



Deliverable No. 5.2.2

Final version of security tools and guidelines

Grant Agreement No.: 600841
Deliverable No.: D5.2.2
Deliverable Name: Final version of security tools and guidelines
Contractual Submission Date: 30/09/2016
Actual Submission Date: 30/09/2016

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.:	D5.2.2
Document name:	Final version of security tools and guidelines
Nature (R, P, D, O) ¹	P
Dissemination Level (PU, PP, RE, CO) ²	CO
Version:	3.2
Actual Submission Date:	30/09/2016
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.

¹ R=Report, P=Prototype, D=Demonstrator, O=Other

² 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)

MODIFICATION CONTROL				
Version	Date	Status	Author	Modification Control
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 Dawn Walker, 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
3.0	27/09/2016	Draft	Elias Neri, Custodix Stelios Sfakianakis, Forth	Update to describe the final version of the security framework with amongst others the following changed: <ul style="list-style-type: none"> - Updated chapter 3.2 "Authorisation" - Added chapter 4 and 5 describing the CHIC research and clinical platform - Removed chapter 7.1.5 on Liferay integration
3.1	28/09/2016	Internal Review	Debora Testi, CINECA	Internal Review
3.2	30/09/2016	Final	Elias Neri, Custodix	

List of contributors

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

Contents

1	EXECUTIVE SUMMARY.....	5
2	INTRODUCTION	6
3	GENERAL SECURITY FRAMEWORK	7
3.1	AUTHENTICATION	7
3.2	AUTHORISATION.....	13
3.3	AUDITING	16
4	THE CHIC RESEARCH PLATFORM.....	22
4.1	SECURITY VIEW	23
4.2	SECURITY VOCABULARY	28
5	THE CHIC CLINICAL PLATFORM.....	29
5.1	SECURITY VIEW	31
5.2	SECURITY VOCABULARY	39
6	SECURITY FRAMEWORK INTEROPERABILITY GUIDELINES.....	40
6.1	TRANSPORT LAYER SECURITY	40
6.2	MESSAGE PROTOCOLS AND MESSAGE LAYER SECURITY	41
6.3	PASSIVE AUTHENTICATION PROTOCOL	47
7	SECURITY FRAMEWORK INTEGRATION TUTORIALS	49
7.1	PASSIVE (BROWSER BASED) AUTHENTICATION	49
7.2	ACTIVE AUTHENTICATION.....	62
7.3	INTEGRATION OF SECURITY FRAMEWORK WITH VPH-OP	83
7.4	AUDITING	86
8	REFERENCES	89
	APPENDIX 1 – AUDIT MESSAGE JSON SCHEMA	91

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 XDS.

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 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 and REST services CHIC implements WS-* which handles message level security and brokered authentication.

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.

Chapter 3 describes the security framework components, and Chapter 6 describes through interoperability guidelines the minimal requirements to which services should adhere to be allowed to integrate with the CHIC architecture.

From a security point of view two distinguishable deployments of the CHIC platform can be identified:

- CHIC as a research platform (Chapter 4) accessible by data providers, model developers and researchers. The research platform makes use of de-facto anonymous data and access is governed through data provision and data access contracts with the Center for Data Protection.
- CHIC as a clinical platform (Chapter 5) accessible by healthcare professionals treating a patient. As this is a care oriented environment the data does not have to be de-facto anonymised. Instead CHIC is provided as a service to hospitals. Hospitals are responsible for managing access such as registration of healthcare professions and assignment of access rights. Patient data access should be limited to the data nurses and physicians directly responsible for a patient's treatment.

The final chapter (Chapter 7) 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 General Security Framework

The CHIC security architecture is based on the security framework developed in past and ongoing EU projects such as ACGT³, P-Medicine⁴, EURECA⁵, and INTEGRATE⁶. 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-* and 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 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/her credentials each time he/she 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.

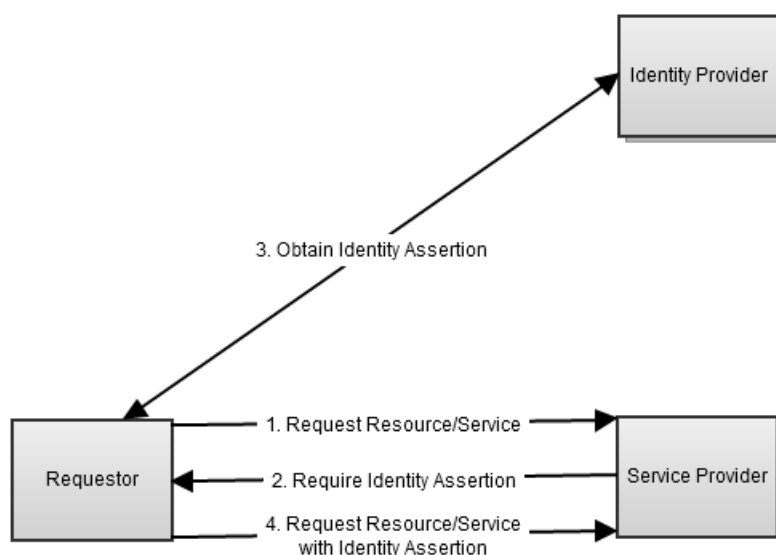


Figure 1 Brokered Authentication Flow

³ <http://acgt.ercim.eu/>

⁴ <http://www.p-medicine.eu/>

⁵ <http://eurecaproject.eu/>

⁶ <http://www.fp7-integrate.eu/>

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 it 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). After the requestor receives the identity assertion from the identity provider, it sends it to the service provider as claims about its identity and, if it is granted access, it receives the requested resource (4).

Enhanced clients (i.e. non-browser clients) such as desktop applications 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 CHIC identity assertions/tokens is defined by the SAML (Security Assertion Markup Language [1]) standard. SAML is an OASIS⁷ standard that defines signed identity assertions in an XML-based format. Such an identity assertion contains the user's authentication and attribute 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.

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.

⁷ <https://www.oasis-open.org/>

3.1.2 Browser-based brokered authentication for Web Sites

There are two important standards that define how a SAML token can be exchanged between an identity and service provider through a browser:

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

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 3.1.3. The WS-Federation Passive Requestor Profile is mainly implemented by Microsoft in its Active Directory suite and extensively used in .NET web applications. Four years ago the WS-Federation Passive Requestor Profile was not yet commonly used and implemented outside of the Microsoft ecosystem.

The **SAML Web Browser SSO Profile** is part of the SAML (Security Assertion Markup Language) 2.0 standard [1]. SAML 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).

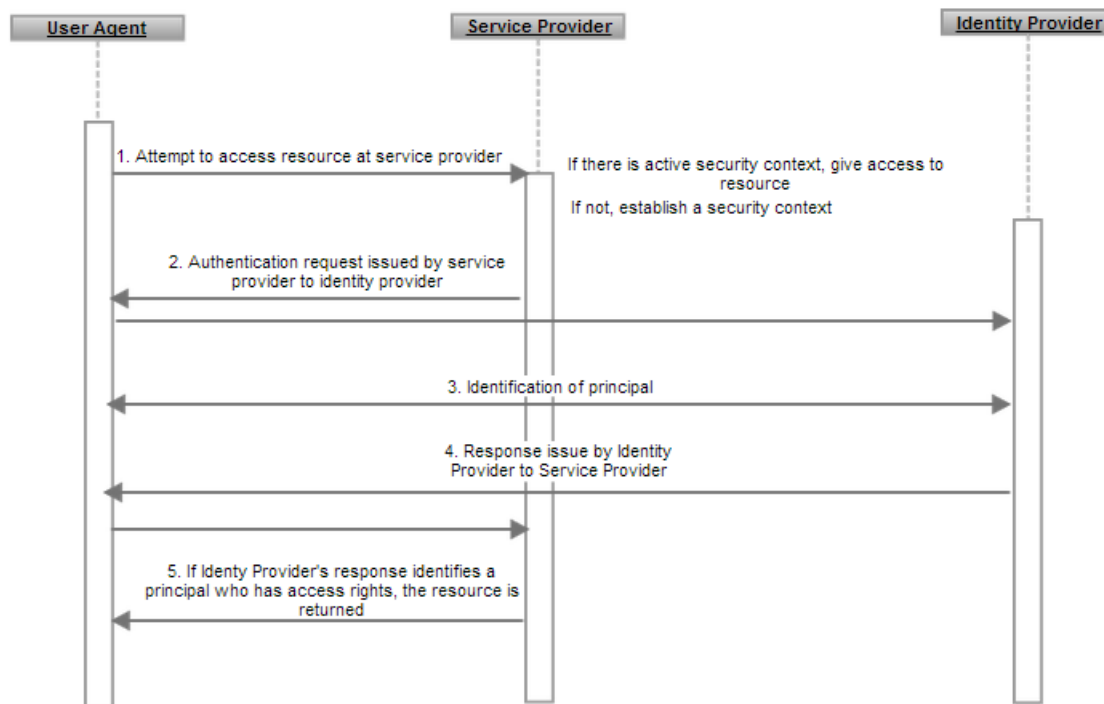


Figure 2 SAML Browser SSO Profile

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/her. 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.

3.1.3 Brokered authentication for web services

Web services such as SOAP or REST web services are typically called by enhanced (non browser) clients such as desktop or server applications. Such enhanced clients usually have the capability to directly contact identity providers contrary to browsers, which as seen before require a redirection from the service provider to the selected identity provider (and vice versa). This will have impact on the communication flow and thus different communication protocols are available for web services than for browsers. Web service protocols are active protocols in the sense that the client actively takes part in the communication protocol by explicitly requesting authentication tokens. A browser in contrast takes a more passive role in the authentication protocol by for example following HTTP redirects.

The **WS-Federation Active Requestor Profile** [18] is the most commonly used communication protocol for the exchange of authentication tokens with a web service provider. The WS-* (such as WS-Federation, WS-Trust) specifications more broadly are a standard to secure SOAP Web Services. Together these WS-* 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.

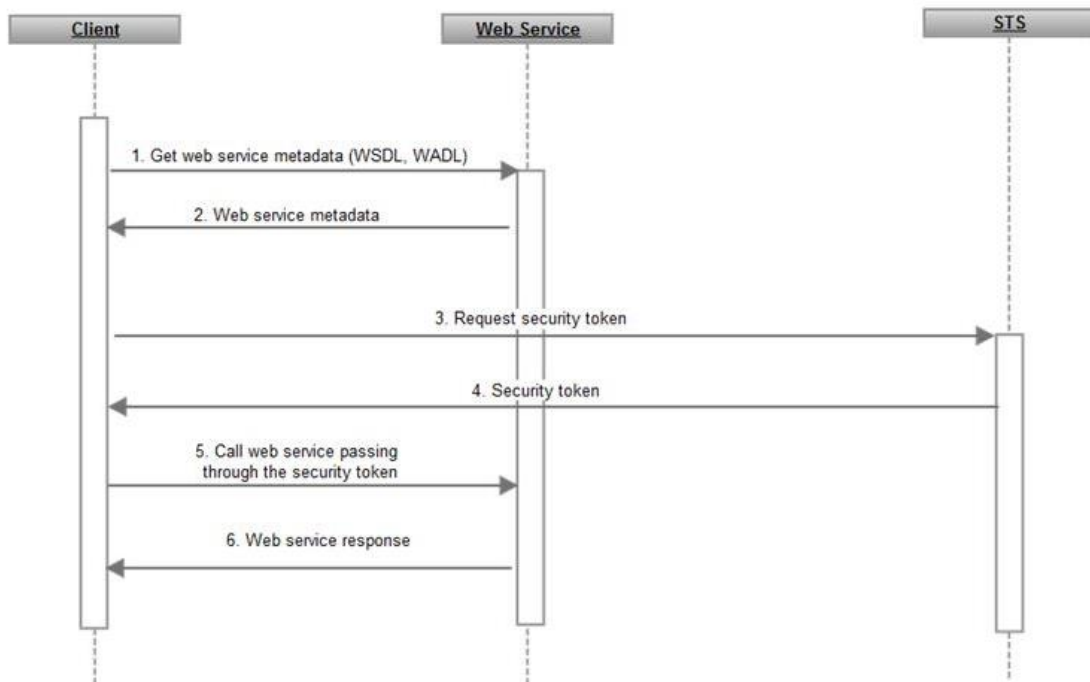


Figure 3 WS-Federation Active Requestor Profile

2. The Passive Requestor Profile, which defines a protocol for browsers to exchange identity token between identity and service providers. Chapter 3.1.2 describes passive protocols in more detail .

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.

The CHIC REST web services also make use of this WS-Trust Secure Token Service. Contrary to SOAP calls, the authentication token retrieved from the STS cannot be passed to the web service through SOAP headers. Instead a SAML token should be passed to a REST service through the HTTP authorization header. The authorization header value is 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.

