev-chic.custodix.com/sts/services/STS</To>
    <ReplyTo xmlns="http://www.w3.org/2005/08/addressing">
      <Address>http://www.w3.org/2005/08/addressing/anonymous</Address>
    </ReplyTo>
    <wsse:Security soap:mustUnderstand="1" xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
      <wsse:UsernameToken xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" wsu:Id="UsernameToken-1">
        <wsse:Username>elias-chic-test</wsse:Username>
        <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#PasswordText">El1a$Ner1</wsse:Password>
      </wsse:UsernameToken>
    </wsse:Security>
  </soap:Header>
  <soap:Body>
    <wst:RequestSecurityToken xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-trust/200512">
      <wst:RequestType>http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue</wst:RequestType>
      <wsp:AppliesTo xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
        <wsa:EndpointReference xmlns:wsa="http://www.w3.org/2005/08/addressing">
          <wsa:Address>https://service-chic-test.custodix.com</wsa:Address>
        </wsa:EndpointReference>
      </wsp:AppliesTo>
      <wst:TokenType>urn:oasis:names:tc:SAML:2.0:assertion</wst:TokenType>
      <wst:KeyType>http://docs.oasis-open.org/ws-sx/ws-trust/200512/Bearer</wst:KeyType>
      <wst:Renewing/>
    </wst:RequestSecurityToken>
  </soap:Body>
</soap:Envelope>
```

```
</wst:RequestSecurityToken>
</soap:Body>
</soap:Envelope>
```

If Authentication successful the STS will return something in the line of (embedded saml token in bold "<saml2:Assertion/>"):

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <Action xmlns="http://www.w3.org/2005/08/addressing">http://docs.oasis-open.org/ws-sx/ws-
trust/200512/RSTRC/IssueFinal</Action>
    <MessageID xmlns="http://www.w3.org/2005/08/addressing">urn:uuid:7a93d5ec-5857-4171-adba-
c59bbf57713e</MessageID>
    <To
xmlns="http://www.w3.org/2005/08/addressing">http://www.w3.org/2005/08/addressing/anonymous</To>
    <RelatesTo xmlns="http://www.w3.org/2005/08/addressing">https://ciam-dev-
chic.custodix.com/sts/services/STS</RelatesTo>
  </soap:Header>
  <soap:Body>
    <ns2:RequestSecurityTokenResponseCollection xmlns="http://docs.oasis-open.org/ws-sx/ws-trust/200802"
xmlns:ns2="http://docs.oasis-open.org/ws-sx/ws-trust/200512"
xmlns:ns3="http://www.w3.org/2005/08/addressing" xmlns:ns4="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" xmlns:ns5="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
      <ns2:RequestSecurityTokenResponse>
        <ns2:TokenType>urn:oasis:names:tc:SAML:2.0:assertion</ns2:TokenType>
        <ns2:RequestedSecurityToken>
          <saml2:Assertion ID="_AD4B1C889ACDD06EB913842633353465" IssueInstant="2013-11-
12T13:35:35.346Z" Version="2.0" xsi:type="saml2:AssertionType"
xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
            <saml2:Issuer>custodix-sts</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/2000/09/xmldsig#rsa-sha1" />
                <ds:Reference URI="#_AD4B1C889ACDD06EB913842633353465">
                  <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#" />
                    <ec:InclusiveNamespaces PrefixList="xs" xmlns:ec="http://www.w3.org/2001/10/xml-exc-
c14n#" />
                  </ds:Transform>
                </ds:Transforms>
                <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
                <ds:DigestValue>zDXX8/TEcbUs/mnQZ1oFDuPTlo=</ds:DigestValue>
              </ds:Reference>
            </ds:SignedInfo>

            <ds:SignatureValue>Y0T9sfAPDO8mrWQNsS0uvnew0bMdFcyiCGc9KB6xDIVoQYxjqTNkjR/blqHtgSeVsVwW
h4otF8YyV06L5O/DrZoDmGeUgeQShQQxnr9+Ms1ydcz6xUCEDsdYIHQtTuNePWncLMLL0InwCeiEhcYXA0XPL
EMITW0//GIUm8Ce42mmC9w41evUJeY0oGpCj4++/77E8mcTmKCi0UKIoM4FenFMYHffHPCmF3DkqUuv1KiC
MWBHUNrx2dZTajWotxhMy5PnrDLUQ7a6P4yCsbDcbr/gxiGmxINR+/1hoqFb8b3Gxzm03rCOd5QYQRGcl3ae
ezPsxZAYKrZINsvpp7A==</ds:SignatureValue>
          <ds:KeyInfo>
            <ds:X509Data>
```

```
<ds:X509Certificate>MIIDMTCCAhmGAWIBAgIESSX1wTANBgqhkiG9w0BAQsFADBJMQswCQYDVQQGEwJCR
TERMA8GA1UE
ChMIQ3VzdG9kaXgxDTALBgNVBAsTBENJQU0xGDAWBgNVBAMTD3N0cy1zaWduaW5nLWtleTAeFw0x
MzEwMjMwOTE0MDIaFw0yMzEwMjEwOTE0MDIaMEkxCzAJBgNVBAYTAkFMRwDwYDVQQKEwhDdXN0
b2RpeDENMAAsGA1UECxEQ0IBTTEYMBYGA1UEAxMPc3RzLXNpZ25pbmcta2V5MIIBIjANBgqhkiG
9w0BAQEFAAOCAQ8AMIIBCgKCAQEAscAcfKzbUKyWr1Ggw+KOr7NUhA/WMIbU8ft3Iy18ZoVyLG6M
IVLuF3op8xt2HwbaN8KIDJDzEufXWKZ1a/oLYGK3Fa0V4XpaAqX9Pz5MNEU4hJJw+s4rqbybve1
c/DCAF4CR2kQKZMp6G2Yot3SkPCYx10uifjLT749V6buDnDM/YelVPOg210qpTcimhmf7L8o2JSk
Jx9Z6pRCSaji9Nsd1jvi09x5xEDXcJu90OW1V6LBv7CXRb0g6WmXPY4BBWR55r5tPlwrnoGvBO1R
38G40Vg4QeYKW7jF2WaF4j17AgGkNFI664PckS25vx/i7ulidfWAjIHYrgJfKGPUJwIDAQABoyEw
HzAdBgNVHQ4EFgQUUQVOQ7dNRWRiaPzg1s0CbZd3M6fQwDQYJKoZIhvcNAQELBQADggEBADL1zXdf
NMnk57Q2xi04p3VKGecKky8VV/MxUFH/kQf174hD+SrDvpTFQw0NvcZxs++q7NqhwZfMpu1+fRkb
kUqm9ImreZVTNqvRuoL8s4DK2w8q19k65ocYIWYRuWVMgvjY9bWu9dbdU8sXGqP9odywPcl3pWf
g7j+BcUIECIIMphSRAfpq//ZQMbmobBrQpIhL2xdP5Pg0jAcgsSw3/xoNnC/q89wiuULKpBhteDK
UroeuA79DcRgT10K7GQfgNs5hbdgK+DxFT08uel+RnSfzVZq+ktG9upraxF+E1WWB72MSERUB1r1
2IDNyvoPOxsHA8FaNnk88CYx7/eOHds=</ds:X509Certificate>
</ds:X509Data>
</ds:KeyInfo>
</ds:Signature>
<saml2:Subject>
<saml2:NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent"
NameQualifier="custodix-sts">e15369ef-e6cb-472c-83d8-1d6396d3c952</saml2:NameID>
<saml2:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer"/>
</saml2:Subject>
<saml2:Conditions NotBefore="2013-11-12T13:35:35.346Z" NotOnOrAfter="2013-11-
12T14:05:35.346Z">
<saml2:AudienceRestriction>
<saml2:Audience>https://service-chic-test.custodix.com</saml2:Audience>
</saml2:AudienceRestriction>
</saml2:Conditions>
<saml2:AttributeStatement>
<saml2:Attribute FriendlyName="mail" Name="urn:oid:0.9.2342.19200300.100.1.3"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri">
<saml2:AttributeValue xsi:type="xs:string">elias@custodix.com</saml2:AttributeValue>
</saml2:Attribute>
<saml2:Attribute FriendlyName="mail" Name="urn:custodix:ciam:1.0:principal:email"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri">
<saml2:AttributeValue xsi:type="xs:string">elias@custodix.com</saml2:AttributeValue>
</saml2:Attribute>
<saml2:Attribute FriendlyName="sn" Name="urn:oid:2.5.4.4"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri">
<saml2:AttributeValue xsi:type="xs:string">Test</saml2:AttributeValue>
</saml2:Attribute>
<saml2:Attribute FriendlyName="domainUuid" Name="urn:custodix:ciam:1.0:domain:uuid"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri">
<saml2:AttributeValue xsi:type="xs:string">74a05798-2af9-4e03-ad24-
2159c2fc95f8</saml2:AttributeValue>
</saml2:Attribute>
<saml2:Attribute FriendlyName="domainName" Name="urn:custodix:ciam:1.0:domain:name"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri">
<saml2:AttributeValue xsi:type="xs:string">CHIC-Dev</saml2:AttributeValue>
</saml2:Attribute>
<saml2:Attribute FriendlyName="uuid" Name="urn:oid:0.9.2342.19200300.100.1.1"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri">
```

```

<saml2:AttributeValue xsi:type="xs:string">e15369ef-e6cb-472c-83d8-
1d6396d3c952</saml2:AttributeValue>
</saml2:Attribute>
<saml2:Attribute FriendlyName="uuid" Name="urn:custodix:ciam:1.0:principal:uuid"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri">
<saml2:AttributeValue xsi:type="xs:string">e15369ef-e6cb-472c-83d8-
1d6396d3c952</saml2:AttributeValue>
</saml2:Attribute>
<saml2:Attribute FriendlyName="givenName" Name="urn:oid:2.5.4.42"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri">
<saml2:AttributeValue xsi:type="xs:string">Elias</saml2:AttributeValue>
</saml2:Attribute>
<saml2:Attribute FriendlyName="displayName" Name="urn:oid:2.16.840.1.113730.3.1.241"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri">
<saml2:AttributeValue xsi:type="xs:string">Elias</saml2:AttributeValue>
</saml2:Attribute>
</saml2:AttributeStatement>
</saml2:Assertion>
</ns2:RequestedSecurityToken>
<ns2:RequestedAttachedReference>
<ns5:SecurityTokenReference wsse11:TokenType="http://docs.oasis-open.org/wss/oasis-wss-saml-
token-profile-1.1#SAMLV2.0" xmlns:wsse11="http://docs.oasis-open.org/wss/oasis-wss-wssecurity-secext-
1.1.xsd">
<ns5:KeyIdentifier ValueType="http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-
1.1#SAMLID">_AD4B1C889ACDD06EB913842633353465</ns5:KeyIdentifier>
</ns5:SecurityTokenReference>
</ns2:RequestedAttachedReference>
<ns2:RequestedUnattachedReference>
<ns5:SecurityTokenReference wsse11:TokenType="http://docs.oasis-open.org/wss/oasis-wss-saml-
token-profile-1.1#SAMLV2.0" xmlns:wsse11="http://docs.oasis-open.org/wss/oasis-wss-wssecurity-secext-
1.1.xsd">
<ns5:KeyIdentifier ValueType="http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-
1.1#SAMLID">_AD4B1C889ACDD06EB913842633353465</ns5:KeyIdentifier>
</ns5:SecurityTokenReference>
</ns2:RequestedUnattachedReference>
<wsp:AppliesTo xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-trust/200512">
<wsa:EndpointReference xmlns:wsa="http://www.w3.org/2005/08/addressing">
<wsa:Address>https://service-chic-test.custodix.com</wsa:Address>
</wsa:EndpointReference>
</wsp:AppliesTo>
<ns2:Lifetime>
<ns4:Created>2013-11-12T13:35:35.384Z</ns4:Created>
<ns4:Expires>2013-11-12T14:05:35.384Z</ns4:Expires>
</ns2:Lifetime>
</ns2:RequestSecurityTokenResponse>
</ns2:RequestSecurityTokenResponseCollection>
</soap:Body>
</soap:Envelope>

