Open Geospatial Consortium

Date: 2013-06-28

External identifier of this OGC® document: <http://www.opengis.net/doc/BP/EOUM/1.1>

Internal reference number of this OGC® document: 07-118r9

Version: 1.1

Category: OGC® Best Practice

Editors: P. Denis, P. Jacques

OGC User Management Interfaces for Earth Observation Services

Warning

This document is not an OGC Standard. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an OGC Standard.

Recipients of this document are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

Document type:	OGC® Best Practice
Document subtype:	interface
Document stage:	Draft proposed version 1.1
Document language:	English

Contents

Figures

Tables

i. Abstract

This document describes how user and identity management information may be included in the protocol specifications for OGC Services. The proposed approach is applicable to the orchestration of EO services, to system of systems and federation scenarios. The approach is meant to be independent from the specific OGC service. The use cases potentially addressed are very wide and in general may cover geospatial services and not only EO (Earth Observation) services. The use cases may range from web map, feature or coverage services, web processing services, to catalogue services. Examples of EO specific use cases are: ordering (Ordering Services for Earth Observation Products [OGC 06-141r6]) and feasibility analysis (OpenGIS Sensor Planning Service Application Profile for EO Sensors [OGC 10-135]).

The document was initially produced during the ESA HMA (Heterogeneous Missions Accessibility) initiative [OR1] and related projects.

This document is not an OGC standard. This document describes how existing specifications from W3C and OASIS can be used in combination to pass identity information to OGC Web services.

ii. Keywords

The following are keywords to be used by search engines and document catalogues

ogcdoc, Identity, STS, RST, Token, SSO

iii. Preface

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The Open Geospatial Consortium shall not be held responsible for identifying any or all such patent rights.

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the standard set forth in this document, and to provide supporting documentation when possible.

iv. Submitting organisations

The following organisations will submit the original document or its revisions to the OGC™ Security Working Group.

- **Spacebel s.a.**
- **ESA – European Space Agency**
- **Intecs**
- **STFC**
- **con terra**
- **Eumetsat**

The editors would like to acknowledge that this work is the result of collaboration and review of many organisations and would like to thank for the comments and contributions from:

- **Astrium**
- **Spot Image**
- **ASI**
- **CNES**
- **DLR**
- **EUSC**
- **MDA**
- **Terradue**
- **Rhea System**
- **Oracle**
- **Siemens**
- **Logica CGI**

Note: this does not imply a complete endorsement by these organisations.

v. Submitters

All questions regarding this submission should be directed to the editor or the submitters:

Current contributors:

Contact	Organisation	Email
Pierre Denis	Spacebel	pierre.denis@spacebel.be
Patrick Jacques	Spacebel	patrick.jacques@spacebel.be
Maria Rosaria Barone	Intecs	mariarosaria.barone@intecs.it
Stefano Puri	Intecs	stefano.puri@intecs.it
Uwe Voges	con terra	u.voges@conterra.de
Michael Schick	Eumetsat	michael.schick@eumetsat.int
Andrea Baldi	ESA	andrea.baldi@esa.int
Christoph Reck	DLR	christoph.reck@dlr.de

Previous contributors:

Contact	Organisation	Email
Andrew Woolf	STFC	andrew.woolf@stfc.ac.uk
Rowena Smillie	Spacebel	rowena.smillie@spacebel.be
Alexandre Cucumel	Spacebel	-
Wouter Van de Weghe	Oracle	wouter.van.de.weghe@oracle.com

1 Scope

This proposed interface document describes the interfaces required to authenticate and authorise users in a federated system of OGC Web Services for Earth Observation. The document has been written with three high level scenarios in mind:

- The orchestration of OGC Web Services as it may occur when (e.g.) Sensor Planning Service, Web processing Service and Web Coverage Service are provided by several cooperating organizations.
- The system of systems of OGC Web Services as it may occur when several organisations may concur and cooperate in the provision of instances of the same service within a federated service provision. Several relevant use cases are proposed within the GEOSS AIP.
- The security and EO products market scenarios which have high level requirements related to the user authentication as well as to the authorisation to the use of the OGC Web Services over geospatial (e.g. area of interest) and/or temporal parameters.

The purpose of this document is to describe how:

HL-REQ010: To perform user authentication (and authorisation) for the use of existing OGC Web Services (i.e. without changes to published OGC standards).

HL-REQ020: To use OASIS and W3C already defined standards for authentication and authorisation of OGC Web Services.

HL-REQ030: To federate different user communities allowing cross authentication for the purpose of using OGC Web Services.