3.1.4 Component Overview

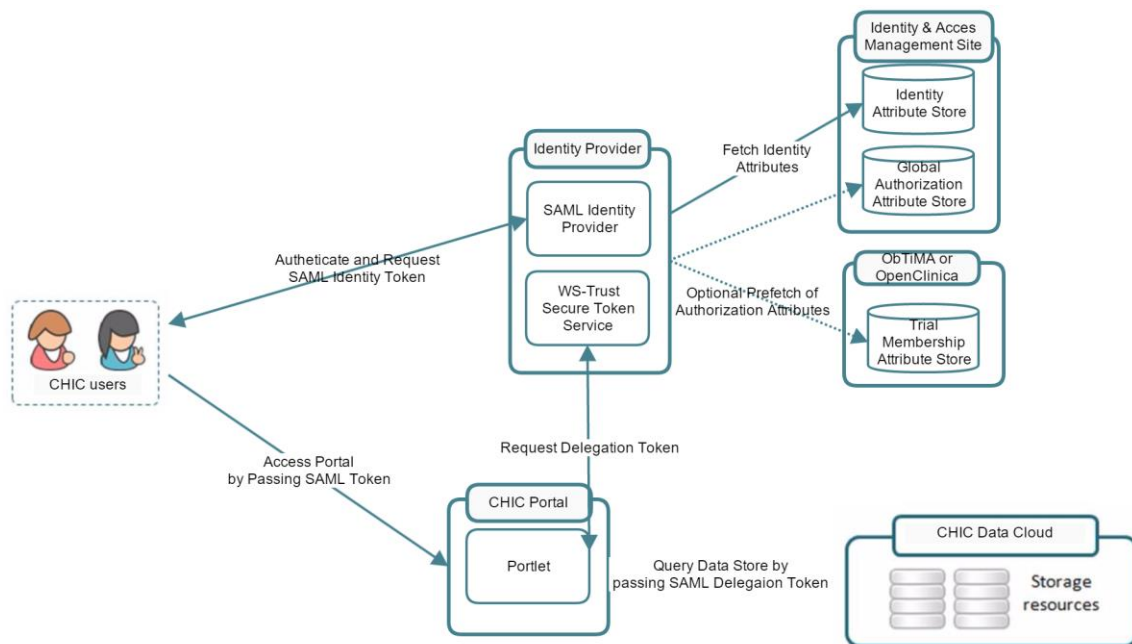


Figure 4 CHIC Security Components (Authentication)

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

CHIC provides a central authorisation services to manage and evaluate access requests coming from the different services and applications that restrict their resources by access control. These requests are evaluated by the authorisation service based on the defined (Access Control) policies.

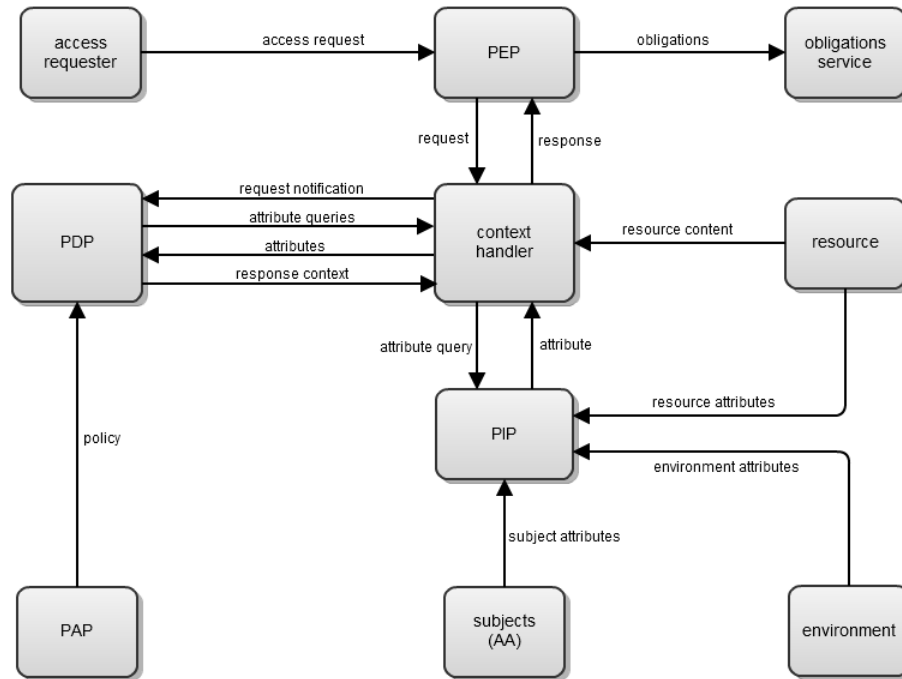


Figure 5: XACML Dataflow Diagram

Within CHIC, these policies will be XACML based and the authorisation service will evaluate XACML decision request messages send by the Policy Enforcement Points (PEPs). The component responsible for the validation (making access decisions) is called the Policy Decision Point (PDP).

Next to a PDP, a context handler component is part of the authorisation service. This component will augment the decision request messages coming from the PEPs with additional (probably missing) attributes obtained from Attribute Authorities (AAs) or Attribute Repositories. The AAs are accessed through a Policy Information Point (PIP). The context handler will also introduce supporting components complementary to a standard XACML PDP specification, which solves some of the limitations (like contextualisation) encountered in the specification.

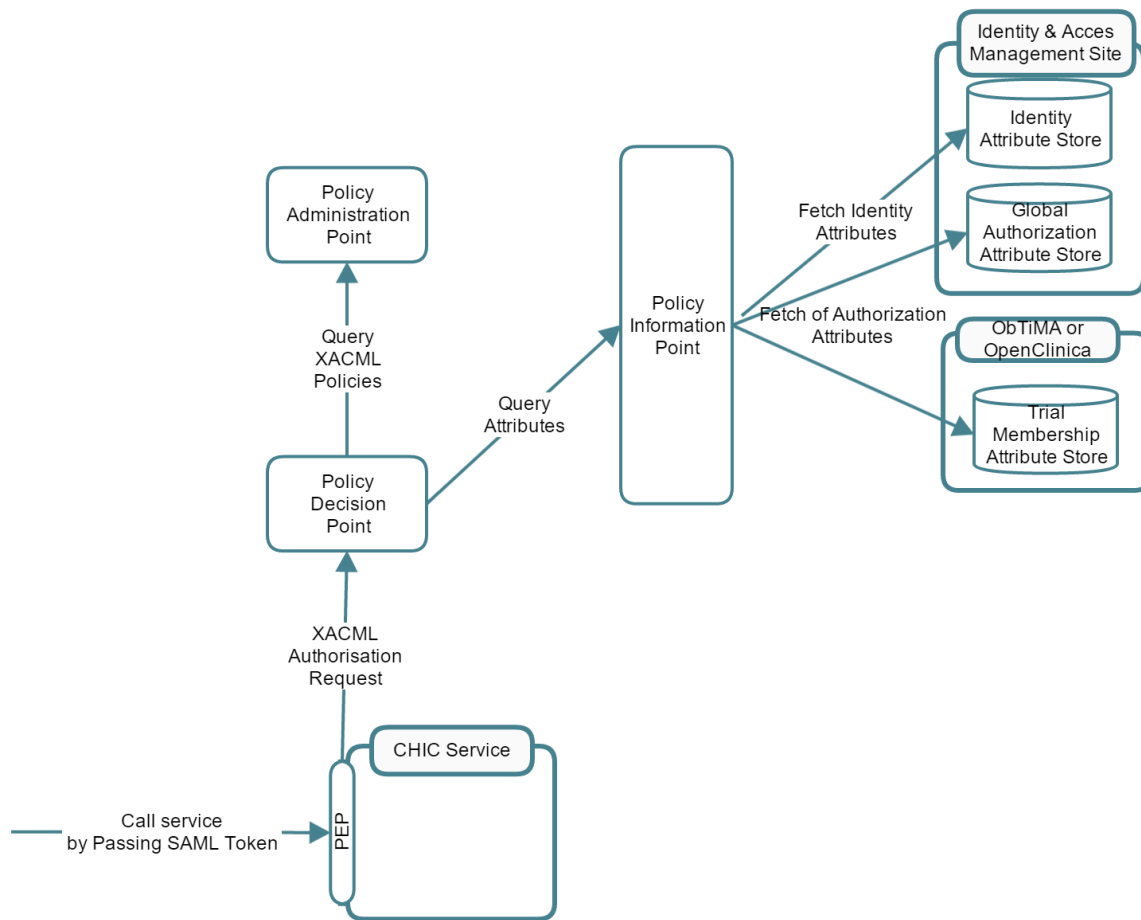


Figure 6 CHIC Security Components (Authorisation)

Figure 6 shows the interaction between the different authorization components. The general flow is straightforward. A Policy Enforcement Point (PEP) integrates a CHIC service in the authorisation framework. When a service needs an authorisation decision, the PEP will create a decision request, send it to the CHIC Policy Decision Point (PDP) and 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 principal performing the action, the resource on which the action is performed and the actual action.

Upon receiving a decision request the PDP will interpret the request and 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 CDR or clinical trial software (e.g. ObTiMA or OpenClinica).

3.2.1 Policy Administration Point

The Policy Administration Point (PAP) is responsible for authoring and management of access control policies. These access control policies are a formal set of rules that define what action, if any, a subject can take on a particular resource. Using these policies the authorisation service can make a decision (grant or deny access) on an incoming access request.

3.2.2 Policy Information Point

The Policy Information Point (PIP) is responsible for the resolution of possible missing attributes in an access control request coming from the Policy Enforcement Point (PEP). These attributes are needed by the authorisation service, i.e. the Policy Decision Point (PDP), to come to an accurate decision about the request. If this authorisation service needs further information to evaluate the incoming request from the PEP, the PIP is called, either directly by the authorisation service itself or by returning missing attributes to the PEP, which have to invoke the PIP.

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 XDAS2 [24].

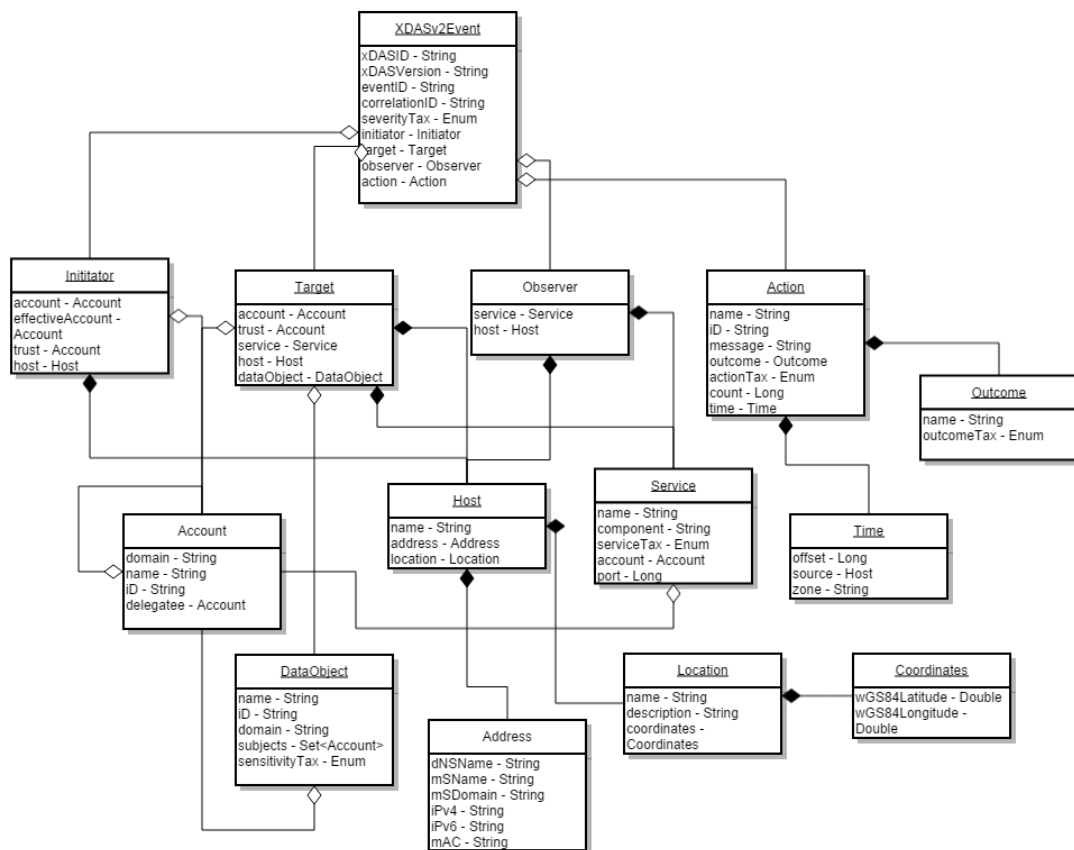


Figure 7 Audit Data Model

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 8Error! Reference source not found.) 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