```

The SAML token should next be extracted by the client and passed together with the SOAP or REST web service call to the web service as authentication token.

If Authentication fails the STS will return:

```

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Header>

```



```
<Action xmlns="http://www.w3.org/2005/08/addressing">http://docs.oasis-open.org/ws-sx/ws-trust/200512/STS/Issue/Fault/WSSecurityException</Action>
  <MessageID xmlns="http://www.w3.org/2005/08/addressing">urn:uuid:67d98c49-4d66-43ad-8be0-f6792ad3ab5b</MessageID>
  <To xmlns="http://www.w3.org/2005/08/addressing">http://www.w3.org/2005/08/addressing/anonymous</To>
  <RelatesTo xmlns="http://www.w3.org/2005/08/addressing">urn:uuid:566fed28-bb9c-467e-8bfc-e25f729d4fb0</RelatesTo>
</soap:Header>
<soap:Body>
  <soap:Fault>
    <faultcode>soap:Server</faultcode>
    <faultstring>The security token could not be authenticated or authorized</faultstring>
  </soap:Fault>
</soap:Body>
</soap:Envelope>
```

#### 5.2.1.2.2 Step 2: Use the obtained SAML token to call a secured REST web service

Through the previous step, the below SAML token was retrieved that identifies the user “Elias Neri”.

```
<saml2:Assertion ID=" AD4B1C889ACDD06EB91384263353465" IssueInstant="2013-11-12T13:35:35.346Z" Version="2.0" xsi:type="saml2:AssertionType"
xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<saml2:Issuer>custodix-sts</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/2000/09/xmldsig#rsa-sha1"/>
<ds:Reference URI="#_AD4B1C889ACDD06EB91384263353465">
<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#">
<ec:InclusiveNamespaces PrefixList="xs"
xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#"/>
</ds:Transform>
</ds:Transforms>
<ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
<ds:DigestValue>zddXx8/TEcbUs/mnQZlOFDuPTlo=</ds:DigestValue>
</ds:Reference>
</ds:SignedInfo>
<ds:SignatureValue>YOT9sfAPDO8mrWQNssS0uvnew0bmDFcyyiCGc9KB6xDlVoQYxjqTNkjr/blqHtgSeVsVwWh4otF8YyV06L5O/DrZoDmGeUgeQShQQxnrt9+Ms1ydcz6xUCEDsdYlHQttTuNePwncLMLL01NwCeiehcYXAoxPLEMTW0/GIUm8Ce42mmC9w41evUJeY0oGpcJ4++/77E8mcTmKciOUKIoM4FenFMYHffHPcmf3DkquUv1KiCMWBHUNrx2dZTajWotxhmY5PnrDLUQ7a6P4yCsbBdcbr/gxiGmxINR+/lhofqF8b3Gxzmo3rCod5QYQRGci3aezPsxZayKrZINSvpp7A==</ds:SignatureValue>
<ds:KeyInfo>
<ds:X509Data>
<ds:X509Certificate>MIIDMTCCAhmgAwIBAgIESXlwTANBgkqhkiG9w0BAQsFADBJMjswcQYDVQQGEwJCRTETMA8GA1UEChMIQ3VzdG9kaXgxdTALEGBAsTBENJATU0xGDAWBGNVBAMTD3N0cy1zaWduaW5nLWtleTaEfW0xmZEwmjMwOTEOMDlaFw0yMEZwMjEwOTEOMDlaMEKxCzAJBgNVBAYTAkJKFRMRewDwyDVQKEwhDdXN0b2RpeDENMASGA1UECxMEQ01BTTEYMBYGAlUEAxMPc3RzLXNPZ225pbmcta2V5MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAscACfKzbUKyWr1Ggw+Kor7NUhA/WmlBU8ft3Iy18ZoVyLG6MIVLuF3op8xtz2HwbAN8KIDJDzEuFXWKZ1a/oLYGK3FaOV4XpaAQX9Pz5MNEU4hJjW+s4rqbybvelc/DCAF2CR2kQKMp6G2Yot3SkPCYx10uifJLT749V6buDnDM/YelVPog210qpTcimhmf7L8o2JSkJx9Z6pRCsaji9Nsdljvi09x5xEDxcJu90OW1V6LBv7CXRB0g6WmXPY4BBWR5r5tPIwrnoGvBO1R38G40VG4QeYKW7jF2WaF4j17AgGKNFLG64PckS25vx/i7uIidfWAjiHYrgJfKGPUJwIDAQABoyEwHzAdBgNVHQ4EFggQUQVOQ7dNRWriaPzg1s0CbZd3M6fQwDQYJKoZIhvcNAQELBQADggEBADL1zXdFNmnk57Q2xi04p3VKGecKky8VV/MxUFH/kQfl74hd+SrDvpTFQw0NvcZxs++q7NqhWzfMpul+fkrbkUqm9lmreZVTNTqvRuOL8s4DK2w8q19k65ocYIWyrUwVMgvjYj9bwu9dbdU8sXGqP9odywPcl3pwfg7j+BcuIECl1MphSRafpq//ZQMbmobBrQpIhL2xdP5Pg0jAcgsSw3/xoNnC/q89wiuULKpBhteDKUroeuA79DCrgT1OK7GQfgNs5hbdgK+DxFt08ueI+RnsfzvZq+ktG9upraxF+E1WWB72MSERUB1r12LDnyvoPOxsSHA8FaNnk88CYx7/eOHds=</ds:X509Certificate>
</ds:X509Data>
</ds:KeyInfo>
```



```

</ds:Signature>
<saml2:Subject>
  <saml2:NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent"
NameQualifier="custodix-sts">e15369ef-e6cb-472c-83d8-1d6396d3c952</saml2:NameID>
  <saml2:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer"/>
</saml2:Subject>
  <saml2:Conditions NotBefore="2013-11-12T13:35:35.346Z" NotOnOrAfter="2013-11-
12T14:05:35.346Z">
    <saml2:AudienceRestriction>
      <saml2:Audience>https://service-chic-test.custodix.com</saml2:Audience>
    </saml2:AudienceRestriction>
  </saml2:Conditions>
  <saml2:AttributeStatement>
    <saml2:Attribute FriendlyName="mail" Name="urn:oid:0.9.2342.19200300.100.1.3"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri">
      <saml2:AttributeValue
xsi:type="xs:string">elias@custodix.com</saml2:AttributeValue>
    </saml2:Attribute>
    <saml2:Attribute FriendlyName="mail"
Name="urn:custodix:ciam:1.0:principal:email" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-
format:uri">
      <saml2:AttributeValue
xsi:type="xs:string">elias@custodix.com</saml2:AttributeValue>
    </saml2:Attribute>
    <saml2:Attribute FriendlyName="sn" Name="urn:oid:2.5.4.4"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri">
      <saml2:AttributeValue xsi:type="xs:string">Test</saml2:AttributeValue>
    </saml2:Attribute>
    <saml2:Attribute FriendlyName="domainUuid"
Name="urn:custodix:ciam:1.0:domain:uuid" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-
format:uri">
      <saml2:AttributeValue xsi:type="xs:string">74a05798-2af9-4e03-ad24-
2159c2fc95f8</saml2:AttributeValue>
    </saml2:Attribute>
    <saml2:Attribute FriendlyName="domainName"
Name="urn:custodix:ciam:1.0:domain:name" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-
format:uri">
      <saml2:AttributeValue xsi:type="xs:string">CHIC-
Dev</saml2:AttributeValue>
    </saml2:Attribute>
    <saml2:Attribute FriendlyName="uuid" Name="urn:oid:0.9.2342.19200300.100.1.1"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri">
      <saml2:AttributeValue xsi:type="xs:string">e15369ef-e6cb-472c-83d8-
1d6396d3c952</saml2:AttributeValue>
    </saml2:Attribute>
    <saml2:Attribute FriendlyName="uuid"
Name="urn:custodix:ciam:1.0:principal:uuid" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-
format:uri">
      <saml2:AttributeValue xsi:type="xs:string">e15369ef-e6cb-472c-83d8-
1d6396d3c952</saml2:AttributeValue>
    </saml2:Attribute>
    <saml2:Attribute FriendlyName="givenName" Name="urn:oid:2.5.4.42"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:uri">
      <saml2:AttributeValue xsi:type="xs:string">Elias</saml2:AttributeValue>
    </saml2:Attribute>
    <saml2:Attribute FriendlyName="displayName"
Name="urn:oid:2.16.840.1.113730.3.1.241" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-
format:uri">
      <saml2:AttributeValue xsi:type="xs:string">Elias</saml2:AttributeValue>
    </saml2:Attribute>
  </saml2:AttributeStatement>
</saml2:Assertion>