HL-REQ040: To perform authentication and authorisation across orchestrated OGC Web Services.

HL-REQ050: To perform authentication and authorisation across a “system of systems” based on OGC Web Services.

HL-REQ060: To map a C2B Web-SSO type authentication environment (e.g. Shibboleth or OpenAM) with a B2B service authorisation environment based on SAML tokens.

Hereafter a brief outline of the document content allows readers to jump directly to the topic of their interest:

- the authentication use cases with the use of the SOAP or HTTP bindings is addressed in the Chapters 6 and 7;
- the mapping of an authentication environment based on Web-SSO with one based on SAML as required by HL-REQ060 above is addressed in Chapter 8;
- security considerations linking selected threats and risks to proposed countermeasures are addressed in Chapter 9;
- the authorisation use cases and the possible link with XACML and GeoXACML are addressed in Chapter 10.

2 Conformance

2.1 Conformance to base specifications

This present section describes the compliance testing required for an implementation of this Best Practice.

It is worth highlighting that this OGC document references and uses specifications (SAML, WS Security, XACML) that come from other organizational bodies (such as the Organization for the Advancement of Structured Information Standards - OASIS) for which the concept of "conformance testing" does not apply; consequently, it is not possible to recursively test the conformance to the compound specifications.

2.2 Conformance classes

We assume that a unique "core" conformance class encompassing all of the specification clauses in the Best Practice is defined and assume that the "Abstract Test Suite" is made up of this unique conformance class ("the core"). This class defines test cases, which covers:

- Test Module Basic requirements
- Test Module Authentication
- Test Module Authorisation

These are detailed in the Abstract Test Suite (see Annex A).

3 References

The following normative documents contain provisions that, through reference in this text, constitute provisions of this document. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. For undated references, the latest edition of the normative document referred to applies.

3.1 Normative references

[NR1] W3C Recommendation January 1999, Namespaces In XML, http://www.w3.org/TR/2000/REC-xml-names

[NR2] W3C Recommendation 6 October 2000, Extensible Markup Language (XML) 1.0 (Second Edition), http://www.w3.org/TR/REC-xml

[NR3] W3C Recommendation 2 May 2001: XML Schema Part 0: Primer, http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/

[NR4] W3C Recommendation 2 May 2001: XML Schema Part 1: Structures, http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/

[NR5] W3C Recommendation 2 May 2001: XML Schema Part 2: Datatypes, http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/

[NR6] W3C Simple Object Access Protocol (SOAP) Version 1.1 W3C Note 08 May 2000, http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

[NR7] WSDL, Web Services Description Language (WSDL) 1.1, http://www.w3.org/TR/wsdl

[NR8] IETF RFC 2119, Keywords for use in RFCs to Indicate Requirement Levels, http://rfc.net/rfc2119.html

[NR9] WS-Security, SOAP Message Security V1.1 http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf

[NR10] SAML, Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V1.1 http://www.oasis-open.org/committees/download.php/3406/oasis-sstc-saml-core-1.1.pdf

[NR11] Web Services Security SAML Token Profile 1.1 http://www.oasis-open.org/committees/download.php/16768/wss-v1.1-spec-os-SAMLTokenProfile.pdf

[NR12] SAML, Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0 http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf

[NR13] Secure Hash Standards (SHA-1) National Institute of Standards and Technology http://csrc.nist.gov/cryptval/shs.htm

[NR14] Glossary for the OASIS Security Assertion Markup Language (SAML) http://www.oasis-open.org/committees/security/docs/cs-sstc-glossary-01.pdf

[NR15] Java Cryptography Architecture API Specification & Reference

http://java.sun.com/j2se/1.5.0/docs/guide/security/CryptoSpec.html

[NR16] OGC 06-121r8, OGC Web Services Standard, Implementation Standard, Version 1.2.0, 2009/09/18

[NR17] XML encryption http://www.w3.org/TR/xmlenc-core/

[NR18] XML signature http://www.w3.org/TR/xmldsig-core/

[NR19] Apache XML Security http://santuario.apache.org/Java/index.html

[NR20] W3C Recommendation 04 September 2007, Web Services Policy 1.5 - Framework, http://www.w3.org/TR/ws-policy/

[NR21] OASIS eXtensible Access Control Markup Language (XACML) TC http://www.oasis-open.org/committees/xacml

[NR22] SOAP Version 1.2 Part 1: Messaging Framework (Second Edition), W3C Recommendation 27 April 2007, http://www.w3.org/TR/soap12-part1/

[NR23] OASIS WS-Trust 1.3 http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.pdf

[NR24] OASIS WS-Security UsernameToken Profile 1.1 http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-UsernameTokenProfile.pdf