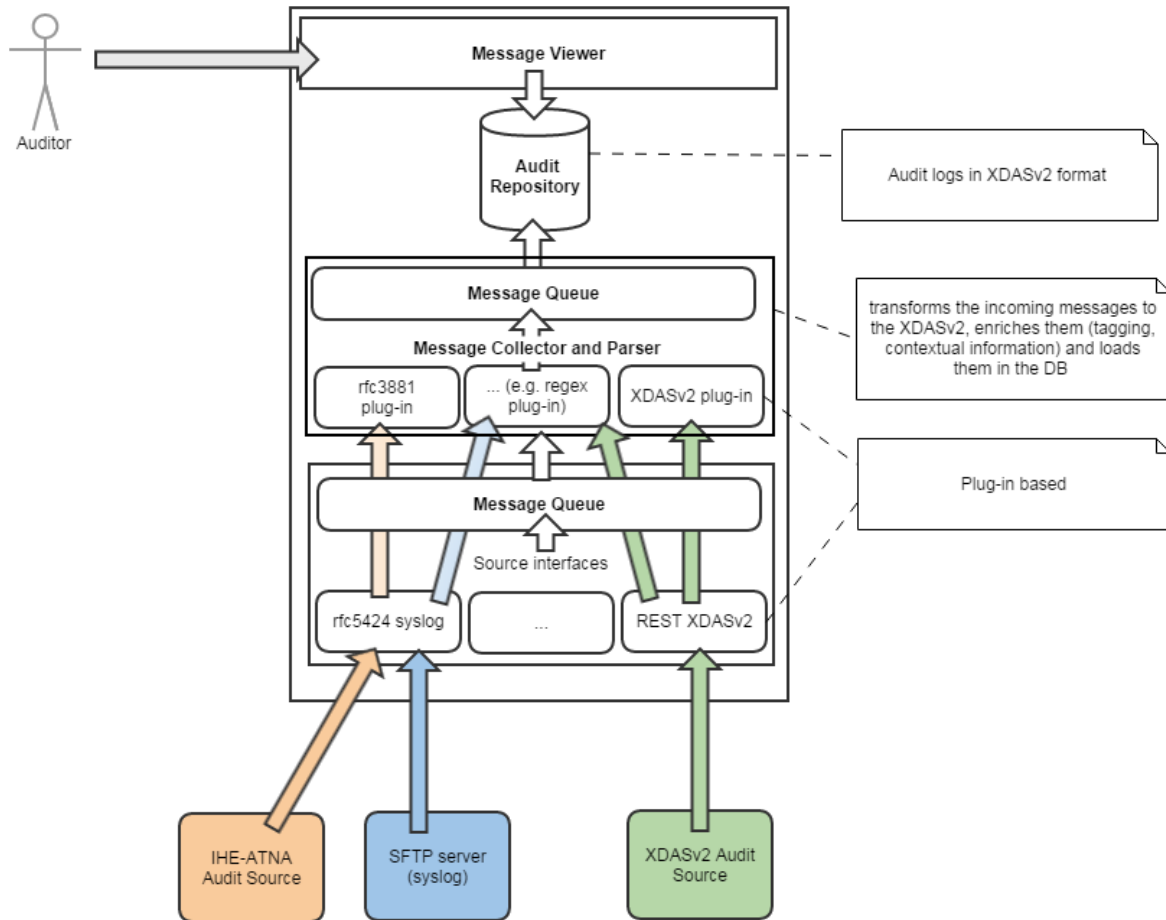


Figure 8 Audit Conceptual Architecture

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 9) depicts how these endpoints can be configured in the CHIC platform. Following fields can be set:
 - a) Port: On what port is the endpoint listening
 - b) Protocol: What's the transport level protocol
 - c) Listening: To enable/disable the endpoint
 - d) Applications: This field is the 'name' of the endpoint. This name can be used to query audit trails in the audit viewer
 - e) 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.

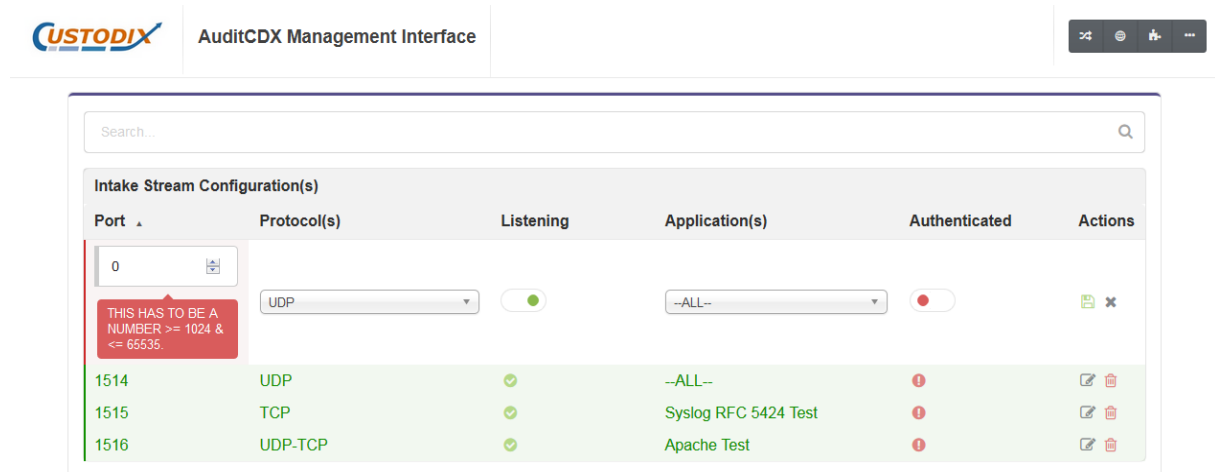


Figure 9 Audit Management Interface

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 'CUSTODIX' logo and the title 'AuditCDX Web Viewer'. Below the header, there are several filter fields: 'Initiator', 'Action Type', 'Observer', 'Target', 'Time from', 'Time to', 'Outcome', 'Severity', and 'Sensitivity'. A dropdown menu is open for the 'Outcome' field, showing a list of error codes: DENY_DENIAL, DENY_INSUFFICIENT_AUTH, DENY_INVALID_CREDENTIAL, DENY_INVALID_IDENTITY, FAIL_ALREADY_ENABLED, FAIL_BUSY, FAIL_DISABLED, FAIL_ENTITY_NON_EXISTENT, FAIL_FAILURE, and FAIL_HARDWARE_FAILURE. Below the filters, there's a table of audit events. The table has columns for Time, Action, Initiator, Observer, Target, Outcome, Severity, and Sensitivity. The first few rows of the table are as follows:

Time	Action	Initiator	Observer	Target	Outcome	Severity	Sensitivity
15:45:16 - 03/06/2014	Modify Process Context	ServiceName	ServiceName	ServiceName	FAIL_ALREADY_ENABLED	High	High
15:45:14 - 03/06/2014	Configure Audit Service	ServiceName	ServiceName	ServiceName	FAIL_ALREADY_ENABLED	High	High
15:45:12 - 03/06/2014	Query Data Item Association	ServiceName	ServiceName	ServiceName	FAIL_ALREADY_ENABLED	High	High
15:45:11 - 03/06/2014	Federate Identity	ServiceName	ServiceName	ServiceName	FAIL_ALREADY_ENABLED	High	High
15:45:09 - 03/06/2014	Terminate Session	ServiceName	ServiceName	ServiceName	FAIL_ALREADY_ENABLED	High	High

At the bottom of the interface, there are pagination controls showing '1' and '2'.

Figure 10 Audit Web Viewer

4 The CHIC research platform

For research purposes, model development, testing and validation, the CHIC platform provides a research platform. Through this CHIC instance authorised researchers are able to develop, test and validate their models by using the available de-facto anonymous patient data. Access is granted to researchers through data access contracts. As researchers work on anonymous data there is no link back to the original patient. Only under rare conditions and with explicit approval by the Center for Data Protection could a link be reconstructed to allow some information (such as important results for the patient's treatment) to go back to the hospital.

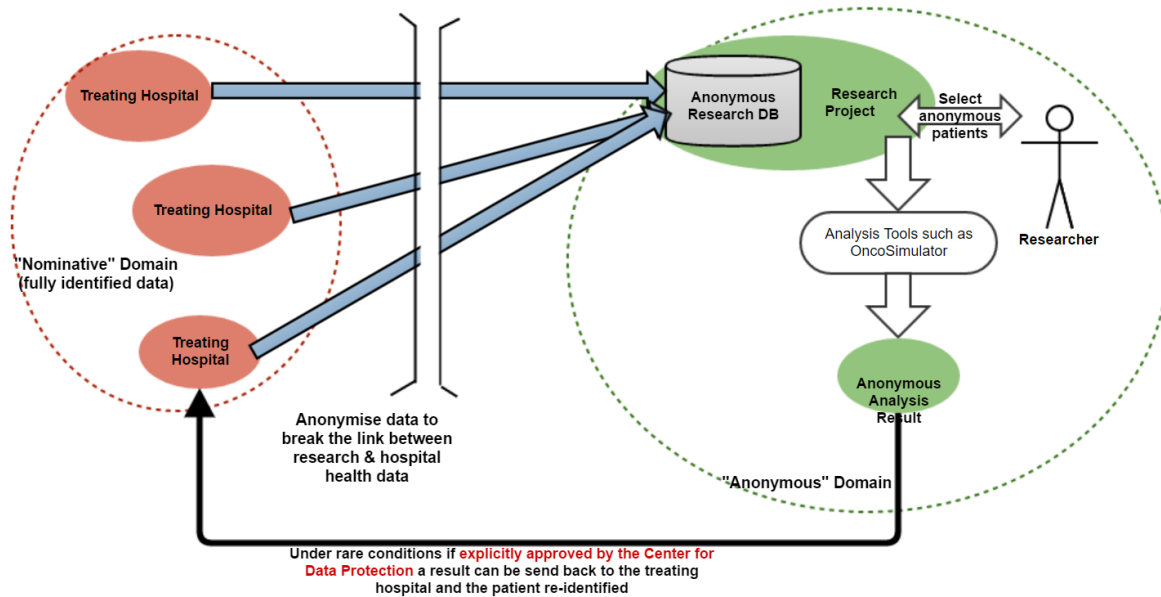


Figure 11 Sensitive data flow in the CHIC research platform

Within the research platform, three main user types can be distinguished:

- Data Uploaders (typically either treating physicians or data nurses). Data uploaders can upload patient data for de-identification into the CHIC platform. This personal patient data is first de-identified at the client level and uploaded through a Trusted Third Party to render the data de-facto anonymous. This de-facto anonymous data is then finally uploaded into the CHIC clinical data warehouse.
- Researchers with data access. These researchers have access to the anonymous CHIC patient data and can thus run their models on this data and access the calculation results.
- Researchers without data access. These users can access the CHIC platform but have no access to the anonymous patient data or respective calculation results.

Access to research data is not fine grained. Researchers who have signed the data access contracts have access to any of the available anonymous patient data. A user's identity token defines whether a user has signed the contract and whether he/she is thus allowed data access.

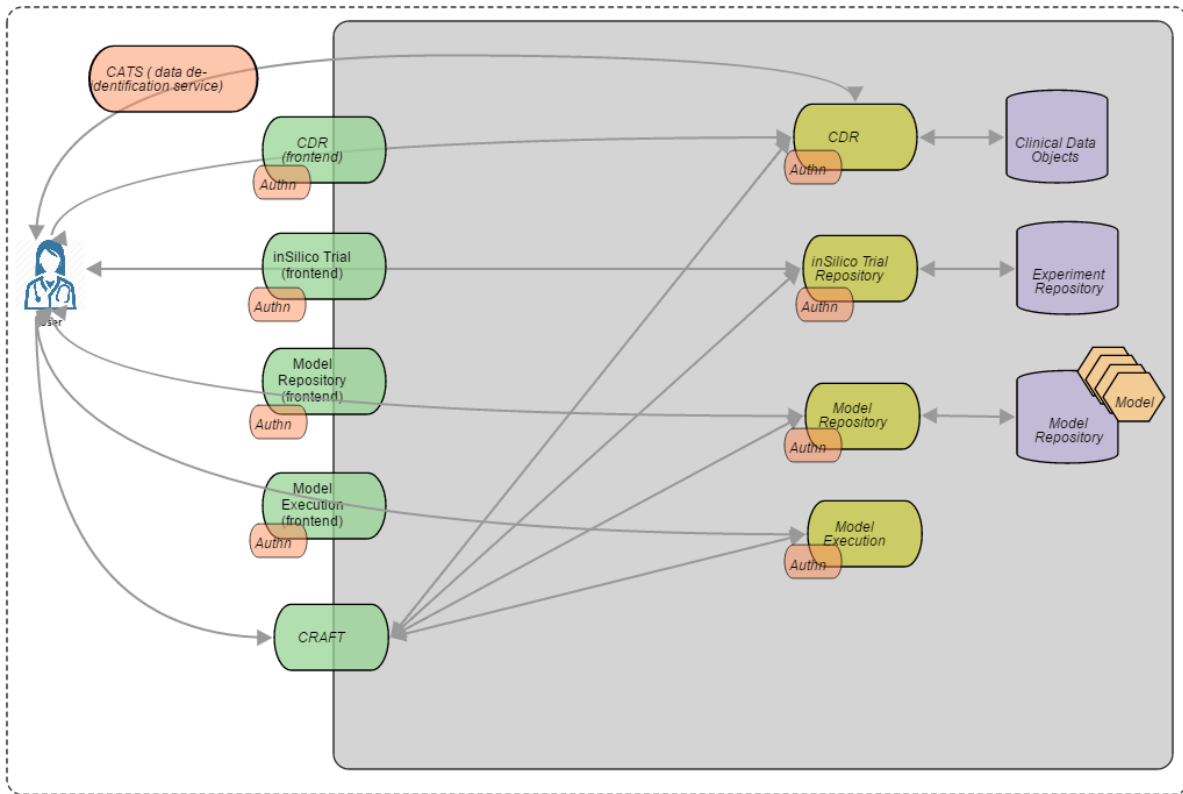


Figure 12 CHIC Research Platform Deployment Diagram

When a researcher accesses one of the CHIC services (such as the CDR frontend) he/she is redirected to the CHIC identity provider (IdP) for authentication. Upon successful authentication the Idp will redirect the user to the CHIC service (such as the CDR frontend) passing through an authentication token. The respective CHIC service will validate the token, check the user's identity attribute and if he/she has signed the data access contracts give the user access to the anonymous patient data.

4.1 Security View

4.1.1 Authentication

Users interacting with the CHIC research services and resources need to be authenticated and authorised before they can access these. A subset of the main general security components as defined in Chapter 3 "General Security Framework" is listed in this section.