```

To call a given protected REST web service this SAML token should be GZIP and Base64 encoded (in that order) (as defined in chapter 3.1.2.2). This encoded SAML Token should be passed within the HTTP resource call to the backed service through as a HTTP Authorization header formatted as:

SAML auth=<Base64 Encoded SAML Token>

A file can for example be pasted through the following POST call to the CHIC development upload service:

<b>Method</b>	POST
<b>URL</b>	/services/rest/processingRequest/?name={name}
<b>Request</b>	<p>POST <a href="https://ttp-dev-chic.custodix.com/cats-ttp/services/rest/processingRequest/?name=data.csv">https://ttp-dev-chic.custodix.com/cats-ttp/services/rest/processingRequest/?name=data.csv</a> HTTP/1.1</p> <p>Accept-Encoding: gzip,deflate</p> <p>Content-Type: application/octet-stream</p> <p>Authorization: SAML auth=&lt;BASE 64 deflated saml token&gt;</p> <p>Content-Length: &lt;ContentLength&gt;</p> <p>Host: ttp-dev-chic.custodix.com</p> <p>Connection: Keep-Alive</p> <p>&lt;Data File Content&gt;</p>
<b>Response</b>	<p>HTTP/1.1 200 OK</p> <p>Server: nginx/1.2.2</p> <p>Date: &lt;Date&gt;</p> <p>Content-Type: application/octet-stream</p> <p>Content-Length: &lt;ContentLength&gt;</p> <p>Connection: keep-alive</p> <p>&lt;File ID&gt;</p>

## 5.2.2 Java Based Rest Client through Custodix Library

The Custodix STS Client Library makes use of CXF in the background.

### 5.2.2.1 Preconditions

- The Custodix STS Client Library and its dependencies should be available on the classpath.
  - STSClient-0.3.7.jar or STSClient-0.3.7.zip (with dependencies)
- Definitions:
  - Example requesting user:
    - username: "elias-chic-test"
    - password: "El1a\$Ner1"

- STS: <https://ciam-dev-chic.custodix.com/sts/services/STS>
- Example web site (service provider) to be integrated: <https://service-chic-test.custodix.com>

### 5.2.2.2 Tutorial

#### 5.2.2.2.1 Step 1: Call and authenticate on STS to fetch a SAML authentication token

```
import com.custodix.rest.security.RestSecurityUtils;
import com.custodix.sts.STSClient;
import org.apache.cxf.ws.security.tokenstore.SecurityToken;
import javax.security.auth.callback.CallbackHandler;
import org.apache.ws.security.WSPasswordCallback;
import javax.security.auth.callback.UnsupportedCallbackException;
import java.io.IOException;

STSClient client = new STSClient("elias-chic-test", new CallbackHandler() {
    @Override
    public void handle(Callback[] callbacks) throws IOException, UnsupportedCallbackException {
        WSPasswordCallback pc = (WSPasswordCallback) callbacks[0];
        pc.setPassword("Ella$Nerl");
    }
}, "https://ciam-dev-chic.custodix.com/sts/services/STS");
//Request a token to call service-chic-test.custodix.com
SecurityToken eliasToken = client.requestSecurityToken("https://service-chic-test.custodix.com");
//This token contains embedded the SAML assertion.
```

#### 5.2.2.2.2 Step 2: Use the obtained SAML token to call a secured REST web service

The API provided by Custodix provides a method that automatically converts a SAML token to an Authorization Header Value.

```
String authzHeaderValue = RestSecurityUtils.createHttpAuthzHeaderValue(eliasToken);
```

This value can then be passed directly as Authorization header to the REST call, e.g.

```
import org.apache.cxf.jaxrs.client.WebClient;

WebClient client = WebClient.create("https://service-chic-test.custodix.com");
client.path("services/rest/principal");
client.type("text/xml").accept("text/plain");
client.header("Authorization", authzHeader);

Response r = client.post("<user>...</user>");
```

## 5.2.3 Java CXF Based SOAP Client

### 5.2.3.1 Preconditions

- A Security STS Service is available and its WSDL is known.
  - <https://ciam-dev-chic.custodix.com/sts/services/STS>
  - <https://ciam-dev-chic.custodix.com/sts/services/STS?wsdl>
- A SOAP web service is available (WSDL url is known) and the web service is configured correctly according to.
- If the service, that you want to connect to, has a self-signed SSL (HTTPS) certificate (which is not encouraged), you have to add this certificate to your java trusted keystore<sup>10</sup> or use the CXF SSL support documentation<sup>11</sup>.

<sup>10</sup> <https://www.google.be/?q=add+certificate+to+java+cacerts>

<sup>11</sup> <http://cxf.apache.org/docs/client-http-transport-including-ssl-support.html>

- Be sure that the clock of your client and your server are synchronised. Bad synchronisation can cause token expiry exceptions on the server side.
- The default timeout value of a CXF client is 60 seconds, after this timeout an exception is thrown. If your client requires a long timeout range, change the ConnectionTimeout parameter in your client<sup>12</sup>.

### 5.2.3.2 Spring (Configuration) Based Tutorial

All library dependencies are available on the CHIC wiki.

1. First we generate from the wsdl of the web service the fully annotated Java code for this web service. For this, use the CXF wsdl2java executable (which can be found in the bin folder of the CXF package) in you OS console as described online<sup>13</sup>.

For example to generate this for the demo web services <https://demo-dev-chic.custodix.com/services/soap/helloWord?wsdl>

```
wsdl2java https://demo-dev-chic.custodix.com/services/soap/helloWorld?wsdl
```

2. Add the auto-generated code to the classpath of your project.
3. Although the wsdl2java executable generates most of the code for us, we need to bind one additional WS-Policy to our code (wsdl2java does not include this automatically). For this, add following file to your classpath (name it bindingPolicy.xml):

```
<?xml version="1.0" encoding="UTF-8" ?>
<wsp:Policy wsu:Id="SoapBindingPolicy"
xmlns:wsp="http://www.w3.org/ns/ws-policy"
xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702"
xmlns:t="http://docs.oasis-open.org/ws-sx/ws-trust/200512"
xmlns:wsx="http://schemas.xmlsoap.org/ws/2004/09/mex"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
xmlns:wsaw="http://www.w3.org/2005/08/addressing">
  <wsp:ExactlyOne>
    <wsp:All>
      <wsam:Addressing wsp:Optional="false">
        <wsp:Policy />
      </wsam:Addressing>
      <sp:TransportBinding>
        <wsp:Policy>
          <sp:TransportToken>
            <wsp:Policy>
              <sp:HttpsToken RequireClientCertificate="false">
                <wsp:Policy />
              </sp:HttpsToken>
            </wsp:Policy>
          </sp:TransportToken>
          <sp:AlgorithmSuite>
            <wsp:Policy>
              <sp:Basic128 />
            </wsp:Policy>
          </sp:AlgorithmSuite>
          <sp:Layout>
            <wsp:Policy>
```

<sup>12</sup> <http://cxf.apache.org/docs/client-http-transport-including-ssl-support.html>

<sup>13</sup> <http://cxf.apache.org/docs/wsdl-to-java.html>

```

        <sp:Lax />
    </wsp:Policy>
</sp:Layout>
<sp:IncludeTimestamp />
</wsp:Policy>
</sp:TransportBinding>
<sp:SignedSupportingTokens>
    <wsp:Policy>
        <sp:IssuedToken
            sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-
securitypolicy/200702/IncludeToken/AlwaysToRecipient">
            <sp:Issuer>
                <wsaw:Address>https://ciam-dev-
chic.custodix.com/sts/services/STS?wsdl</wsaw:Address>
            </sp:Issuer>
            <sp:RequestSecurityTokenTemplate>
                <t:TokenType>urn:oasis:names:tc:SAML:2.0:assertion</t:TokenType>
                <t:KeyType>http://docs.oasis-open.org/ws-sx/ws-trust/200512/Bearer</t:KeyType>
            </sp:RequestSecurityTokenTemplate>
        </wsp:Policy>
        <sp:RequireInternalReference />
    </wsp:Policy>
</sp:IssuedToken>
</wsp:Policy>
</sp:SignedSupportingTokens>
<sp:Wss11>
    <wsp:Policy>
        <sp:MustSupportRefIssuerSerial />
        <sp:MustSupportRefThumbprint />
        <sp:MustSupportRefEncryptedKey />
    </wsp:Policy>
</sp:Wss11>
<sp:Trust13>
    <wsp:Policy>
        <sp:MustSupportIssuedTokens />
        <sp:RequireClientEntropy />
        <sp:RequireServerEntropy />
    </wsp:Policy>
</sp:Trust13>
</wsp>All>
</wsp:ExactlyOne>
</wsp:Policy>