[NR25] OGC 07-026r2, Geospatial eXtensible Access Control Markup Language (GeoXACML), 1.0

[NR26] Web Services Federation Language (WS-Federation) Version 1.2 http://www.oasis-open.org/apps/group_public/download.php/31658/ws-federation-1.2-spec-cs-01.doc

[NR27] IETF RFC 2616, Hypertext Transfer Protocol -- HTTP/1.1 http://www.ietf.org/rfc/rfc2616.txt

[NR28] IETF RFC 2045, Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies http://www.ietf.org/rfc/rfc2045.txt

[NR29] IETF RFC 3986, Uniform Resource Identifier (URI): Generic Syntax http://www.ietf.org/rfc/rfc3986.txt

[NR30] IETF RFC 2234, Augmented BNF for Syntax Specifications: ABNF http://www.ietf.org/rfc/rfc2234.txt

[NR31] IETF RFC 6750, The OAuth 2.0 Authorization Framework: Bearer Token Usage http://www.ietf.org/rfc/rfc6750.txt

[NR32] W3C Recommendation 04 September 2007, Web Services Policy 1.5 – Attachment, http://www.w3.org/TR/ws-policy-attach/

[NR33] OASIS WS-SecurityPolicy 1.2, 1 July 2007 http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/ws-securitypolicy-1.2-spec-os.pdf

3.2 Other references

[OR1] Heterogeneous Missions Accessibility – Design Methodology, Architecture and Use of Geospatial Standards for the Ground Segment Support of Earth Observation missions ESA TM-21 http://www.esa.int/About_Us/ESA_Publications/ESA_TM-21_Heterogeneous_Missions_Accessibility

[OR2] Shibboleth http://shibboleth.net/

[OR3] OpenAM http://openam.forgerock.org/

4 Terms and definitions

For the purposes of this document, the following terms and definitions apply:

4.1.
Authentication [NR14]
Verification that a potential partner in a conversation is capable of representing a person or organization.

4.2.
circle of trust
A federation of Service Providers and identity providers within which Service Providers accept the authentication asserted by the identity provider.

4.3.
Claim
A statement made about a client, service or other resource (e.g. name, identity, key, group, privilege, capability, etc.).

4.4.
client
Software component that can invoke an **operation** from a **server** i.e. a service consumer.

4.5.
identifier
A character string that may be composed of numbers and characters that is exchanged between the client and the server with respect to a specific identity of a resource.

4.6.
identity provider [NR14]
A kind of Service Provider that creates, maintains, and manages identity information for principals and provides principal authentication to other Service Providers within a federation, such as with Web browser profiles.

4.7.
interface
Named set of operations that characterise the behaviour of an entity [ISO 19119].

4.8.
operation
Specification of a transformation or query that an object may be called to execute [ISO 19119].

4.9.
parameter
Variable whose name and value are included in an operation **request** or **response**.

4.10.
PEP
Policy Enforcement Point.

4.11.
principal [NR14]
A system entity whose identity can be authenticated.

4.12.
Relying Party [NR26]
A Web application or service that consumes **Security Tokens** issued by a **Security Token Service**.

4.13.
request
Invocation of an **operation** by a **client**.

4.14.
response
Result of an **operation,** returned from a **server** to a **client**.

4.15.
Security Token
A collection of **claims**. In the present Best Practice, the so-called "SAML token" is a specific kind of security token where the claims are SAML assertions.

4.16.
Security Token Service
A security token service (STS) is a Web service that issues security tokens. The generation of a security token may be delegated by the Delegating STS to a Delegate STS.

4.17.
server service instance
A particular instance of a **service** [ISO 19119].

4.18.
service
Distinct part of the functionality that is provided by an entity through interfaces [ISO 19119].

Capability which a Service Provider entity makes available to a service user entity at the interface between those entities [ISO 19104 terms repository].

4.19.
service interface
Shared boundary between an automated system or human being and another automated system or human being [ISO 19101].

4.20.
Service Provider [NR14]
A role donned by a system entity where the system entity provides services to principals or other system entities.

4.21.
transfer protocol
Common set of rules for defining interactions between distributed systems [ISO 19118].

5 Conventions

This sections provides details and examples for any conventions used in the document. Examples of conventions are symbols, abbreviations, use of XML schema, or special notes regarding how to read the document.