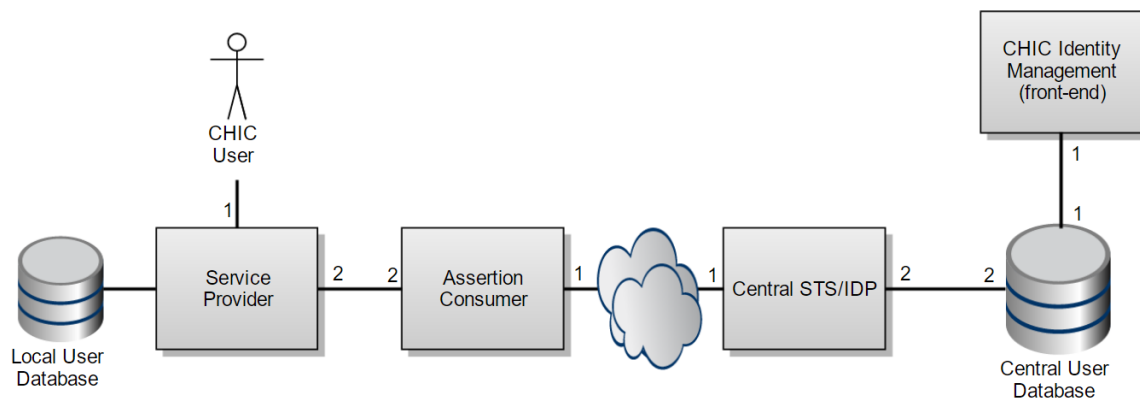


Figure 13: Authentication security view

4.1.1.1 Service Provider

A service provider represents a secured service in the CHIC research platform accessible by authenticated CHIC users. A service provider can have a local storage for storing service specific user data.

Interface	Description
1	A CHIC user interacts with the front-end part of a CHIC service provider. This front-end requires that the user is authenticated.
2	<i>No interface exposed to the assertion consumer</i>

4.1.1.2 Assertion Consumer

The assertion consumer will check whether a user is authenticated to the service provider. If not, the user is redirected to the IdP where he/she must enter his/her credentials. The returned authentication token of the IdP is validated by the assertion consumer.

Interface	Description
1	<i>No interface exposed to the central STS/IdP</i>
2	The assertion consumer exposes following functionality to the service provider: <ul style="list-style-type: none"> Return user identity information of an authenticated user Enforce authentication of a user

4.1.1.3 Central STS/IdP

The security token service (STS) or identity provider (IdP) is responsible for the actual authentication. Together with the security framework, it provides the necessary functionality to successfully authenticate users. It requests credentials, provides the identity assertion, etc.

Interface	Description
1	The central STS/IdP provides following functionality to the assertion consumer: <ul style="list-style-type: none"> Provide identity assertion based on given credentials
2	<i>No interface exposed to the Central User Database.</i>

4.1.1.4 CHIC Identity Management

The CHIC Identity Management component handles user management. The component is a front-end to the central user database, allowing for easier management of the different users within the CHIC platform.

Interface	Description
1	<i>No interface exposed to the Central User Database</i>

4.1.1.5 Central User Database

The Central User Database stores the CHIC user identities.

Interface	Description
1	The Central User Database exposes following functionality to the master user management: <ul style="list-style-type: none"> Modification of user data stored in the central user database
2	The Central User Database exposes following functionality to the central STS/IDP: <ul style="list-style-type: none"> Retrieval of user information from the central user database

4.1.2 Authorization

Access to data within the CHIC research platform is governed by the Center of Data Protection through data access contracts. Model researchers, who have signed the data access contract, are allowed access to the research data. The IdP and STS will issue an attribute in all identity tokens stating whether a user has signed the data access contract and is thus allowed access.

The CHIC Research Platform does not provide a central authorisation service. Instead all services should take their own authorisation decision based on the identity token and only give a user access to patient data and derived calculated results if the authenticated principal has signed the data access contract.

4.1.3 Data Upload and de-identification

As described in D4.3.2 data is de-identified in two de-identification rounds before being uploaded to the CHIC Research Platform.

It is the responsibility of data providers to de-identify a dataset before making it available for export (first de-identification round). A data provider can achieve this requirement through either third party data de-identification tools or by using the default implementation provided by CHIC (i.e. the Data Upload Tool).

The second de-identification round is performed on the CATS server hosted by Custodix. This service will encrypt the pseudonyms with a key held by a Trusted Third Party.

When the CHIC default tools are used, a CDP (Centre for Data Protection) operative will assist the data provider in creating privacy profiles that define how the, to be uploaded, data files should be de-identified. These privacy profiles are made available to the Upload tool through the CHIC Privacy Profile Store. The data provider can then use the data upload tool to select, de-identify and upload data files.

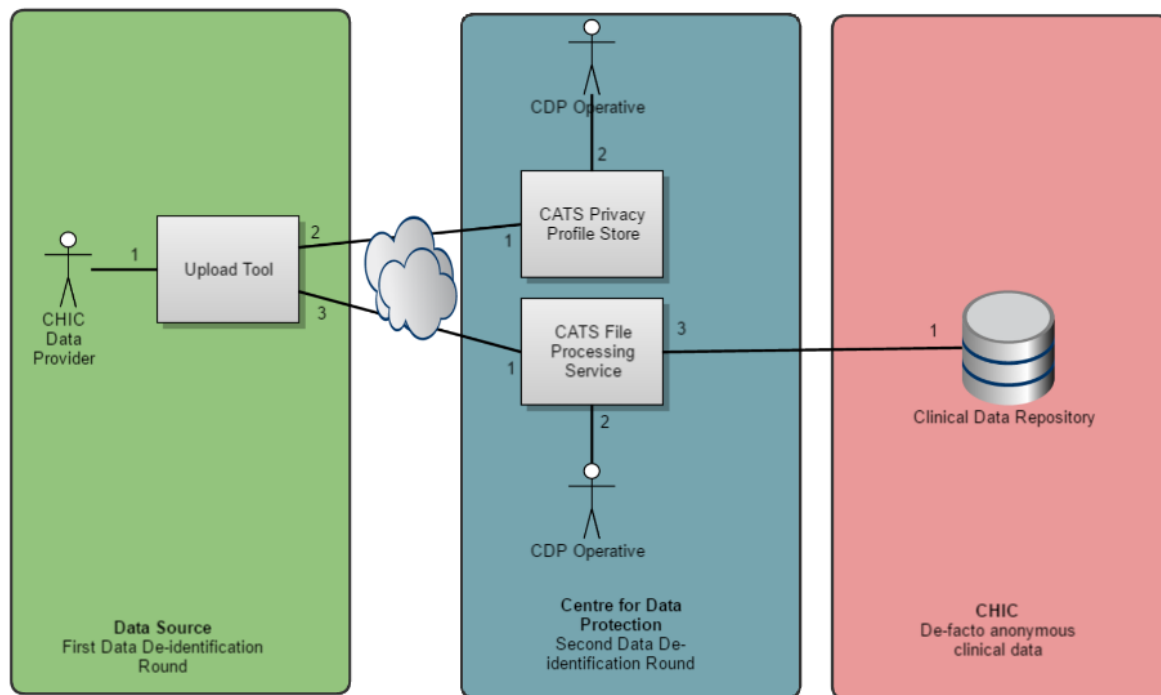


Figure 14 Research Platform Data De-Identification View

4.1.3.1 Upload Tool

The CHIC Data upload Tool provides CHIC data upload functionality to data providers from the hospital to the CHIC research environment. Various data format types are supported such as CSV, DICOM and MIRNA. The data upload tool will perform (if data has not yet been de-identified by the uploader) a first de-identification round by fetching and applying applicable privacy profiles from the CHIC privacy profile store. These privacy profiles define how a given file should be de-identified before it is uploaded into the CHIC research platform.

File History Help													
Upload Upload results													
hsa-let-7c	hsa-let-7c*	hsa-let-7d	hsa-let-7d*	hsa-let-7e	hsa-let-7e*	hsa-let-7f	hsa-let-7f-1*	hsa-let-7f-2*	hsa-let-7g	hsa-let-7g*	hsa-let-7i	hsa-let-7i*	hsa-miR-1
4.7233838...	4.7233838...	11.424745...	4.723383843	4.7233838...	6.544654879	6.359494822	6.5446548...	4.723383843	7.6196362...	7.7940808...	4.7233838...	6.703082941	4.723383843
9.2123146...	7.0120977...	11.790628...	8.418359249	7.3116601...	4.734239033	9.0024427...	4.166071497	3.658014102	7.5436428...	7.012097743	8.81697378	7.7202668...	4.3794021...
8.8013786...	11.340131...	8.514525357	6.638784...	5.704085169	9.0928976...	6.082630241	3.019118671	8.0648267...	6.424074833	9.0928976...	7.4837815...	3.0191186...	8.472305179
8.8521333...	5.8330899...	8.8692833...	5.676882024	5.5033767...	4.858986777	7.44902022	4.166071497	3.005617705	7.3502546...	7.260677989	1.785572...	8.2790788...	3.7588338...
6.8930136...	5.1958141...	8.9008331...	6.205484368	4.5053585...	5.414088181	5.6949632...	3.728155796	3.955066435	4.8805781...	7.253825225	6.5863114	8.3403028...	3.7666300...
10.021368...	6.3483779...	11.087588...	5.172541757	7.2740231...	2.888684219	8.8521333...	7.372382042	2.888684219	8.2220050...	9.928780705	6.640969...	8.5544899...	4.6391284...
3.0035947...	3.8537799...	5.50529086	5.379592981	3.6580141...	5.704085169	1.7083965...	4.076923526	2.557816333	5.2121560...	6.494878661	4.0769235...	8.29548852	4.8433042...
6.7924262...	3.8537799...	10.099861...	8.733381746	7.2538252...	4.378129346	7.5778777...	3.009236826	3.009236826	6.5977643...	6.597764334	7.6579320...	3.3574601...	3.0092368...
5.8668485...	6.6863376...	9.666710...	8.087606125	7.11764331	3.762803205	8.3960311...	3.762803205	3.762803205	3.7628032...	5.866848585	6.9210741...	7.11764331	3.3382304...
7.3116601...	3.766680...	9.4466940...	9.002442774	7.4863171...	5.998360546	8.3754153...	5.998360546	3.7666804...	3.766680...	7.311660123	7.4863171...	3.766680...	9.00829578
8.0876061...	6.71211177	10.500806...	6.885372759	7.6146722...	4.076923526	8.8325689...	6.999355386	5.870598994	8.1725417...	5.870598994	8.7333917...	6.71211177	7.4709164...
9.0808295...	3.2162851...	10.500806...	7.787379607	7.819610...	6.387462072	8.9809662...	4.808509107	4.808509107	8.6354799...	6.0514310...	5.5242612...	6.0514310...	5.2167420...
8.3403028...	5.0466706...	10.021368...	8.254187269	7.1785316...	5.379592981	8.66239472	6.933355665	5.379592981	7.8237219...	6.557006216	8.7434725...	7.8550221...	6.0826302...
9.2657434...	2.2635218	11.087588...	8.66239472	7.7034090...	4.844067734	9.0808295...	3.765135589	3.79927801	7.9107799...	6.355715272	8.72331097	7.9699273...	5.0873465...
6.3644987...	4.0913690...	5.8266503...	4.091369032	4.0913690...	4.091369032	4.0913690...	6.096373112	4.091369032	4.0913690...	6.364498772	4.0913690...	8.7026903...	4.0913690...
2.45721951	6.5750602...	8.2060138...	7.277263767	4.8589867...	5.413542895	4.135428...	5.106951239	2.45721951	2.45721951	7.740924565	5.4135428...	5.5885594...	5.4135428...
2.53699564	5.2543122...	2.8826390...	4.87674568	1.8346213...	5.615486544	4.7336144...	5.087346528	4.082973119	1.8346213...	7.420548013	1.8346213...	9.7576201...	3.5045446...
8.2271846...	2.4691242...	10.218166...	8.66239472	6.0889343...	5.641267736	7.7409245...	4.091369032	9.904890294	7.1285563...	6.088934313	7.0444862...	8.4129045...	2.4691242...
8.7130006...	2.6388251...	10.751009...	8.713000666	7.0444862...	6.348377773	9.0121917...	3.845919257	5.222737492	7.8847575...	7.271934836	8.81697378	8.3960311...	4.1660714...
9.0219408...	3.4581448...	11.284334...	8.320275876	6.9511480...	3.458144882	9.4182591...	6.451356491	3.458144882	7.9937501...	7.290836182	8.9666085...	8.4383027...	3.4581448...

Figure 15 CHIC Data Upload Tool

Interface	Description
1	The data upload tool provides to a CHIC data provider data upload and de-

	identification functionality.
2	<i>No interface exposed to the CATS Privacy Profile Store</i>
3	<i>No interface exposed to the CATS File Processing Service</i>

4.1.3.2 CATS Privacy Profile Store

The CATS Privacy Profile Store is a store of privacy profiles that define how data files should be de-identified. This includes data mapping and the definition of operations on the mapped data such as suppression, generalisation or encryption.

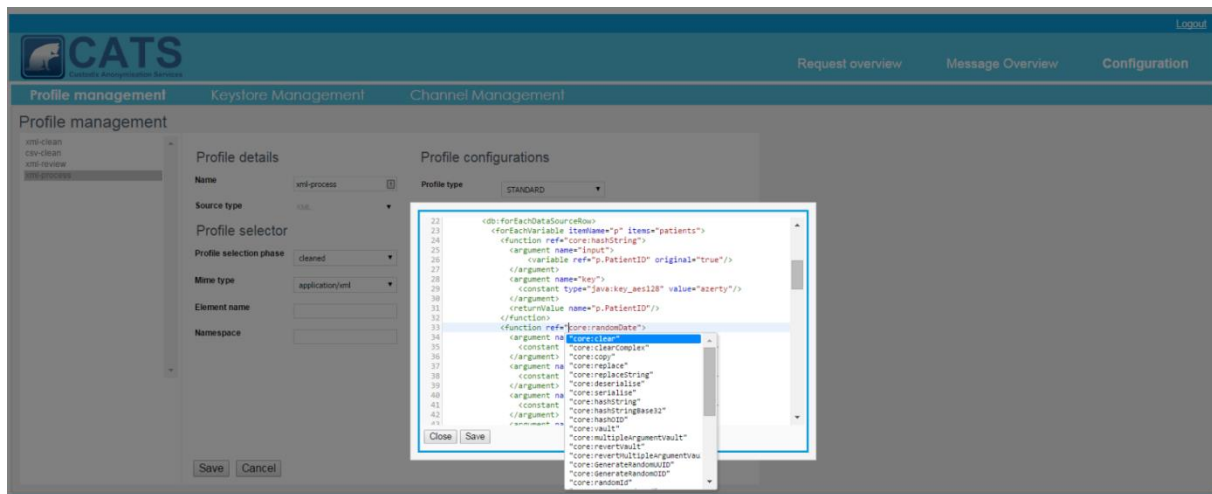


Figure 16 CATS Privacy Profile Editor

Although data providers can create their own data privacy profiles, usually this is a task that will be performed by a CDP operative. Through their privacy expertise they will in collaboration with the data providers create privacy profiles that define in a given dataset the identifiers, quasi-identifiers and sensitive fields and the transformations to be performed.