```

Include this policy in your web client by going to the web service interface (porttype) code generated by wsdl2java (where the @Webservice annotation is located) and place following annotation just beneath the @Webservice annotation.

```

@Policies({
    @Policy(uri = "POLICY_LOCATION", placement =
        Policy.Placement.BINDING)
})

```

Parameter	Description	Example
-----------	-------------	---------

POLICY_LOCATION	The place on the classpath where the bindingPolicy.xml can be found.	com/custodix/chic/demo/bindingPolicy.xml
-----------------	--	--

For example:

```
package com.custodix.chic.schemas;

import javax.xml.ws.WebMethod;
import javax.xml.ws.WebResult;
import javax.xml.ws.WebService;
import org.apache.cxf.annotations.Policies;
import org.apache.cxf.annotations.Policy;
import com.custodix.chic.demo;

/**
 * Endpoint interface for a HelloWorld Demo Soap Service
 *
 * @author Custodix, Kristof De Schepper
 */
@WebService
@Policies({
    @Policy(uri = "POLICY_LOCATION", placement =
        Policy.Placement.BINDING)
})
public interface IHELLOWORLDSERVICE {
    @WebMethod
    @WebResult(name="helloWorld")
    public String hello();
}
```

- Next we configure the client in a spring configuration, for this you generate a new spring context file and add it to your classpath. Add the following lines in the spring configuration (more information can be found in the comments):

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:http="http://cxf.apache.org/transports/http/configuration"
    xmlns:cxf="http://cxf.apache.org/core"
    xmlns:jaxws="http://cxf.apache.org/jaxws"
    xmlns:wsa="http://cxf.apache.org/ws/addressing"
    xmlns:wsp="http://cxf.apache.org/policy"
    xmlns:soap="http://cxf.apache.org/bindings/soap"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
        http://cxf.apache.org/transports/http/configuration
        http://cxf.apache.org/schemas/configuration/http-conf.xsd
        http://cxf.apache.org/core
        http://cxf.apache.org/schemas/core.xsd
        http://cxf.apache.org/jaxws
        http://cxf.apache.org/schemas/jaxws.xsd
        http://cxf.apache.org/ws/addressing
        http://cxf.apache.org/schemas/ws-addr-conf.xsd
```

```

http://cxf.apache.org/policy
http://cxf.apache.org/schemas/policy.xsd
http://cxf.apache.org/bindings/soap
http://cxf.apache.org/schemas/configuration/soap.xsd">

<!-- Jax-WS client, will make a stub for the service provider.-->
<jaxws:client id="YOUR_CLIENT_ID"
  serviceClass="YOUR_SERVICE_JAXWS_INTERFACE"
  address="YOUR_SERVICE_WSDL_LOCATION">
  <!-- only use this binding if the service provider requires SOAP version 1.2 -->
  <jaxws:binding>
    <soap:soapBinding version="1.2"/>
  </jaxws:binding>
  <!-- the service provider is bound to an STS -->
  <jaxws:properties>
    <entry key="ws-security.sts.client">
      <ref bean="sts-client"/>
    </entry>
  </jaxws:properties>
</jaxws:client>

<!-- STS client, will make a stub for the STS-->
<!-- Tokens will be cached till the saml token timestamp is expired (no extra calls to
the STS will be made till expiry-->
<!-- To disable caching of tokens, change the scope of the STS bean to prototype. Be aware
that this will create a call to the STS for each client operation-->
<!-- If you have more than one client in your project with the same STS, this only has to be included
once -->
<bean id="sts-client" class="org.apache.cxf.ws.security.trust.STSClient">
  <constructor-arg ref="cxf"/>
  <!-- disable renewal of tokens, due to bug in STS -->
  <property name="sendRenewing" value="false"/>
  <property name="allowRenewing" value="false"/>
  <property name="allowRenewingAfterExpiry" value="false"/>
  <!-- STS location -->
  <property name="wsdlLocation" value="YOUR_STS_WSDL_LOCATION"/>
  <property name="serviceName" value="YOUR_STS_SERVICE_NAME"/>
  <property name="endpointName" value="YOUR_STS_ENDPOINT_NAME"/>
  <!-- enter here the username/password for your principal-->
  <property name="properties">
    <map>
      <entry key="ws-security.username" value="YOUR_USERNAME"/>
      <entry key="ws-security.password" value="YOUR_PASSWORD"/>
    </map>
  </property>
</bean>

<!-- Bus configuration, this will enable WS-Policy, logging and WS-Addressing in each client -->
<!-- If you have more than one client in your project, this only has to be included once -->
<cxf:bus>
  <cxf:features>
    <wsp:policies enabled="true"/>
    <wsa:addressing/>
    <cxf:logging/>
  </cxf:features>
</cxf:bus>

```

```
<!-- disallow chunking on each https service -->
<!-- If you have more than one client in your project, this only has to be included once -->
<http:conduit name="https://.*">
  <http:client Connection="Keep-Alive" AllowChunking="false" />
</http:conduit>
</beans>
```

Replace following parameters in the above file.

Parameter	Description	Example
YOUR_CLIENT_ID	A unique id for your client, this will be the bean reference you use to call the service.	helloWorldWSClient
YOUR_SERVICE_JAXWS_INTERFACE	Classpath reference to the interface generated in step 1.	com.custodix.chic.schemas.IHELLOWORLDService
YOUR_SERVICE_WSDL_LOCATION	The wsdl location of the web service	https://demo-dev-chic.custodix.com/services/soap/helloWorld?wsdl
YOUR_STS_WSDL_LOCATION	The wsdl location of the STS	https://ciam-dev-chic.custodix.com/sts/services/STS?wsdl
YOUR_STS_SERVICE_NAME	The STS service name (including namespace)	{http://docs.oasis-open.org/ws-sx/ws-trust/200512/}SecurityTokenService
YOUR_STS_ENDPOINT_NAME	The STS endpoint name (including namespace)	{http://docs.oasis-open.org/ws-sx/ws-trust/200512/}CustodixSTS
YOUR_USERNAME	The username credential used for authenticating	
YOUR_PASSWORD	The password credential used for authenticating	

5) The above code should make the web service available in a client bean called YOUR\_CLIENT\_ID. Use this bean to call the web service.

### 5.2.3.3 Spring (Configuration) Based Tutorial

All library dependencies are available on the CHIC wiki.



1. First we generate from the wsdl of the web service the fully annotated Java code for this web service. For this, use the CXF wsdl2java executable (which can be found in the bin folder of the CXF package) in you OS console as described online<sup>14</sup>.

For example to generate this for the demo web services <https://demo-dev-chic.custodix.com/services/soap/helloWord?wsdl>

```
wsdl2java https://demo-dev-chic.custodix.com/services/soap/helloWorld?wsdl
```

2. Add the auto-generated code to the classpath of your project.
3. Although the wsdl2java executable generates most of the code for us, we need to bind one additional WS-Policy to our code (wsdl2java does not include this automatically). For this, add following file to your classpath (name it bindingPolicy.xml):

```
<?xml version="1.0" encoding="UTF-8" ?>
<wsp:Policy wsu:Id="SoapBindingPolicy"
xmlns:wsp="http://www.w3.org/ns/ws-policy"
xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702"
xmlns:t="http://docs.oasis-open.org/ws-sx/ws-trust/200512"
xmlns:wsx="http://schemas.xmlsoap.org/ws/2004/09/mex"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
xmlns:wsaw="http://www.w3.org/2005/08/addressing">
  <wsp:ExactlyOne>
    <wsp:All>
      <wsam:Addressing wsp:Optional="false">
        <wsp:Policy />
      </wsam:Addressing>
      <sp:TransportBinding>
        <wsp:Policy>
          <sp:TransportToken>
            <wsp:Policy>
              <sp:HttpsToken RequireClientCertificate="false">
                <wsp:Policy/>
              </sp:HttpsToken>
            </wsp:Policy>
          </sp:TransportToken>
          <sp:AlgorithmSuite>
            <wsp:Policy>
              <sp:Basic128 />
            </wsp:Policy>
          </sp:AlgorithmSuite>
          <sp:Layout>
            <wsp:Policy>
              <sp:Lax />
            </wsp:Policy>
          </sp:Layout>
          <sp:IncludeTimestamp />
        </wsp:Policy>
      </sp:TransportBinding>
      <sp:SignedSupportingTokens>
        <wsp:Policy>
          <sp:IssuedToken
```

<sup>14</sup> <http://cxf.apache.org/docs/wsdl-to-java.html>

```

        sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-
securitypolicy/200702/IncludeToken/AlwaysToRecipient">
        <sp:Issuer>
        <wsaw:Address>https://ciam-dev-
chic.custodix.com/sts/services/STS?wsdl</wsaw:Address>
        </sp:Issuer>
        <sp:RequestSecurityTokenTemplate>
        <t:TokenType>urn:oasis:names:tc:SAML:2.0:assertion</t:TokenType>
        <t:KeyType>http://docs.oasis-open.org/ws-sx/ws-trust/200512/Bearer</t:KeyType>
        </sp:RequestSecurityTokenTemplate>
        <wsp:Policy>
        <sp:RequireInternalReference />
        </wsp:Policy>
        </sp:IssuedToken>
        </wsp:Policy>
    </sp:SignedSupportingTokens>
    <sp:Wss11>
        <wsp:Policy>
        <sp:MustSupportRefIssuerSerial />
        <sp:MustSupportRefThumbprint />
        <sp:MustSupportRefEncryptedKey />
        </wsp:Policy>
    </sp:Wss11>
    <sp:Trust13>
        <wsp:Policy>
        <sp:MustSupportIssuedTokens />
        <sp:RequireClientEntropy />
        <sp:RequireServerEntropy />
        </wsp:Policy>
    </sp:Trust13>
    </wsp:All>
    </wsp:ExactlyOne>
</wsp:Policy>

```

Include this policy in your web client by going to the web service interface (porttype) code generated by wsdl2java (where the @Webservice annotation is located) and place following annotation just beneath the @Webservice annotation.