5.1 Symbols (and abbreviated terms)

Some frequently used abbreviated terms:

ABNF	Augmented Backus-Naur Form
ATC	Abstract Test Case
ATS	Abstract Test Suite
B2B	Business to Business
C2B	Consumer to Business
DAIL	Data Access Integration Layer
EO	Earth Observation
GEOSS	Global Earth Observation System of Systems
GMES	Global Monitoring for Environment and Security
GSCDA	GMES Space Component Data Access
HMA	Heterogeneous Missions Accessibility
HTTP	HyperText Transfer Protocol
IdP	Identity Provider
ISO	International Organisation for Standardisation
KVP	Keyword Value Pair
LDAP	Lightweight Directory Access Protocol
OASIS	Organization for the Advancement of Structured Information Standards
OGC	Open Geospatial Consortium
PDP	Policy Decision Point
PEP	Policy Enforcement Point
RST	Request Security Token
RSTR	Request Security Token Response
SAML	Security Assertion Markup Language
SOAP	Simple Object Access Protocol
SP	Service Provider
SSO	Single Sign-On
STS	Security Token Service

TLS	Transport Layer Security
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
URN	Uniform Resource Name
WS	Web Service
WSDL	Web Service Definition Language
W3C	World Wide Web Consortium
XACML	eXtensible Access Control Markup Language
XML	eXtensible Markup Language

5.2 Document terms and definitions

This document uses the specification terms defined in Subclause 5.3 of [NR16].

6 System context

This section documents special requirements and describes the context of use.

6.1 Application domain

Web service requests are received by Service Providers. These Service Providers should be able to identify who issued the request and react accordingly. The following approach is proposed:

1) A Security Token Service (STS) provides a Request Security Token operation (RST), which returns a SAML token, an artefact representing an authenticated user. Depending whether the STS is in charge of authentication or not, two main cases are defined:

    a. The STS *is* in charge of authentication: the RST contains user identifier, password and optionally his identity provider. This service may delegate the authentication and SAML token generation to another STS acting as an identity provider.

    b. The STS *is not* in charge of authentication, i.e. this is taken in charge by an external IdP (Web-SSO), which is trusted by the STS: the RST just contains a user identifier (no password); it shall be signed in order to check that the requester is trusted. This service may delegate the SAML token generation to another STS.

2) Each subsequent service request by the client (Web service consumer) should include the SAML token as described later in this document.

3) Each Service Provider accepts service requests only via an Authorisation Service or "Policy Enforcement Point" (PEP). The PEP first checks the existence of SAML token and decrypts it.

4) The PEP verifies the SAML token (signature and expiry time)

5) The PEP decides based on the content of the message body, the contents of the message header (including authentication token) and the context (i.e. applicable policies) whether to accept or to refuse the service request or to reroute it. Although this is not imposed, the use of XACML [NR21] or GeoXACML [NR25] for definition of policy rules is recommended.

6) If the request is authorised, then the request is forwarded to and processed by the target SP.

If any of the steps from 3) to 5) fails, then a fault response is returned to the client.

The full authentication & authorisation process is detailed in the following figure. This figure highlights the typical sequence of steps from authentication to request authorisation and processing (the two authentication cases are here abstracted).

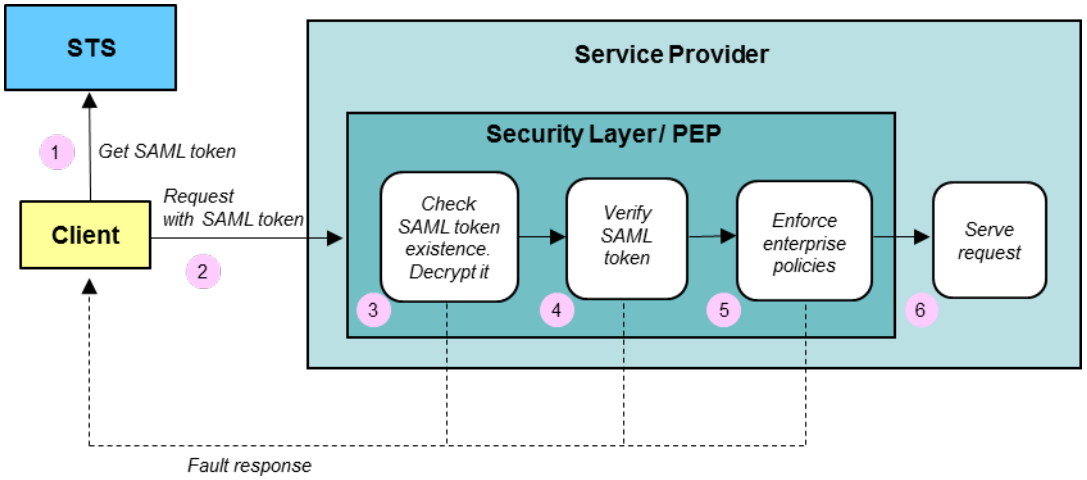


Figure 1: Sequence of authentication/authorisation activities

The distinction between steps 1.a. and 1.b, which discriminate on the IdP responsibility, is depicted in the following diagram (client A authenticates on STS, client B authenticates on external IdP).