Interface	Description
1	<p>The CATS Privacy Profile Store exposes the following functionality to the Data Upload Tool:</p> <ul style="list-style-type: none"> • Download matching privacy profiles
2	The CATS Privacy Profile store provides to an operative of the Center for Data Protection the ability to create and manage privacy profiles.

4.1.3.3 CATS File Processing Service

Once de-identified (first round) the upload tool will upload the data files to the CATS File Processing Service for further processing. CATS will fetch privacy profiles that define the second de-identification round (such as encryption of patient pseudonyms by the TTP). After full de-identification CATS requests a CDP operative to approve the de-identified files. Approved files can be

considered de-facto anonymous and are uploaded to the clinical data repository. Once a specific data flow has been validated and fully approved, files can be processed automatically without requiring a CDP operative to intervene and manually approve each file.

Interface	Description
1	The CATS File Processing Service exposes the following functionality to the Data Upload Tool: <ul style="list-style-type: none"> • Upload Data Files • Fetch the status of previously uploaded data files
2	The CATS File Processing Service provides to an operative of the Center for Data Protection the ability to verify data files. An operative of the CDP can verify whether a data file has been sufficiently de-identified before it is uploaded to the clinical data repository.
3	<i>No interface exposed to the clinical data repository</i>

4.1.3.4 Clinical Data Repository (CDR)

The Clinical Data Repository provides authorised CHIC users access to clinical data. In addition, the CDR provides mechanisms for data annotation to ensure that a high level of data quality can be maintained and to allow for easy identification and retrieval of the stored information.

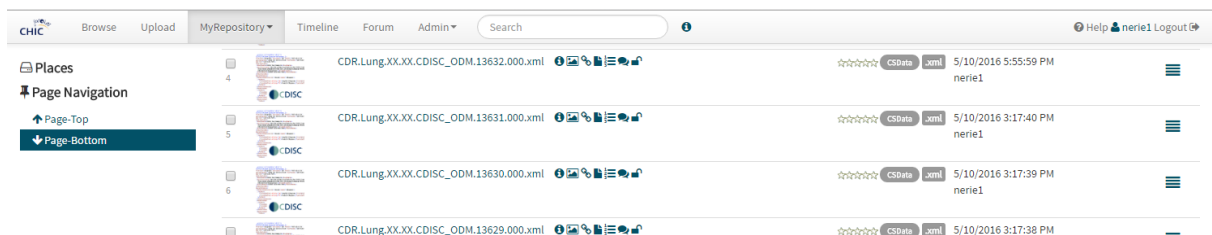


Figure 17 Clinical Data Repository

The CDR does expose a variety of interfaces to end users and other CHIC components. These however are not in scope of this section and therefore are omitted.

Interface	Description
1	<i>No interface exposed to the assertion consumer</i>

4.2 Security Vocabulary

This section provides an overview of the security vocabulary used in the CHIC research environment.

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:custo0dix: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.

5 The CHIC clinical platform

Within the clinical use case, patient's data should only be accessible by the healthcare professionals involved in their treatments such as data nurses and physicians. Through the CHIC access control components such as the authentication and authorisation services it is possible to set up one platform both for research and clinical use and restrict access based on the role of the user accessing the platform. Model developers and other researchers, though, need direct access to the virtual machines in the research platform for development, testing and deployment purposes. In a

production environment direct virtual machine access cannot be tolerated and hence a separate clinical platform has been setup with the following user types:

- Healthcare professionals ranging from, for example, data nurses, to physician assistants, resident physicians and professors.
- Modellers to deploy validated models.
- Administrators with direct virtual machine access for administration and development purposes.

The separate individual CHIC services are not directly accessible by the healthcare professionals. Instead a healthcare professional access the CHIC functionality through CRAF. CRAF is an online (web based) application that provides a “CHIC-in-a-box” unified, simple abstraction for healthcare professionals while, at the same time, demonstrating its full potential and empowering the clinician to use the underlying technologies for the benefit of the patient. CRAF aims to provide a single tool in the clinical domain to run the CHIC models that have been verified for clinical use in order to gain potentially valuable information for the best treatment or diagnosis of their patients. Using CRAF a healthcare professional can enrol new patients, upload data, run models, access experiment results, delete patient and upload patient data to the research platform.

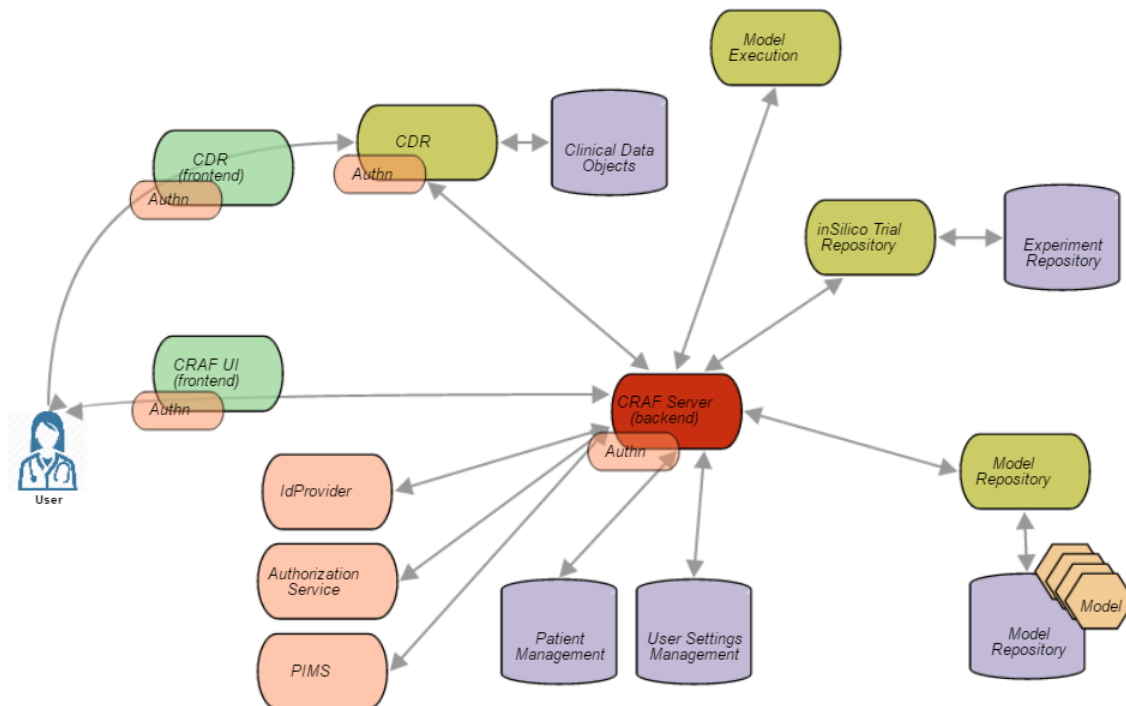



Figure 18 CHIC clinical platform deployment diagram

The figure above depicts the interactions of CRAF with the other components in the clinical platform. More details about CRAF will be provided in Deliverable 10.5 “The CHIC Clinical Research integrated platform”.

Patient Creation

 Patient Card

Patient Card

Patient's Information

Demographics

First Name
 Last Name
 Middle Name

Gender
 Date Of Birth
 Pseudonym
 E-mail

Pseudonym

City
 Region
 Country

Street
 House Number

Post Code
 Mailbox

Other

0/250

SAVE

Figure 19 CRAF Patient Enrollment

5.1 Security View

5.1.1 Authentication

The Authentication View of the clinical platform does not defer from the research platform as described in Chapter 4.1.1.

5.1.2 Authorization

One of the major differences between the research and clinical platform are the authorisation requirements for accessing the platform's resources such as patients. In the clinical environment a patient should only be accessible by the healthcare professionals involved in his/her care. Typically, this will be group of healthcare professionals within a treatment department. More advanced use cases are also supported by the authorisation framework such as for example sharing patient access with a physician from a different hospital department.

A set of the main general security components as defined in Chapter 3 “General Security Framework” is listed in this section.

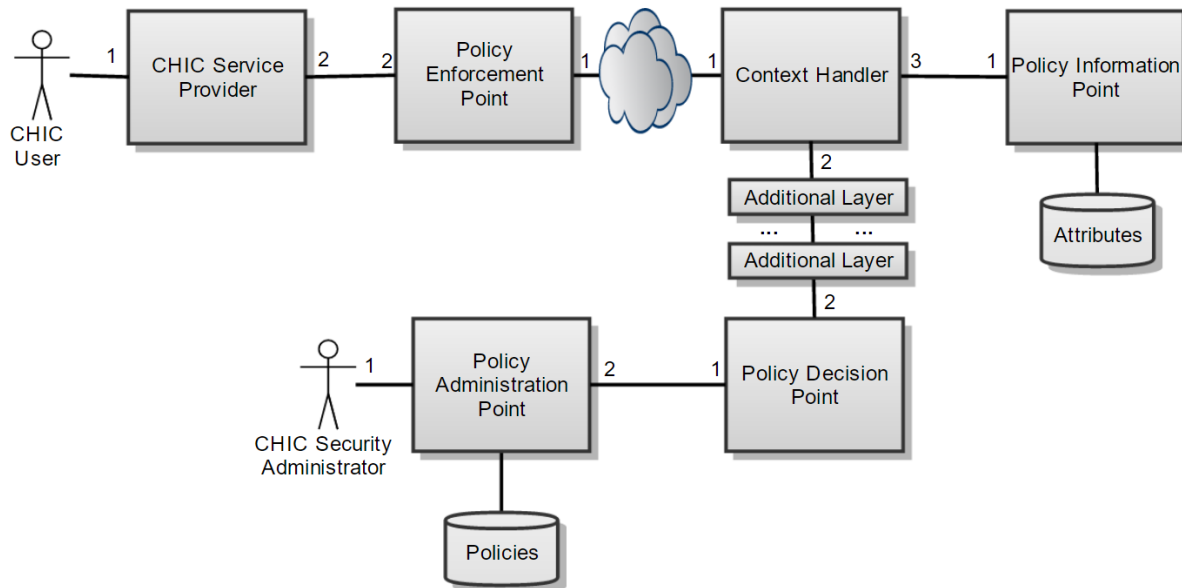


Figure 20: Authorisation security view

5.1.2.1 Service Provider

A service provider represents a secured service in the CHIC clinical platform that is protected by the central CHIC access control.

Interface	Description
1	A CHIC user interacts with the front-end part of a CHIC service. This front-end offers functionality that is protected by access control
2	<i>No interface exposed to the policy enforcement point</i>

5.1.2.2 Policy Enforcement Point (PEP)

When a subject performs an action on a resource in a CHIC service protected by access control, the PEP will intercept this access request. It will trigger the context handler and provides access decision information. After the PDP (Policy Decision Point) has made a decision, the PEP will allow/deny access to the resource depending on the response content.

Interface	Description
1	<i>No interface exposed to the context handler</i>
2	The policy enforcement point exposes following functionality to the service provider:

	<ul style="list-style-type: none"> Enforces policy decision point decisions
--	--

5.1.2.3 Context Handler

The context handler is triggered by the PEP and will generate a decision request containing access information. This request is sent to the PDP where a decision is made and the response(s) is send back to the context handler. The context handler finally pushes this response(s) back to the PEP.

Interface	Description
1	The context handler exposes following functionality to the policy enforcement point: <ul style="list-style-type: none"> Generating requests using the access information coming from the PEP
2	<i>No interface exposed to the policy information point</i>
3	<i>No interface exposed to the policy decision point</i>

5.1.2.4 Policy Decision Point (PDP)

The PDP interprets the requests coming (and generated) from the context handler and evaluates them using the policies that were registered in the PAP (Policy Administration Point) of the CHIC platform. When a decision(s) is made the PDP generates a decision response and sends it back to the context handler.

Interface	Description
1	<i>No interface exposed to the policy administration point</i>
2	The policy decision point exposes following functionality to the context handler: <ul style="list-style-type: none"> Make an access decision for a given authorisation request using the defined policies.

5.1.2.5 Policy Information Point (PIP)

If a request to the PDP contains insufficient information, the PDP can request extra external information for making an access control decision to the PIP (over the context handler). The PIP is connected to the CHIC central user database where the missing attributes can be found.

Interface	Description
1	The policy information point exposes following functionality to the context handler: <ul style="list-style-type: none"> Provides missing attributes to the context handler

5.1.2.6 Policy Administration Point (PAP)

The PAP is responsible for managing the policies of the CHIC authentication framework. For this it offers functionality to generate, maintain, remove and secure these policies that will be used for access control decisions.