```

@Policies({
    @Policy(uri = "POLICY_LOCATION", placement =
        Policy.Placement.BINDING)
})

```

Parameter	Description	Example
POLICY_LOCATION	The place on the classpath where the bindingPolicy.xml can be found.	com/custodix/chic/demo/bindingPolicy.xml

For example:

```
package com.custodix.chic.schemas;
```

```
import javax.jws.WebMethod;
import javax.jws.WebResult;
import javax.jws.WebService;
import org.apache.cxf.annotations.Policies;
import org.apache.cxf.annotations.Policy;
import com.custodix.chic.demo;

/**
 * Endpoint interface for a HelloWorld Demo Soap Service
 *
 * @author Custodix, Kristof De Schepper
 */
@WebService
@Policies({
    @Policy(uri = "POLICY_LOCATION", placement =
        Policy.Placement.BINDING)
})
public interface IHELLOWORLDSERVICE {
    @WebMethod
    @WebResult(name="helloWorld")
    public String hello();
}
```

4. Next we configure the client in a code first approach, for this create a new java class and add it to your classpath. Add the following lines in the class (more information can be found in the comments):

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.apache.cxf.Bus;
import org.apache.cxf.BusFactory;
import org.apache.cxf.feature.AbstractFeature;
import org.apache.cxf.feature.LoggingFeature;
import org.apache.cxf.interceptor.LoggingInInterceptor;
import org.apache.cxf.jaxws.JaxWsProxyFactoryBean;
import org.apache.cxf.ws.addressing.WSAddressingFeature;
import org.apache.cxf.ws.policy.WSPolicyFeature;
import org.apache.cxf.ws.security.SecurityConstants;
import org.apache.cxf.ws.security.trust.STSClient;

/**
 * @author Custodix
 */
public class ClientTest{
    public static void main(String[] args){
        //Bus configuration, this will enable WS-Policy, logging and WS-Addressing in each client
        //If you have more than one client in your project, this only has to be included once
        Bus bus = BusFactory.getDefaultBus();
        List<AbstractFeature> features = new ArrayList<AbstractFeature>();
```

```

        WSPolicyFeature policyFeature = new WSPolicyFeature();
        policyFeature.setEnabled(true);
        features.add(new WSAddressingFeature());
        features.add(policyFeature);
        features.add(new LoggingFeature());
        for(AbstractFeature feature : features) {
            feature.initialize(bus);
        }

        //setup security token service
        STSClient stsClient = new STSClient(BusFactory.getDefaultBus());
        stsClient.setServiceName ("YOUR_STS_SERVICE_NAME");
        stsClient.setEndpointName("YOUR_STS_ENDPOINT_NAME");
        stsClient.setWsdLocation("YOUR_STS_WSDL_LOCATION");
        Map<String, Object> props = new HashMap<String, Object>();
        props.put("soap.no.validate.parts", "true");
        props.put("sendRenewing", "false");
        props.put("allowRenewing", "false");
        props.put("allowRenewingAfterExpiry", "false");
        props.put("ws-security.username", "YOUR_USERNAME");
        props.put("ws-security.password", "YOUR_PASSWORD");
        stsClient.setProperties(props);
        stsClient.getInInterceptors().add(new LoggingInInterceptor());
        stsClient.setFeatures(Collections.singletonList(new LoggingFeature()));

        //create a jaxws endpoint client
        JaxWsProxyFactoryBean factory = new JaxWsProxyFactoryBean();
        factory.setBus(bus);
        factory.setServiceClass(YOUR_SERVICE_JAXWS_INTERFACE);
        factory.setAddress("YOUR_SERVICE_WSDL_LOCATION");

        //set properties and features
        Map<String, Object> propss = new HashMap<String, Object>();
        propss.put(SecurityConstants.STS_CLIENT, stsClient);
        factory.setProperties(propss);
        factory.setFeatures(features);

        //create client
        YOUR_SERVICE_JAXWS_INTERFACE YOUR_CLIENT_ID = (YOUR_SERVICE_JAXWS_INTERFACE)
factory.create();
    }
}

```

Replace following parameters in the above file.

Parameter	Description	Example
YOUR_CLIENT_ID	A unique id for your client, this will be the bean reference you use to call the service.	helloWorldWSClient
YOUR_SERVICE_JAXW S_INTERFACE	Classpath reference to the interface generated in step 1.	com.custodix.chic.schemas.IHELLOW ORLDSERVICE

YOUR_SERVICE_WSDL_LOCATION	The wsdl location of the web service	<a href="https://demo-dev-chic.custodix.com/services/soap/helloWorld?wsdl">https://demo-dev-chic.custodix.com/services/soap/helloWorld?wsdl</a>
YOUR_STS_WSDL_LOCATION	The wsdl location of the STS	<a href="https://ciam-dev-chic.custodix.com/sts/services/STS?wsdl">https://ciam-dev-chic.custodix.com/sts/services/STS?wsdl</a>
YOUR_STS_SERVICE_NAME	The STS service name (including namespace)	{ <a href="http://docs.oasis-open.org/ws-sx/ws-trust/200512/">http://docs.oasis-open.org/ws-sx/ws-trust/200512/</a> }SecurityTokenService
YOUR_STS_ENDPOINT_NAME	The STS endpoint name (including namespace)	{ <a href="http://docs.oasis-open.org/ws-sx/ws-trust/200512/">http://docs.oasis-open.org/ws-sx/ws-trust/200512/</a> }CustodixSTS
YOUR_USERNAME	The username credential used for authenticating	
YOUR_PASSWORD	The password credential used for authenticating	

5) The above code should make the web service available in a client bean called YOUR\_CLIENT\_ID. Use this bean to call the web service.

## 5.2.4 Visual Studio Based C# SOAP Client

### 5.2.4.1 Preconditions

- A Security STS Service is available and its WSDL is known.
  - <https://ciam-dev-chic.custodix.com/sts/services/STS>
  - <https://ciam-dev-chic.custodix.com/sts/services/STS?wsdl>
- A SOAP web service is available (WSDL url is known) and the web service is configured correctly according to.
- If the service, that you want to connect to, has a self-signed SSL (HTTPS) certificate (which is not encouraged), you have to add this certificate to your windows certificate store<sup>15</sup>.
  - It is recommended that you make a program that automatically installs the certificates for the end user.
- Be sure that the clock of your client and your server are synchronised. Bad synchronisation can cause token expiry exceptions on the server side.
- The default timeout value of a .net client is 60 seconds, after this timeout an exception is thrown. If your client requires a long timeout range, change the timing timeout values as explained on msdn<sup>16</sup>.

<sup>15</sup> <http://windows.microsoft.com/en-us/windows/import-export-certificates-private-keys#1TC=windows-7>

<sup>16</sup> <http://msdn.microsoft.com/en-us/library/hh924831%28v=vs.110%29.aspx>

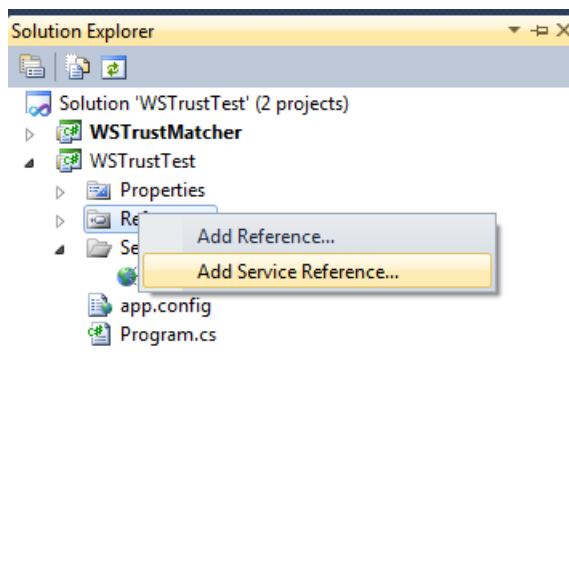
### 5.2.4.2 Tutorial

This tutorial is based on Visual Studio Express 2012 for windows desktop.

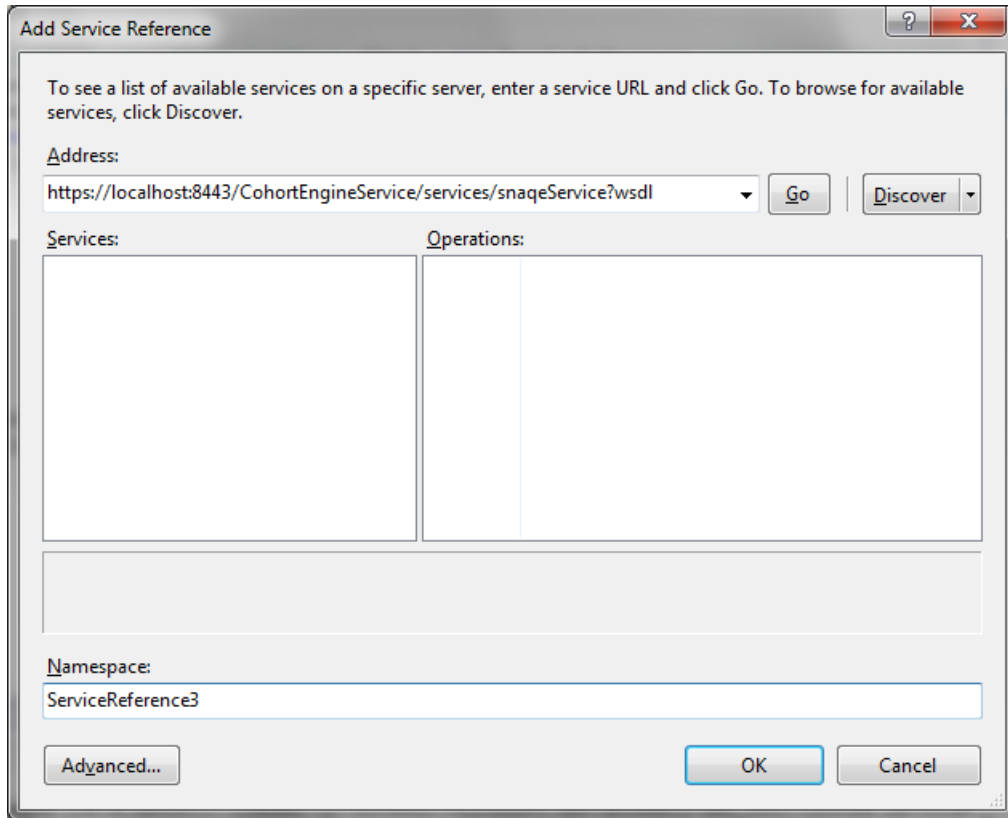
The following library dependencies are required (these are all standard supported in .NET 4.0):

- Microsoft.CSharp
- System
- System.Core
- System.Data
- System.Data.DataSetExtensions
- System.Runtime.Serialization
- System.ServiceModel
- System.Xml
- System.Xml.Linq

1. Start by adding a new web service reference to the client project (visual studio).



2. Enter the WSDL url of the web service we want to integrate



Visual Studio will create the needed service classes and an app.config containing the binding configuration.

- For some reason (bug?) Visual Studio does not auto-create a binding for the STS itself, so we have to include this manually in the app.config file. Following binding should be included in the customBinding tag of the app.config file.

```
<binding name="ChicSTS">
  <security authenticationMode="UserNameOverTransport" requireDerivedKeys="true"
    securityHeaderLayout="Strict" includeTimestamp="false"
    messageSecurityVersion="WSSecurity11WSTrust13WSSecureConversation13WSSecurityPolicy12Basic
    SecurityProfile10">
    <secureConversationBootstrap />
  </security>
  <textMessageEncoding messageVersion="Soap11WSAddressing10" />
  <httpsTransport />
</binding>
```

- Next, reference to the STS binding in the autogenerated issuer tag of the webservice binding.

```
<issuer address="https://ciam-dev-chic.custodix.com/sts/services/STS?wsdl"
  binding="customBinding" bindingConfiguration="ChicSTS"/>
```

- After this your app.config should look like this:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
  </startup>
  <system.serviceModel>
    <bindings>
```

```

<customBinding>
  <binding name="SNAQEServiceServiceSoapBinding">
    <security defaultAlgorithmSuite="Basic128"
authenticationMode="IssuedTokenOverTransport" enableUnsecuredResponse="true"
      requireDerivedKeys="true" securityHeaderLayout="Lax" includeTimestamp="true"

messageSecurityVersion="WSSecurity11WSTrust13WSSecureConversation13WSSecurityPolicy12Basic
SecurityProfile10">
      <issuedTokenParameters keyType="BearerKey"
tokenType="urn:oasis:names:tc:SAML:2.0:assertion">
        <additionalRequestParameters>
          <trust:SecondaryParameters xmlns:trust="http://docs.oasis-open.org/ws-sx/ws-
trust/200512">
            <t:TokenType xmlns:t="http://docs.oasis-open.org/ws-sx/ws-
trust/200512">urn:oasis:names:tc:SAML:2.0:assertion</t:TokenType>
            <t:KeyType xmlns:t="http://docs.oasis-open.org/ws-sx/ws-trust/200512">
http://docs.oasis-open.org/ws-sx/ws-trust/200512/Bearer</t:KeyType>
          </trust:SecondaryParameters>
        </additionalRequestParameters>
        <issuer address="https://ciam-dev-chic.custodix.com/sts/services/STS?wsdl"
binding="customBinding" bindingConfiguration="ChicSTS" />
        </issuedTokenParameters>
        <localClientSettings detectReplays="false" />
        <localServiceSettings detectReplays="false" />
      </security>
      <textMessageEncoding />
      <httpsTransport />
    </binding>
    <binding name="ChicSTS">
      <security authenticationMode="UserNameOverTransport" requireDerivedKeys="true"
securityHeaderLayout="Strict" includeTimestamp="false"

messageSecurityVersion="WSSecurity11WSTrust13WSSecureConversation13WSSecurityPolicy12Basic
SecurityProfile10" />
      <textMessageEncoding messageVersion="Soap11WSAddressing10" />
      <httpsTransport />
    </binding>
  </customBinding>
</bindings>
<client>
  <endpoint address="https://localhost:8443/CohortEngineService/services/snaqeService"
binding="customBinding" bindingConfiguration="SNAQEServiceServiceSoapBinding"
contract="ServiceReference3.ISNAQEService" name="snaqeServicePort" />
</client>
</system.serviceModel>
</configuration>

```

6. Finally you have to include username and password in order to authenticate to the STS. This can be done by including it in the client code itself (you can also include it in the configuration, but this is less flexible):