Interface	Description
1	An administrator interacts with the front-end part of the policy administration point. This front-end offers functionality to the administrator to generate new policies.
2	The policy administration point exposes following functionality to the policy decision point: <ul style="list-style-type: none"> Provides all policies that are stored in the policy database

5.1.2.7 Additional Layers

Additional layers can be placed between the context handler and the PDP in order to extend the basic implementation of XACML without changing the decision engine of the. The added components manipulate the requests and pass them to the next layer, this process repeats until the request is sent to the PDP. These layers need to have strictly defined interfaces in order to provide a generic solution where new additional layers can be easily fit in.

Patient access authorization decisions are a contextual undertaking. Access to a patient data (read/write/update) depending on the role of the user requesting access within the treatment group the patient is member of. Out of the box XACML does not support such context awareness.

Within CHIC the following solution is implemented to provide context awareness in XACML. An additional layer is added in front of the PDP (see Figure 21), this eliminates the need to extend the PDP itself allowing PDP implementations to be interchanged. It is the responsibility of this layer to handle the context-based roles, the PDP is not aware of the context in which the request is made.

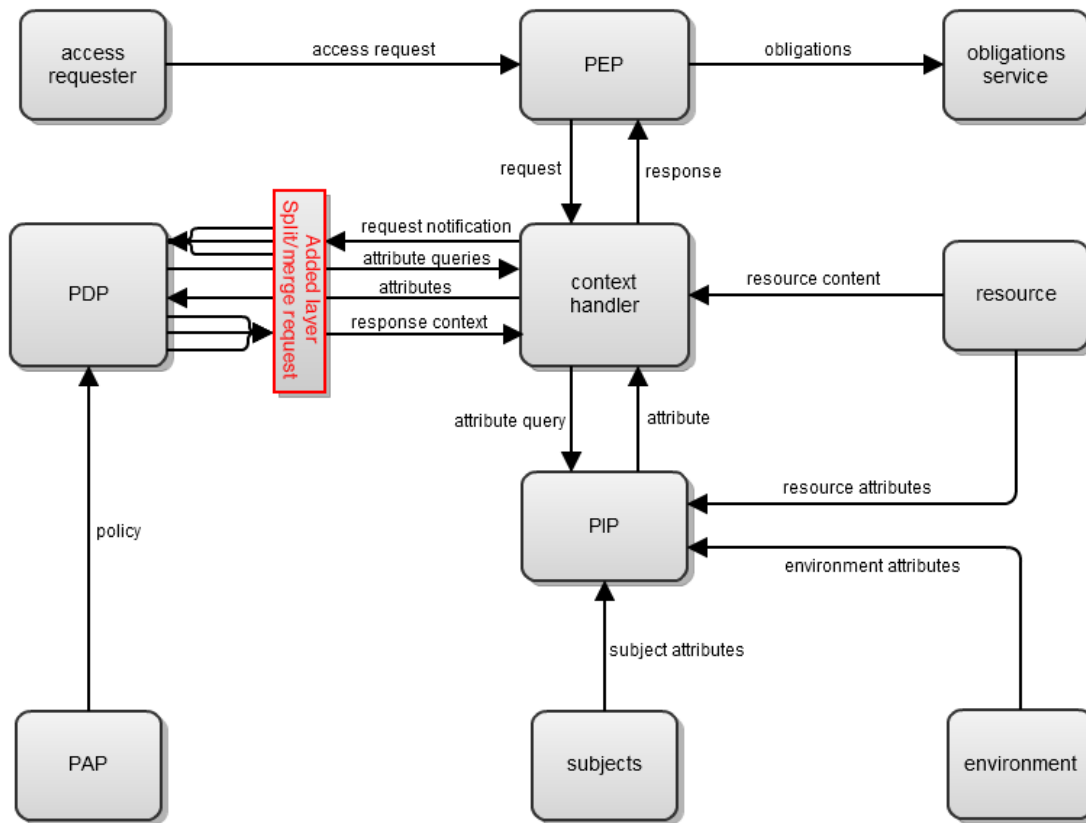


Figure 21 Contextual Attributes Solution

The additional layer (Figure 22) intercepts the requests and responses that are exchanged between the context handler and the PDP (1). The layer splits requests into one general context request and multiple single-context requests, based on the contextual part of the attributes (2). This context can for example be derived from the treatment group to which a patient belongs. It then adds the context of the single-context request to the environment element and sends these requests to the PDP (3). The PDP handles these requests and constructs a response based on context aware policies (4). The responses (5) are sent from the PDP to the additional layer where they are combined (6) to one general response. This general response is finally sent to the context handler.

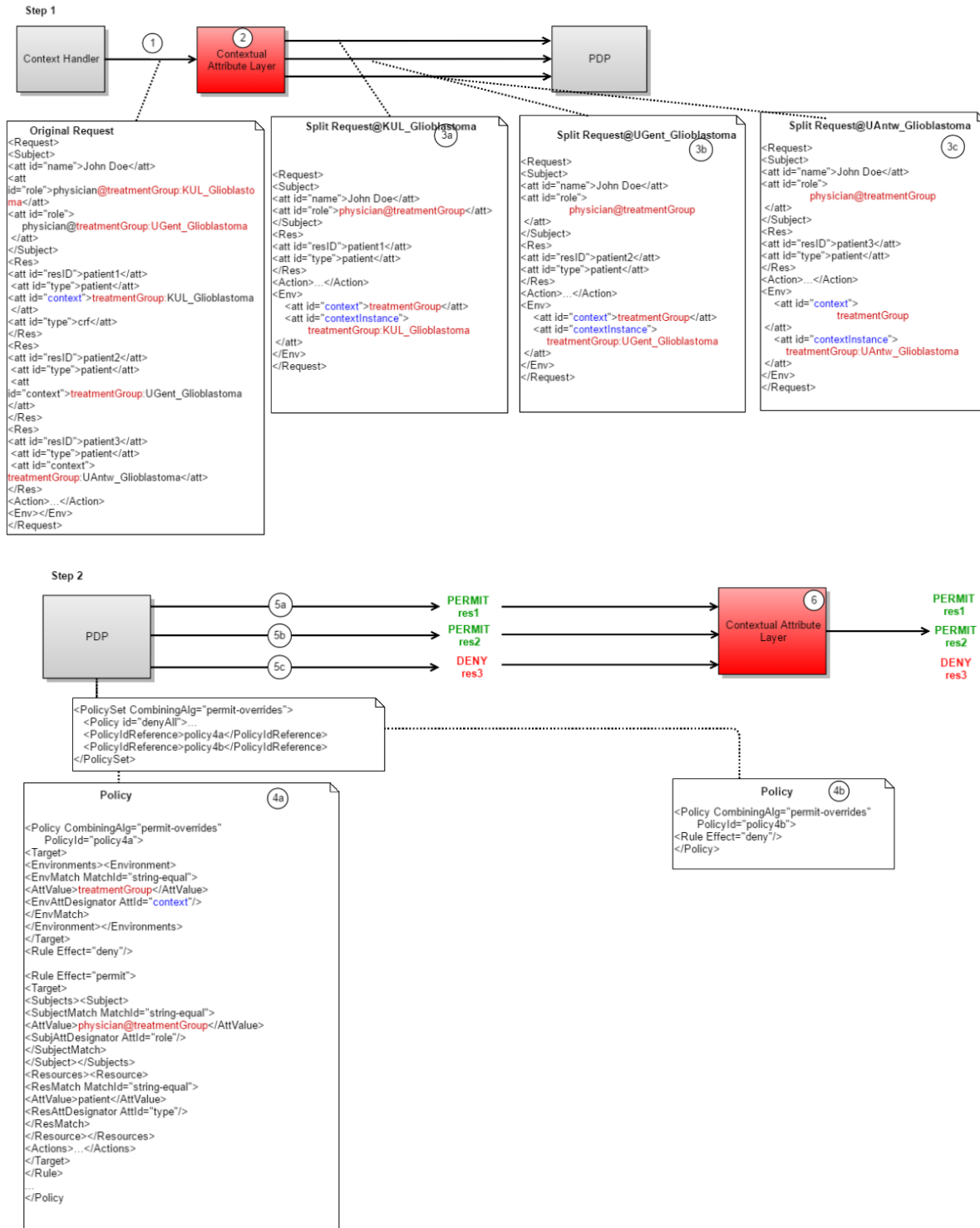


Figure 22: Contextual Attributes Example

The proposed solution for adding extended functionality to the XACML framework, introducing an XACML request proxy in front of the PDP, is generic. Multiple proxy layers can be added before the PDP. The added components manipulate the requests and pass them to the next layer, this process repeats until the request is sent to the PDP. This architecture looks promising because of its

simplicity. However, further research is needed into the possible (unwanted) effects introduced by stacking different “functionalities”.

5.1.3 Data upload and de-identification

The CHIC clinical platform handles patient enrolment and clinical data upload through CRAF.

Patient demographic information provided to CRAF by an authorised healthcare professional during patient enrolment is not stored in the CHIC clinical data repository. Instead all demographic information is stored in a Patient Identity Management System (PIMS). PIMS securely stores demographic information and assigns a pseudonym to the patient. Clinical data on the other hand, also uploaded through CRAF, is stored in the clinical data repository and assigned the pseudonym issued by PIMS. Because of the separation of clinical and demographic information users and system administrators with access to the clinical data repository cannot easily or by accident re-identity a patient through his/her demographics.

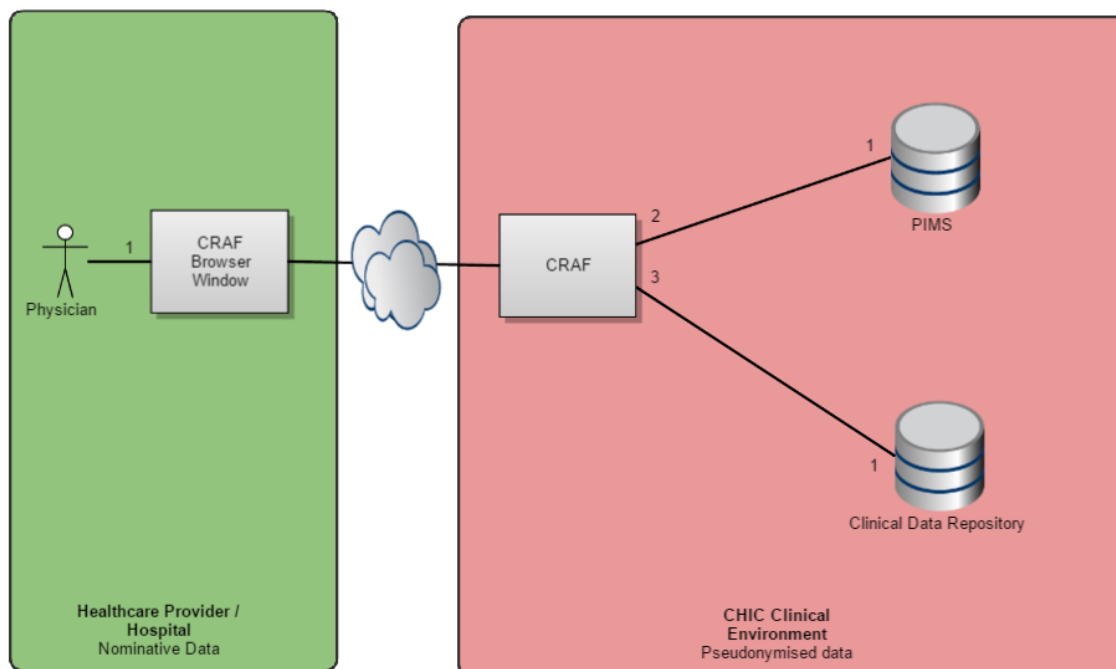


Figure 23 Clinical Platform Data De-Identification/Upload

It is the responsibility of the data uploader (such as the healthcare professional) to make sure all uploaded clinical data files (e.g. DICOM) are sufficiently de-identified (e.g. removal of all direct identifiers). Many repositories within a hospital, such as a DICOM repository, already support the export of de-identified clinical data out of the box. In cases de-identification functionality is not available CHIC provides the tools developed within the CHIC research platform to healthcare professionals to assist them in the data de-identification process.

5.1.3.1 CRAF (Browser Window & back-end application)

In the clinical environment CRAF provides the functionality and web forms for patient creations and the upload of corresponding clinical data such as DICOM, miRNA, or other clinical data. The uploaded data are then forwarded to the CDR while CRAF keeps a reference to the data (using the CDR

provided data identifiers) and the patient's pseudonym. CRAF does not maintain any patient identifiers or demographic information whatsoever.

Interface	Description
1	The CRAF front-end provides to a physician amongst others the following functionality: <ul style="list-style-type: none"> • Patient enrollment • Upload of clinical data such as DICOM & MIRNA • Patient removal
2	<i>No interface exposed to PIMS</i>
3	<i>No interface exposed to CDR</i>

5.1.3.2 PIMS

PIMS (Patient Identity & Access Management) is a master patient index. PIMS will store all patient demographic information and issue pseudonyms. Other CHIC components such as the CDR, Insilico Trial Repository and model exaction framework will refer to patients through their pseudonym instead of using the patient demographics.

As there is currently no need for interlinking patients between treatment departments in the clinical environment, PIMS will not index patients over departments.

Interface	Description
1	PIMS exposes the following functionality to CRAF: <ul style="list-style-type: none"> • Register patient demographics and issue pseudonym • Re-identify a pseudonym into patient demographics

5.1.3.3 Clinical Data Repository (CDR)

The Clinical Data Repository provides authorised CHIC users access to clinical data. In addition the CDR provides mechanisms for data annotation to ensure that a high level of data quality can be maintained and to allow for easy identification and retrieval of the stored information.