```

ServiceReference3.SNAQEServiceClient client3 = new ServiceReference3.SNAQEServiceClient();
client3.ClientCredentials.UserName.UserName = "USERNAME";
client3.ClientCredentials.UserName.Password = "PASSWORD";
ServiceReference3.snaqeResult result = client3.executeSNAQLScript(...);

```

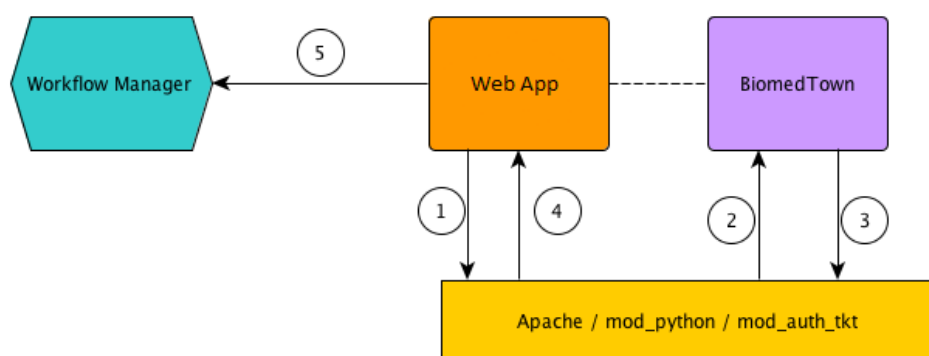


### 5.3 Integration of Security Framework with VPH-OP

VPH-OP is an existing hypermodelling infrastructure which will be the starting point for the implementation of the CHIC hypermodelling framework and which will be part of the CHIC architecture and be integrated with the security framework. This section aims to summarize the existing VPH-OP security implementation and how it will be integrated in the CHIC security components.

#### 5.3.1 VPH-OP Authentication

In VPH-OP, there is an authentication service, which provides the mechanisms to authenticate the user into the system. The VPH-OP authentication service features for accounting and granting permissions only to certain parts of the hypermodel. Figure 15 shows an example of the permission granting process. The services are implemented using the Biomed Town<sup>17</sup> OpenID Identity provider and the Apache mod\_auth\_tkt module<sup>18</sup>.



**Figure 14 VPH-OP Authentication Service Scheme**

The mechanism is based on a session ticket with an expiry time, which is passed to all services by the Workflow Manager and authorises the user to access the different components of the hypermodelling infrastructure.

If the user is recognised as a valid one by the OpenID Identity Provider, a valid ticket is generated by the authentication service; the ticket holds all the information about the user to prove its authenticity.

This is the authorisation workflow currently in place in VPH-OP:

- When a workflow is submitted:
  - the Workflow Manager securely stores the user's Session Ticket,
  - the Workflow Manager makes the Session Ticket available to all involved services.
- The Storage Service grants access to the user's resources only if:
  - the given Session Ticket is validated,
  - the requestor is involved into the workflow execution.
- Data is encrypted with a public-key cryptography algorithm.

<sup>17</sup> <http://www.biomedtown.org>

<sup>18</sup> <http://search.cpan.org/~gavinc/Apache-AuthTkt-0.08/AuthTkt.pm>

- All communication uses HTTPS protocol.

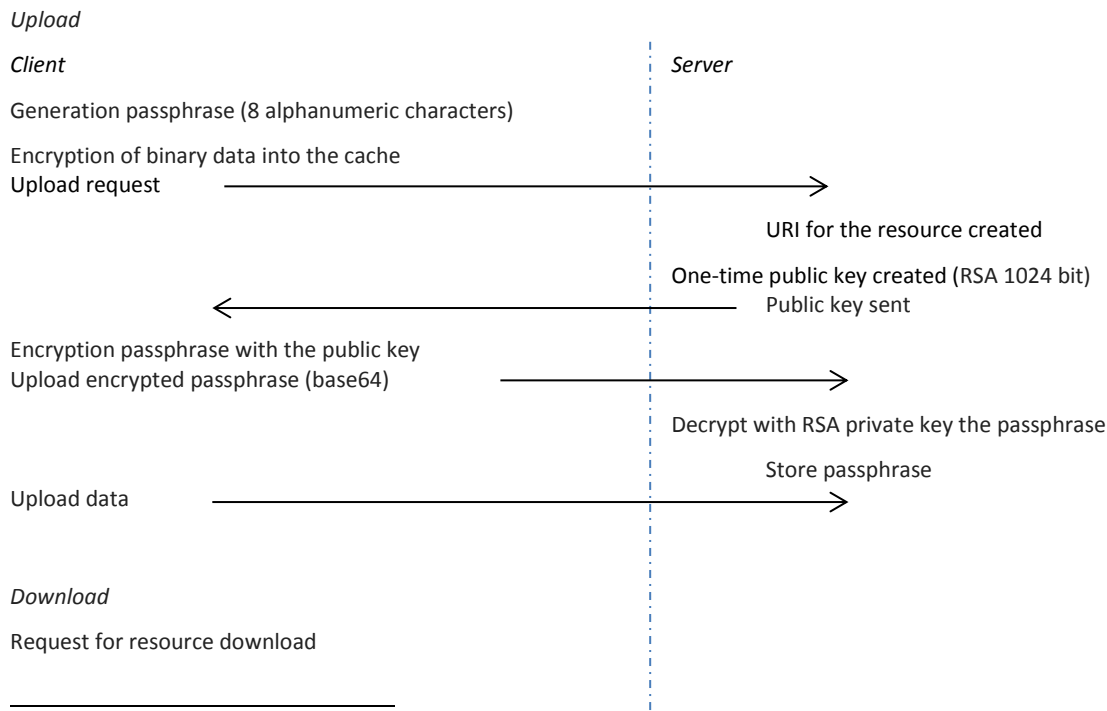
The concepts of the CHIC security framework, as explained in previous chapters, are quite similar to the VPH-OP concepts. CHIC though is based on different standards and token formats; this implies the following changes to be implemented for integration:

1. Apache mod\_auth\_tk tokens should be replaced with SAML tokens.
2. The OpenID BiomedTown Identity Provider should be replaced by a SAML Identity Provider and a SAML WS-Trust Security Token Service.

VPH-OP contains C++ implementation code, therefore C++ integration profiles of both the Passive and Active Authentication will be provided by Custodix during the VPH-OP integration. These will be made available on the CHIC wiki and in an updated revision of this deliverable.

### 5.3.2 VPH-OP Encryption Services

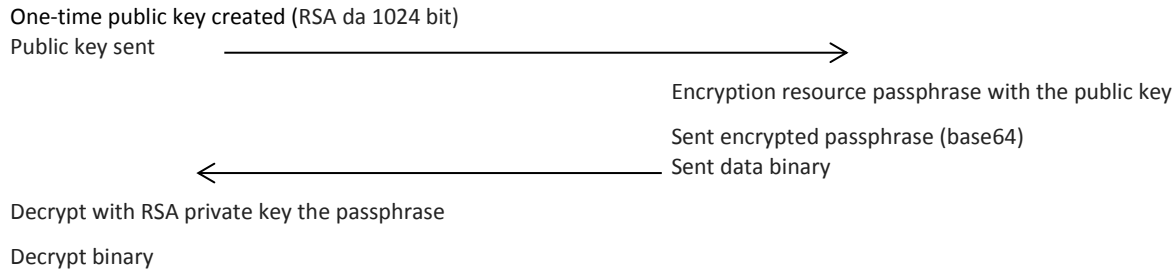
All data exchanged between the VPH-OP hypermodelling framework and its associated data repository (PhysiomeSpace) is encrypted through a public-key cryptography algorithm.<sup>19</sup> Public-key cryptography refers to a cryptographic system requiring two separate keys, one to lock or encrypt the plaintext, and one to unlock or decrypt the cypher text. The distinguishing technique used in public key cryptography is the use of asymmetric key algorithms, where the key used to encrypt a message is not the same as the key used to decrypt it. Each user has a pair of cryptographic keys, a public encryption key and a private decryption key. The publicly available encrypting-key is widely distributed, and the private decrypting-key is known only to the recipient. Messages are encrypted with the recipient's public key and can be decrypted only with the corresponding private key. The keys are related mathematically, but parameters are chosen so that determining the private key from the public key is prohibitively expensive. This encryption mechanism has been implemented in PhysiomeSpace<sup>20</sup> using the *pycrypto*<sup>21</sup> library. Figure 15 represents the scheme on how the system works between the client and the server side for each data resource.



<sup>19</sup> [http://en.wikipedia.org/wiki/Public-key\\_cryptography](http://en.wikipedia.org/wiki/Public-key_cryptography)

<sup>20</sup> <https://www.physioespace.com>

<sup>21</sup> <https://www.dlitz.net/software/pycrypto/>



**Figure 15 Schematic representation on the encryption mechanism**

VPH-OP handles public and private key management but requires the functionality for passphrases to be associated to the resources.

To support the above functionality the CHIC security framework will implement a secured web service through which passphrases can be assigned to resource URIs.

## 5.4 Auditing

The CIAM Audit parser deployed for CHIC publishes a secured REST endpoint listening on port 443 over SSL. The endpoint accepts http POST requests only. The url that must be used by audit clients is:

<https://audit-dev-chic.ustodix.com/logParser/send/record>

The body of the request should contain a JSON string representing the audit log message in XDASv2 format.

Since the REST interface is secured, the http request must contain a security header containing the SAML token. In case of a server sending the log message, the header will contain a delegation token.

The endpoint allows only messages formatted in XDASv2 which means no parsing must be performed which increases performance.

### 5.4.1 Java Spring Based Audit Integration Tutorial

This section gives a tutorial on how to integrate a service provider with the auditing component. In this description it is assumed that you send auditing messages using the XDASv2Event model.

It is assumed that you work with Spring. It is also assumed that the service provider is integrated with the CHIC security framework. All dependencies used in this tutorial can be downloaded from the CHIC wiki.

First add the google gson dependency. The logParsing module expects the XDASv2Event messages to be JSON compatible. The GSON class adds methods to convert POJO objects to JSON objects that can be sent to the logParsing service.

Add the *audit-common-xxx.jar* library to your classpath. It will be accessible at the following location:

This library contains the XDASv2Event class and all its component classes.

The following code gives an example for how to send an event to the auditing service:

```

XDASv2Event logRecord = new XDASv2Event ();
logRecord.setInitiator(initiator);
logRecord.setObserver(observer);
  
```

```
logRecord.setAction(action);
logRecord.setTarget(target);

//setting http headers to Json
HttpHeaders headers = new HttpHeaders();
headers.setContentType(MediaType.APPLICATION_JSON);
headers.setAccept(Arrays.asList(MediaType.APPLICATION_JSON));
//Create the request entity
HttpEntity<String> requestEntity = new HttpEntity<String>( (new Gson()).toJson(logRecord),headers);

//build the rest template and post
RestTemplate template = new RestTemplate();
ResponseEntity<String> entity =
    template.postForEntity("http://chic-parser-hostname/logParsing/send/record",requestEntity, String.class);
if (!entity.getStatusCode().equals(HttpStatus.OK)) {
    String errorMessage = "Exception while sending logs to audit service: HTTP CODE " + entity.getStatusCode();
    LOGGER.error(errorMessage);
}
```

#### 5.4.1.1 Create the XDASv2Event object

The XDASv2Event object and its components, and are bean objects, which means that they can be easily created. This is an example where an XDASv2Event object is created and populated with an existing initiator, observer, action and target.

```
XDASv2Event logRecord = new XDASv2Event ();
logRecord.setInitiator(initiator);
logRecord.setObserver(observer);
logRecord.setAction(action);
logRecord.setTarget(target);
```

The other objects are created in a similar manner.

#### 5.4.1.2 Configure the headers

```
HttpHeaders headers = new HttpHeaders();
headers.setContentType(MediaType.APPLICATION_JSON);
headers.setAccept(Arrays.asList(MediaType.APPLICATION_JSON));
```

Create the headers for the REST call. The log parser expects a message of the type JSON, so be sure to set the content type of the REST message to JSON.

#### 5.4.1.3 Create the request

```
HttpEntity<String> requestEntity = new HttpEntity<String>( (new Gson()).toJson(logRecord),headers);
```

The request is a `HttpEntity`, and uses a body and headers to create. In this case the body is the JSON representation of the `XDASv2Event`, which was created by using the `GSON toJson` method on the `XDASv2Event` object.

#### 5.4.1.4 Build the rest template and post

```
RestTemplate template = new RestTemplate();
ResponseEntity<String> entity =
    template.postForEntity("http://chic-parser-hostname/logParsing/send/record",requestEntity, String.class);
if (!entity.getStatusCode().equals(HttpStatus.OK)) {
    String errorMessage = "Exception while sending logs to audit service: HTTP CODE " + entity.getStatusCode();
    LOGGER.error(errorMessage);
}
```

Create a new REST template, and send it to the log parser using the *postForEntity* method. This method needs the URL of the log parser, the request object (body with `XDASv2Event` and headers), and a response type). It can be checked whether the message was well received by looking at the response entity.

## 6 References

- [1] OASIS, Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0, 20wsd5, <http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf>
- [2] OpenID Foundation, OpenID Authentication, version 2.0, 2007, <http://openid.net/specs/openid-authentication-2.0.html>
- [3] OpenID Foundation, OpenID Connect Core, version 1.0, 2014, <http://openid.net/specs/openid-connect-core-1.0.html>
- [4] W3C, Web Services Description Language (WSDL), version 1.1, <http://www.w3.org/TR/wsdl>
- [5] W3C, WSDL 1.1 Binding Extension for SOAP 1.2, <http://www.w3.org/Submission/wsdl11soap12/>
- [6] OASIS, WS-Security specification, version 1.0, 2004, <http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf>
- [7] OASIS, WS-Security specification, version 1.1, 2006, <https://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf>
- [8] W3C, Web Services Policy, Version 1.5, <http://www.w3.org/TR/ws-policy/>
- [9] W3C, Web Services Addressing, <http://www.w3.org/Submission/ws-addressing>
- [10] W3C, Web Services Addressing - SOAP Binding, <http://www.w3.org/TR/ws-addr-soap/>
- [11] OASIS, WS-Trust specification, version 1.3, 2007, <http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.pdf>
- [12] OASIS, WS-Trust specification, version 1.4, 2009, <http://docs.oasis-open.org/ws-sx/ws-trust/v1.4/os/ws-trust-1.4-spec-os.pdf>
- [13] OASIS, WS-SecureConversation specification, version 1.4, 2009, <http://docs.oasis-open.org/ws-sx/ws-secureconversation/v1.4/os/ws-secureconversation-1.4-spec-os.pdf>
- [14] OASIS, WS-Policy specification, version 1.5, 2007, <http://www.w3.org/TR/ws-policy/>
- [15] OASIS, WS-Policy Attachment Specification, version 1.5, 2007, <http://www.w3.org/TR/ws-policy-attach/>
- [16] OASIS, WS-SecurePolicy, version 1.3, 2009, <http://docs.oasis-open.org/ws-sx/ws-securitypolicy/v1.3/os/ws-securitypolicy-1.3-spec-os.pdf>
- [17] OASIS, WS-SecureConversation, version 1.4, 2009, <http://docs.oasis-open.org/ws-sx/ws-secureconversation/v1.4/ws-secureconversation.html>
- [18] OASIS, WS-Federation specification, version 1.2, 2009, <http://docs.oasis-open.org/wsfed/federation/v1.2/os/ws-federation-1.2-spec-os.pdf>
- [19] IETF, rfc 6101: The Secure Sockets Layer (SSL) Protocol, Version 3.0, 2011, <http://tools.ietf.org/html/rfc6101>
- [20] IETF, rfc 2246: The Transport Layer (TLS) Protocol, Version 1.0, 1999, <http://tools.ietf.org/html/rfc2246>
- [21] IETF, rfc 4346: The Transport Layer (TLS) Protocol, Version 1.1, 2006, <http://tools.ietf.org/html/rfc4346>
- [22] IETF, rfc 5246: The Transport Layer (TLS) Protocol, Version 1.2, 2011, <http://tools.ietf.org/html/rfc5246>

- [23] NHS, Approved Cryptographic Algorithms Good Practice Guideline, version 3.0, 2012, <http://systems.hscic.gov.uk/infogov/security/infrasec/gpg/acs.pdf>
- [24] <http://www.opengroup.com/xdas/2008>
- [25] Oracle, Key and Certificate Management Tool, <http://docs.oracle.com/javase/6/docs/technotes/tools/solaris/keytool.html>
- [26] Network Working Group, RFC 1951 DEFLATE Compressed Data Format Specification, version 1.3, 1996
- [27] RabbitMQ, <http://www.rabbitmq.com>
- [28] MongoDB, <http://www.mongodb.org>

## Appendix 1 – Audit Message Json Schema

```
{
  "id": "XDASv2",
  "title": "XDAS Version 2 JSON Schema",
  "description": "A JSON representation of an XDASv2 event record.",
  "type": "object",
  "required": [ "XDASID", "XDASVersion", "id", "initiator", "observer", "action" ],
  "properties": {
    "XDASID": {
      "type": "string",
      "default": "XDASv2CIAM"
    },
    "XDASVersion": {
      "type": "string",
      "default": "http://www.custodix.com/xdasv2/2014/v1"
    },
    "id": {
      "type": "string",
      "description": "A locally unique id to reference this event."
    },
    "source": {
      "description": "The original source of the event, if applicable.",
      "type": "string"
    },
    "initiator": {
      "description": "The authenticated entity or access token that causes an event.",
      "type": "object",
      "required": [ "account", "entity" ],
      "additionalProperties": false,
      "properties": {
        "account": {
          "$ref": "#/account"
        },
        "entity": {
          "$ref": "#/entity"
        }
      },
      "assertions": {
        "description": "Attribute/value assertions about an identity.",
        "type": "array",
        "uniqueItems": true,
        "items": {
          "description": "An assertion attribute entry.",
          "type": "object",
          "required": [ "key", "value" ],
          "additionalProperties": false,

```



```

        "properties": {
            "key": {
                "type": "string"
            },
            "value": {
                "type": "string"
            }
        }
    }
},
"target": {
    "description": "The target object, account, data item, etc of the event.",
    "type": "object",
    "required": [ "name", "entity" ],
    "additionalProperties": false,
    "properties": {
        "name": {
            "description": "Name of the target, used in the audit viewer table.",
            "type": "string"
        },
        "account": {
            "$ref": "#/account"
        },
        "entity": {
            "$ref": "#/entity"
        },
        "targetDataAttributes": {
            "description": "A set attribute/value pairs describing the target object.",
            "type": "array",
            "additionalProperties": false,
            "uniqueItems": true,
            "items": {
                "description": "A data attribute entry.",
                "type": "object",
                "required": [ "key", "value" ],
                "additionalProperties": false,
                "properties": {
                    "key": {
                        "type": "string"
                    },
                    "value": {
                        "type": "string"
                    }
                }
            }
        }
    }
}

```

```

    }
  }
},
"observer": {
  "description": "The recorder (ie., the XDASv2 service) of the event.",
  "type": "object",
  "required": [ "entity" ],
  "additionalProperties": false,
  "properties": {
    "account": {
      "$ref": "#/account"
    },
    "entity": {
      "$ref": "#/entity"
    }
  }
},
"action": {
  "description": "The action describes the event in a uniform manner.",
  "type": "object",
  "required": [ "event", "time", "outcome" ],
  "additionalProperties": false,
  "properties": {
    "event": {
      "description": "The event identifier in standard XDASv2 taxonomy.",
      "type": "object",
      "required": [ "id", "name" ],
      "additionalProperties": false,
      "properties": {
        "identifier": {
          "description": "The XDASv2 taxonomy event identifier.",
          "type": "string",
          "enum": ["Create Account", "Delete Account", "Disable Account", "Enable Account", "Query Account", "Modify Account", "Modify Account Security Token", "Query Account Security Token", "Delete Account Security Token", "Create Session", "Terminate Session", "Query Session", "Modify Session", "Create Data Item", "Delete Data Item", "Query Data Item Attribute", "Modify Data Item Attribute", "Install Service", "Remove Service", "Query Service Configuration", "Modify Service Configuration", "Disable Service", "Enable Service", "Invoke Service", "Terminate Service", "Query Process Context", "Modify Process Context", "Create Peer Association", "Terminate Peer Association", "Query Association Context", "Modify Association Context", "Receive Data Via Association", "Send Data Via Association", "Create Data Item Association", "Terminate Data Item Association", "Query Data Item Association", "Modify Data Item Association", "Query Data Item Contents", "Modify Data Item Contents", "Request Work Flow Approval", "Receive Work Flow Approval", "Escalate Work Flow Approval", "Send Work Flow Notification", "Create Role", "Delete Role", "Disable Role", "Enable Role", "Query Role", "Modify Role", "Start System", "Shutdown System", "Resource Exhaustion", "Resource Corruption", "Resource Unavailable", "Resource Available", "Back up Data Store", "Recover Data Store", "Configure Audit Service", "Audit Data Store Full", "Audit Data Store Corrupted", "Authenticate Session", "Unauthenticate Session", "Federate Identity", "Unfederate Identity", "Create Access Token", "Destroy Access Token", "Other"]
        }
      }
    },
    "name": {
      "description": "A short descriptive name for the specific event.",
      "type": "string"
    }
  }
}

```

```

    },
    "correlationId": {
      "description": "A locally unique id to allow linking correlated events (i.e. belonging to the same event trail).",
      "type": "string"
    }
  }
},
"subEvent": {
  "type": "object",
  "description": "Describes the actual domain specific event that has occurred.",
  "additionalProperties": false,
  "properties": {
    "name": {
      "description": "A short descriptive name for this event.",
      "type": "string"
    },
    "message": {
      "description": "Descriptive message explaining the event.",
      "type": "string"
    }
  }
},
"log": {
  "description": "Client-specified logging attributes.",
  "additionalProperties": false,
  "properties": {
    "severity": {
      "type": "integer"
    },
    "priority": {
      "type": "integer"
    },
    "facility": {
      "type": "integer"
    }
  }
},
"time": {
  "description": "The time the event occurred.",
  "type": "object",
  "required": [ "offset" ],
  "additionalProperties": false,
  "properties": {
    "offset": {
      "description": "Milliseconds since Jan 1, 1970.",
      "type": "integer"
    }
  }
}

```

```

    },
    "tolerance": {
      "description": "A tolerance value in milliseconds.",
      "type": "integer"
    },
    },
    "certainty": {
      "description": "Percentage certainty of tolerance.",
      "type": "integer",
      "minimum": 0,
      "maximum": 100
    },
    },
    "source": {
      "description": "The time source (eg., ntp://time.nist.gov).",
      "type": "string"
    },
    },
    "zone": {
      "description": "A valid timezone symbol (eg., MST/MDT).",
      "type": "string"
    }
  }
}
},
"outcome": {
  "description": "The XDASv2 taxonomy outcome identifier.",
  "type": "string",
  "enum": [ "Success","Success Priv Used","Success Priv Granted","Success Priv Revoked", "Fail","Fail Service
Unavailable","Fail Service Failure","Fail Hardware Failure", "Fail Service Error", "Fail Busy","Fail Disabled","Fail Invalid
Input","Fail Entity Exists","Fail Entity Non Existent", "Deny","Deny Insufficient Authorization","Deny Invalid Identity","Deny Invalid
Credentials", "Other" ]
},
"extendedOutcome": {
  "description": "An ID for the services specific outcome.",
  "type": "string"
}
}
},
"account": {
  "description": "A representation of an XDAS account.",
  "type": "object",
  "required": [ "id" ],
  "additionalProperties": false,
  "properties": {
    "domain": {
      "description": "A (URL) reference to the authority managing this account.",
      "type": "string"
    },
  },
  "name": {

```

```
"description": "A human-readable account name.",
"type": "string"
},
"identifier": {
  "description": "A machine-readable unique account identifier value.",
  "type": "string"
},
"delegatee": {
  "$ref": "#/account"
}
}
},
"entity": {
  "description": "A representation of an addressable entity.",
  "type": "object",
  "additionalProperties": false,
  "properties": {
    "sysAddr": {
      "type": "string"
    },
    "sysName": {
      "type": "string"
    },
    "svcName": {
      "type": "string"
    },
    "svcComp": {
      "type": "string"
    }
  }
}
}
```