The screenshot shows the CHIC web interface. The top navigation bar includes 'Browse', 'Upload', 'MyRepository', 'Timeline', 'Forum', and 'Admin'. A search bar is present. On the left, there is a 'Places' sidebar with 'Page Navigation' options: 'Page-Top' and 'Page-Bottom'. The main content area displays a list of data files under the 'MyRepository' tab. Each entry includes a file icon, a filename (e.g., 'CDR.Lung.XXX.XX.CDISC_ODM.13632.000.xml'), a status icon, and a timestamp (e.g., '5/10/2016 5:55:59 PM'). The user 'nerie1' is logged in.

Figure 24 Clinical Data Repository

The CDR does expose a variety of interfaces to end users and other CHIC components. These however are not in scope of this section and therefore are omitted.

Interface	Description
1	The CDR exposes to CRAF the functionality to upload, download, manage and delete clinical data resources such as DICOM and MIRNA files.

5.2 Security Vocabulary

This section provides an overview of the security vocabulary used in the CHIC research environment.

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.
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

In the clinical environment, contrary to the research one, the CHIC role is contextual. It will define the role of the principal in the context of a treatment department. For example, a clinical user who is physician in the USAAR lung cancer department will have the following role: "clinician@USAARLungDepartement".

6 Security Framework Interoperability Guidelines

6.1 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.

6.2 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.

6.2.1 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 guidelines⁸.

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.

⁸ 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.wssep.mu%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/>

6.2.1.1 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.

6.2.1.2 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.

6.2.1.3 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.

6.2.1.4 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.

6.2.1.4.1 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.

6.2.1.4.2 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.

6.2.1.4.3 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.

6.2.1.5 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.

6.2.1.6 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.

6.2.1.7 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.

6.2.2 REST

CHIC does not specify message level security for restful messages. Instead REST endpoints should always make use of transport level security.

6.3 Passive Authentication Protocol

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 25 Example SAML metadata

7 Security Framework Integration Tutorials

7.1 *Passive (Browser based) Authentication*

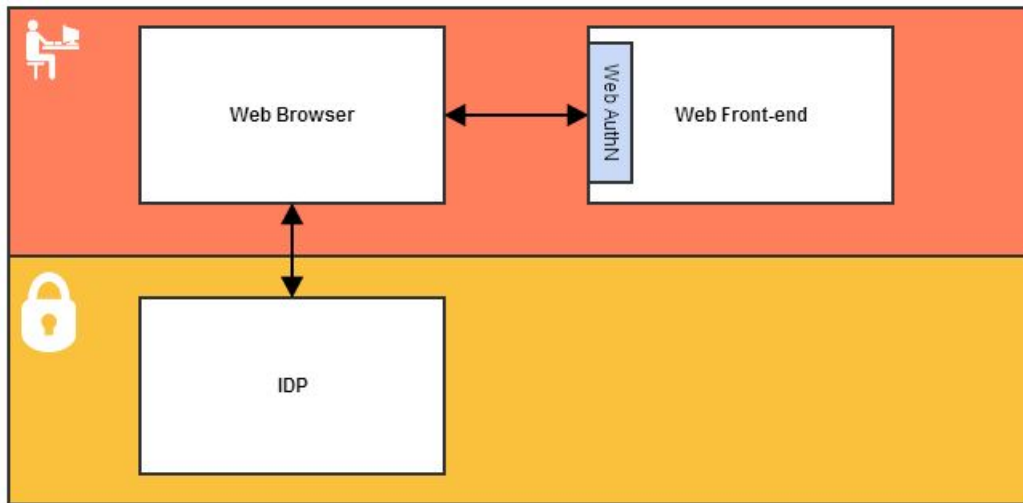


Figure 26 Passive Authentication Profile

7.1.1 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.

7.1.2 Java Based Service Providers through Spring Security

7.1.2.1 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 path⁹:
 - 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

⁹ 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")

7.1.2.2 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 7.1.2.3.

7.1.2.3 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!

7.1.3 PHP Based Service Providers through simplesamlphp

7.1.3.1 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>

7.1.3.2 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

7.1.3.3 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.

7.1.3.4 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.

7.1.4 Integrate Service Providers through a Security Gateway

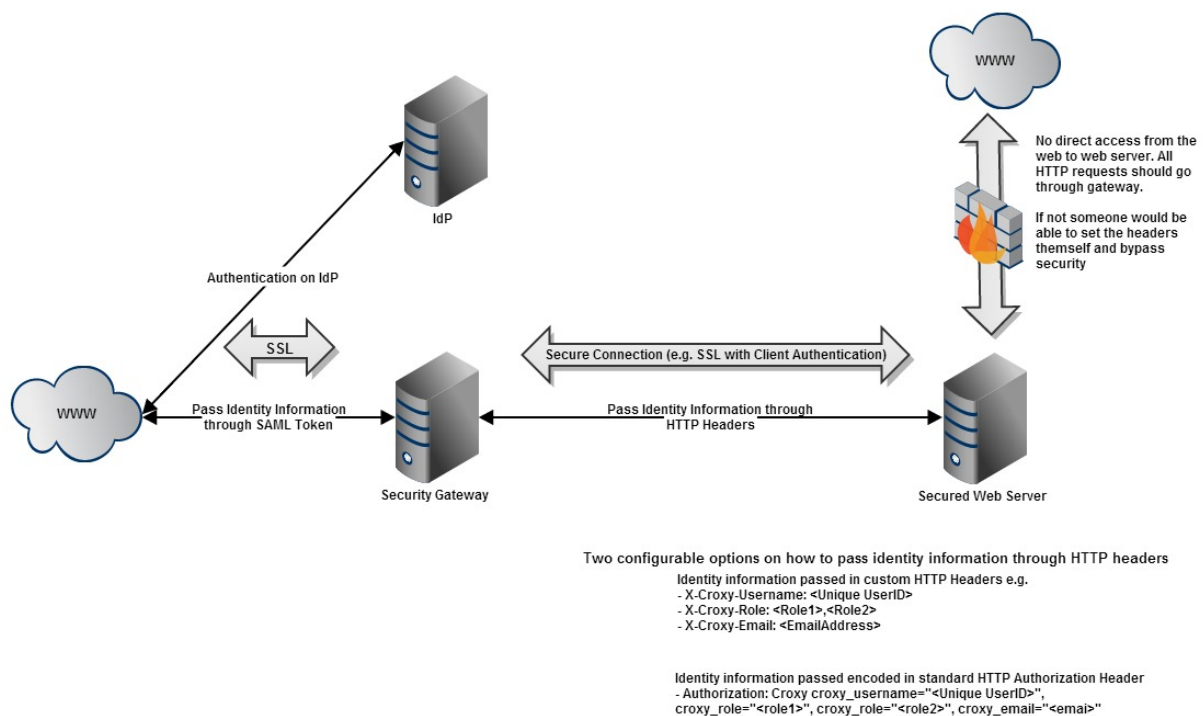


Figure 27 Security Gateway Architecture

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 headers¹⁰.

¹⁰ 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.

7.2 Active Authentication

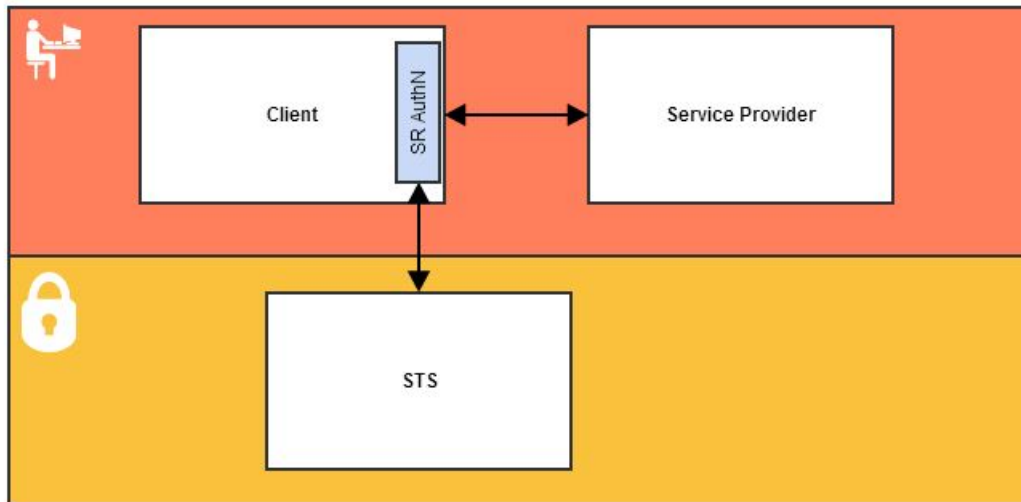


Figure 28 Activate Authentication Profile

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.

7.2.1 XML Based REST or SOAP Client

7.2.1.1 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>

7.2.1.2 Tutorial

7.2.1.2.1 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>

```

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="_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>zDdX8/TEcbUs/mnQZ1oFDuPTlo=</ds:DigestValue>
</ds:Reference>
</ds:SignedInfo>

<ds:SignatureValue>Y0T9sfAPDO8mrWQNsS0uvnew0bMdFcyiCGc9KB6xDIVoQYxjqTNkjR/blqHtgSeVsVwW
h4otF8YyV06L5O/DrZoDmGeUgeQShQXnrt9+Ms1ydcz6xUCEDsdYIHQtTuNePWncLMLL0INwCeiEhYXAOXPL
EMITW0//GIUm8Ce42mmC9w41evUJeY0oGPcJ4++/77E8mcTmKCioUKIoM4FenFMYHffHPCmF3DkqUuv1KiC
MWBHUNrx2dZTajWotxhMy5PnrDLUQ7a6P4yCsbDcbr/gxiGmxINR+/1hoqFb8b3Gxzm03rCOd5QYQRGcl3ae
ezPsxZAYkrZINsvpp7A==</ds:SignatureValue>

<ds:KeyInfo>
  <ds:X509Data>

<ds:X509Certificate>MIIDMTCCAhmGAwIBAgIESX1wTANBgkqhkiG9w0BAQsFADBJMQswCQYDVQQGEwJCR
TERMA8GA1UE
ChMIQ3VzdG9kaXgxDTALBgNVBAsTBENJQU0xGDAWBgNVBAMTD3N0cy1zaWduaW5nLWtleTAeFw0x
MzEwMjMwOTE0MDIaFw0yMzEwMjEwOTE0MDIaMEkxGzAJBgNVBAYTAkFMRwEwDwYDVQQKEwhDdXN0
b2RpeDENMA5GA1UECxEQ0IBTEYMBYGA1UEAxMPc3RzLXNpZ25pbmcta2V5MIIBIjANBgkqhkiG
9w0BAQEFAAOCAQ8AMIIBCgKCAQEAscACfKzbUKyWr1Ggw+KOr7NUhA/WMIbU8ft3Iy18ZoVylG6M
IVLuF3op8xtz2HwbaN8KIDJDzEufXWKZ1a/oLYGK3Fa0V4XpaAqX9Pz5MNEU4hJw+s4rqbybve1
c/DCAF4CR2kQKZMp6G2Yot3SkPCYx10uifjLT749V6buDnDM/YelVPOg210qpTcimhmf7L8o2JSk
Jx9Z6pRCSaji9Nsd1jvi09x5xEDXcJu90OW1V6LBv7CXRb0g6WmXPy4BBWR55r5tPlwrnoGvBO1R
38G40Vg4QeYKW7jF2Waf4j17AgGkNFIG64PckS25vx/i7ulidfWAjIHYrgJFkGPUJwIDAQABoyEw
HzAdBgNVHQ4EFgQUUQVOQ7dNRWriaPzg1s0CbZd3M6fQwDQYJKoZIhvcNAQELBQADggEBADL1zXdf
NMnk57Q2xi04p3VKGecKky8VV/MxUFH/kQfI74hD+SrDvpTFQw0NvcZxs++q7NqhwZfMpu1+fRkb
kUqm9lrmreZVTNqvRuoL8s4DK2w8q19k65ocYIWYRuWVMgvjY9bWu9dbdU8sXGqP9odywPcl3pWf
g7j+BcUIECIIIMphSRAfpq//ZQMbmobBrQplhL2xdP5Pg0jAcgsSw3/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>
```

7.2.1.2.2 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="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#" />
        </ds:Transforms>
        <ec:InclusiveNamespaces PrefixList="xs"
xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#" />
      </ds:Reference>
    </ds:Signature>
  </saml2:Assertion>
```

```

        </ds:Transforms>
        <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
        <ds:DigestValue>zDXX8/TEcbUs/mnQZ1oFDuPT1o=</ds:DigestValue>
        </ds:Reference>
    </ds:SignedInfo>
    <ds:SignatureValue>Y0T9sfAPD08mrWQNsS0uvnew0bMdFcyiCGc9KB6xD1VoQYxjqTNkjR/blqHtgSeVsVw
Wh4otF8YyV06L50/DrZoDmGeUgeQShQQxnrt9+Ms1ydcz6xUCEDsdYlHQttTuNePWncLMLL01NwCeiEhcYXAOXPLEM1TW0/
/GIUm8Ce42mmC9w41evUJeY0oGPcJ4++/77E8mcTmKC10UKIoM4FenFMYHffHPCmF3DkqUuv1KiCMWBHUNrx2dZTajWotx
hMy5PnrDLUQ7a6P4yCsvBdchr/gxiGmxINR+/lhoqFb8b3Gxzm03rC0d5QYQRGcI3aezPsxZAYKrZINsvpp7A==</ds:S
ignatureValue>
    <ds:KeyInfo>
        <ds:X509Data>
            <ds:X509Certificate>MIIDMTCCAhmGAWIBAgIESSX1wTANBgkqhkiG9w0BAQsFADBJMQswCQYDVQQGEwJCRTE
RMA8GA1UEChMIQ3VzdG9kaXgxDALBgNVBAsTBENJQU0xGDAWBgNVBAMTD3N0cy1zaWduaW5nLWtleTAeFw0xMzEwMjMwO
TE0MDI1aFw0YmZlMjEwMjE0MDEkxZAJBgNVBAYTAkFJbG9uYmVhY2FkZmFmREwDwYDVQQKEWhhdXN0b2RpeDENMAsGA1UECjMEQ01BTTE
YMBYGA1UEAxMC3RzLXNpZ25pbmcta2V5MIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAscACfKzBUKyWr1Ggw
+KOr7NUhA/WMLBU8fT3Iy18ZoVyLG6MIVLuF3op8xtz2HwbaN8KIDJDzEufXWKZ1a/oLYGK3Fa0V4XpaAqX9Pz5MNEU4hJ
Jw+s4rqbybve1c/DCAF4CR2kQKZMp6G2Yot3SkPCYx1uifjLT749V6buDnDM/YelVPOg210qpTcimhm7L8o2JSkXj9Z6
pRCSaj9Nsd1jvi09x5EDXcJu90OW1V6LBv7CXRb0g6WmXPy4BBWR55r5tPIwrnoGvBO1R38G40Vg4QeYKW7jF2WaF4j1
7AgGkNf1G64PckS25vx/i7uIdfWAjIHYrgJfKGPuJwIDAQABoyEwHzAdBgNVHQ4EFgQUQVOQ7dNRWriaPzg1s0Cbzd3M6
fQwDQYJKoZIhvcNAQELBQADggEBADL1zXdfNMnk57Q2xi04p3VKGecKky8VV/MxUFH/kQf174hd+SrDvpTFQw0NvcZxs++
q7NqhwZfMpu1+fRkKbUqm9lmreZVTNqvRuoL8s4DK2w8q19k65ocYIWYRuWVMgvjYj9bWu9dbdU8sXGqP9odywPcl3pWfg
7j+BcU1EC1lMphSRAfpq//ZQMbMobBrQpIhL2xdP5Pg0jAcgsSw3/xoNnC/q89wuuULKpBhteDKUroea79DcRgT10K7GQ
fgNs5hbdgK+DxFT08ueI+RnSfzVZq+ktG9upraxF+E1WWB72MSERUB1r121DNyvoPOxsHA8FaNnk88CYx7/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 **Error! Reference source not found.**). 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:

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

	Connection: keep-alive <File ID>
--	---

7.2.2 Java Based Rest Client through Custodix Library

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

7.2.2.1 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>

7.2.2.2 Tutorial

7.2.2.2.1 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("El1a$Ner1");
    }
}, "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.
```

7.2.2.2.2 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>");
```

7.2.3 Java CXF Based SOAP Client

7.2.3.1 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 keystore¹¹ or use the CXF SSL support documentation¹².
- 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 client¹³.

7.2.3.2 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 online¹⁴.

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>
```

¹¹ <https://www.google.be/?q=add+certificate+to+java+cacerts>

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

¹³ <http://cxf.apache.org/docs/client-http-transport-including-ssl-support.html>

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

```

<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
      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: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 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: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 -->
<cx:bus>
  <cx:features>
    <wsp:policies enabled="true"/>
    <wsa:addressing/>
    <cx:logging/>
  </cx:features>
</cx: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.

7.2.3.3 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 online¹⁵.

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>
        </wsp:Policy>
      </sp:TransportBinding>
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

¹⁵ <http://cxf.apache.org/docs/wsdl-to-java.html>

```

        </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
            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 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.setWsdllLocation("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.	helloWorldWSCClient
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.

7.2.4 Visual Studio Based C# SOAP Client

7.2.4.1 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 store¹⁶.
 - 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 msdn¹⁷.

7.2.4.2 Tutorial

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

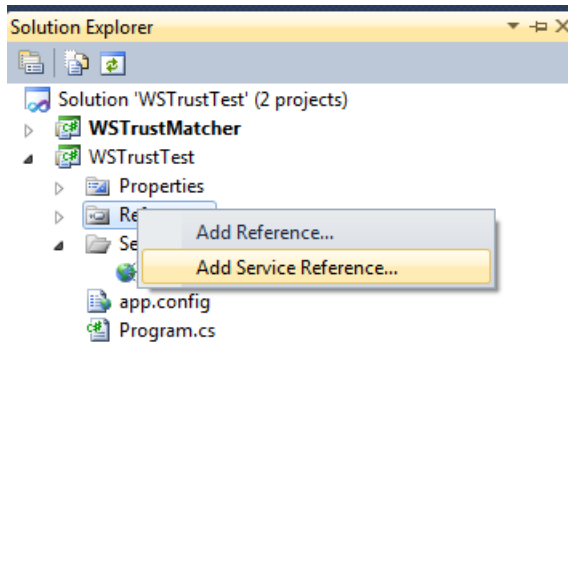
The following library dependencies are required (these are all standard supported in .NET 4.0):

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

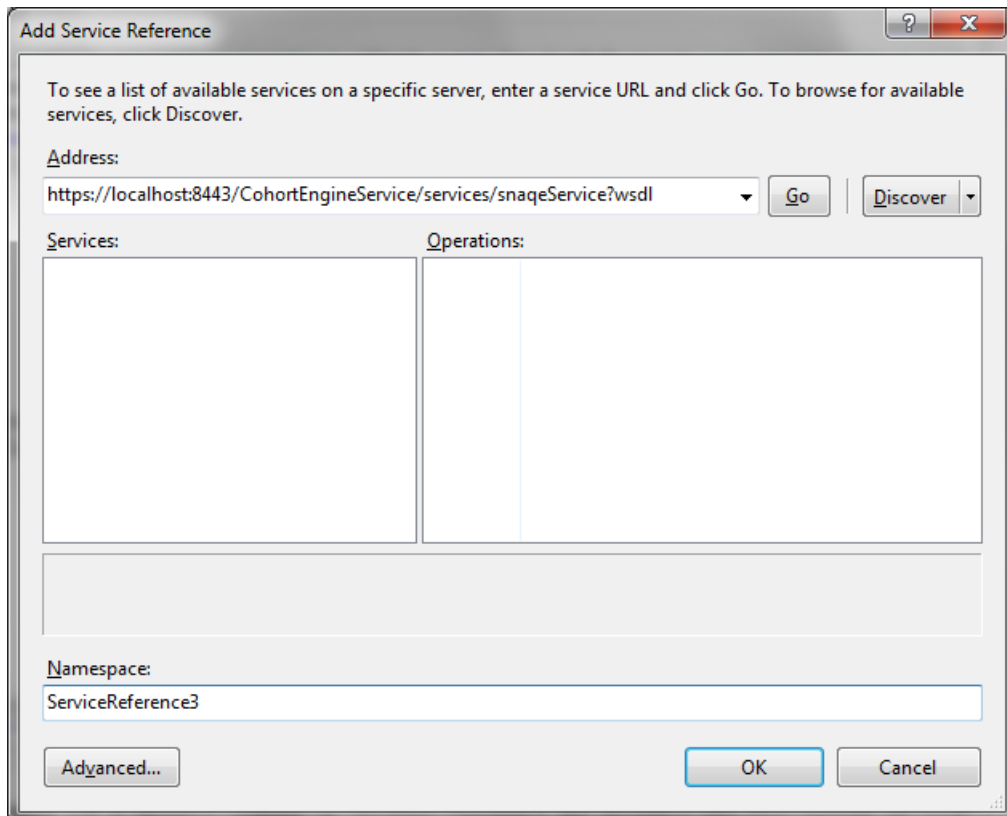
1. Start by adding a new web service reference to the client project (visual studio).

¹⁶ <http://windows.microsoft.com/en-us/windows/import-export-certificates-private-keys#1TC=windows-7>

¹⁷ <http://msdn.microsoft.com/en-us/library/hh924831%28v=vs.110%29.aspx>



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.

3. 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="WSSecurity11WSTrust13WSecureConversation13WSSecurityPolicy12Basic
SecurityProfile10">
  <secureConversationBootstrap />
</security>
<textMessageEncoding messageVersion="Soap11WSAddressing10" />
<httpsTransport />
</binding>

```

4. 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"/>

```

5. 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="WSSecurity11WSTrust13WSecureConversation13WSSecurityPolicy12Basic
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="WSSecurity11WSTrust13WSecureConversation13WSSecurityPolicy12Basic
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(...);
```

7.3 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.

7.3.1 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 29 shows an example of the permission granting process. The services are implemented using the Biomed Town¹⁸ OpenID Identity provider and the Apache mod_auth_tkt module¹⁹.

¹⁸ <http://www.biomedtown.org>

¹⁹ <http://search.cpan.org/~gavinc/Apache-AuthTkt-0.08/AuthTkt.pm>

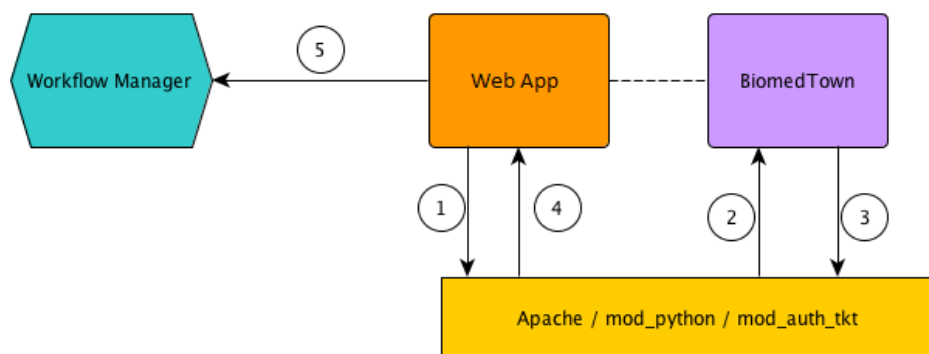


Figure 29 VPH-OP Authentication Service Scheme

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

7.3.2 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.²⁰ Public-key

²⁰ http://en.wikipedia.org/wiki/Public-key_cryptography

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 PhysiomeSpace²¹ using the *pycrypto*²² library. Figure represents the scheme on how the system works between the client and the server side for each data resource.

Upload

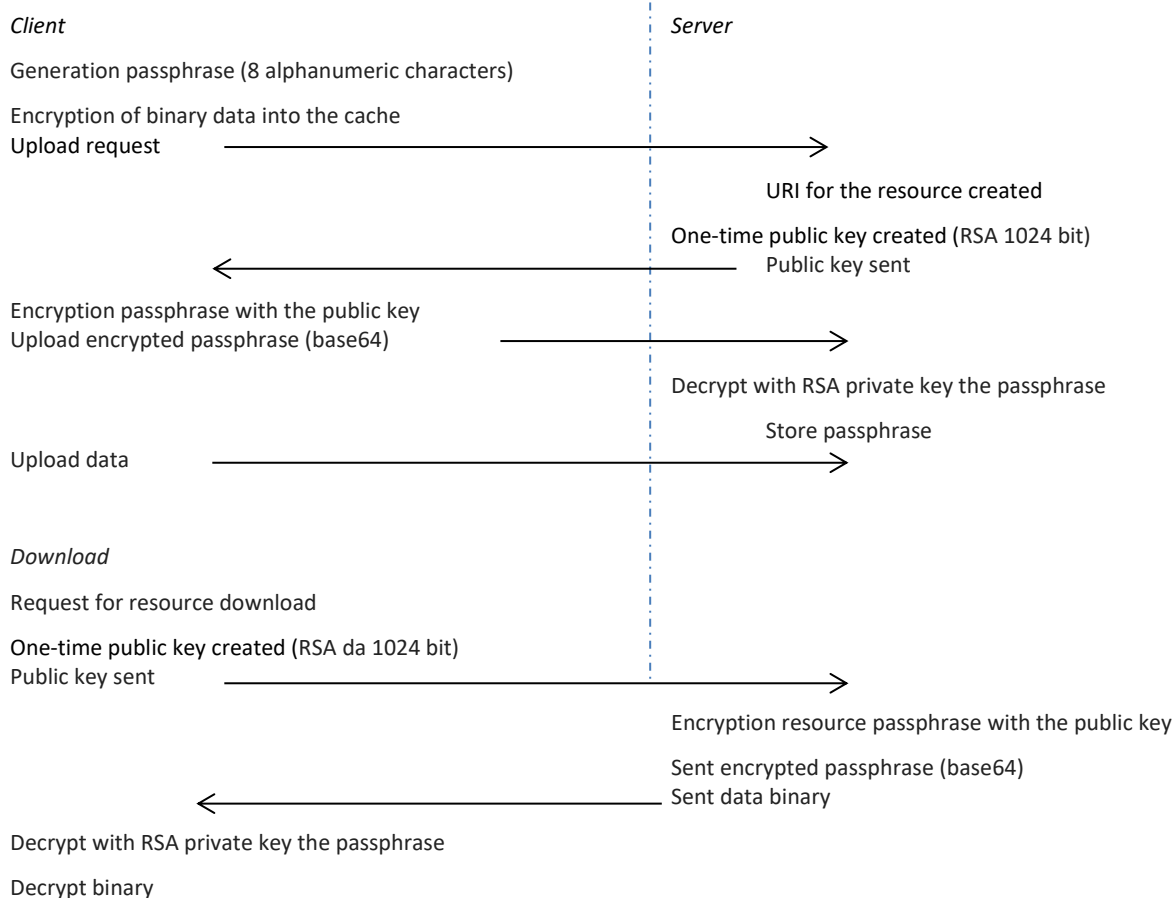


Figure 30 Schematic representation on the encryption mechanism

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.

²¹ <https://www.physiomespace.com>

²² <https://www.dlitz.net/software/pycrypto/>

7.4 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.

7.4.1 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);
}

```

7.4.1.1 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.

7.4.1.2 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.

7.4.1.3 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.

7.4.1.4 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.

8 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"
    }
  }
}
}
}

```