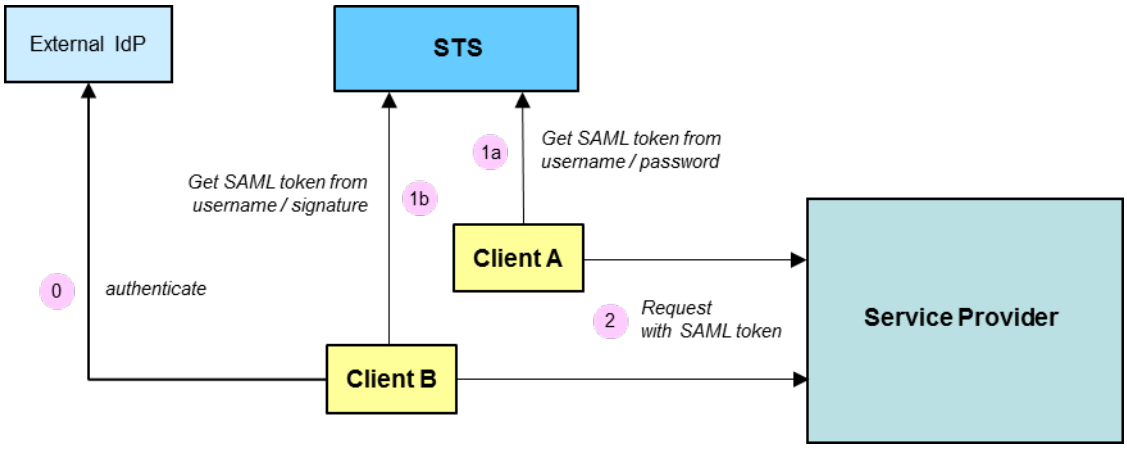


Figure 2: Two cases of authentication

These two cases are refined and detailed in section 6.4.3.

6.2 Protocol bindings

Two protocol bindings are covered in the present Best Practice:

- the *SOAP binding* : SOAP 1.2 messages[1] [NR22], transported through the POST method of HTTP(S) 1.1 request, with document/literal style,
- the *HTTP binding* : HTTP(S) 1.1 request [NR27], whatever the method (e.g. GET, HEAD, POST, PUT, DELETE); this shall be used in particular for REST-compliant Web services.

An **authentication request** shall be a WS-Trust's RST, that is an XML wst:RequestSecurityToken element. The RST and its response (RSTR) are transported according to the protocol binding (see section 7.1):

- for the *SOAP binding*, the RST and RSTR are put in the body part of the SOAP envelope, which is transmitted on HTTPS; the header part of the RST's SOAP envelope may be used to include a detached signature (see below);
- for the *HTTP binding*, the RST is put in the message body of an HTTPS POST request. The HTTPS header may be used to include a detached signature (see below).

Note: The protocol binding used for authentication by an external trusted IdP (Web-SSO) is irrelevant to this Best Practice document.

A **service request** uses its native protocol binding, while attaching the SAML token in a non-invasive way (see section 7.2):

- for a *SOAP binding*, the SAML token is included in the header part of the SOAP envelope;
- for an *HTTP binding*, the SAML token is included in the HTTP header.

The format of the request and response, for both types of request and both protocol bindings, are detailed in the next sections.

6.3 Basic use case

The use cases covered by this Best Practice are shown in the following sequence diagram:

- Authentication: A Request Security Token (RST) is first issued to the Security Token Service (STS). Four authentication use cases are described in section 6.4.3 but only the first one is discussed here.
- Authorisation: A service request is then sent to the Service Provider (SP). The service requests can be synchronous or asynchronous via WS-Addressing. This is transparent for the purposes of this Best Practice.

In all the use cases presented in the document, the "Client" is the service consumer that issues requests to the STS or SP. It is typically not the front-end application used by the human user (e.g. Web browser); this front-end application accesses the client of the STS or SP, but is typically not by itself such client. This remark is especially important for authentication use cases relying on "trusted clients" (see 6.4.3.3 and 6.4.3.4).

[1] The optional coverage of SOAP 1.1 is described in Annex C.

The policy enforcement on the SP is non-invasive meaning that it is independent of the SP implementation.

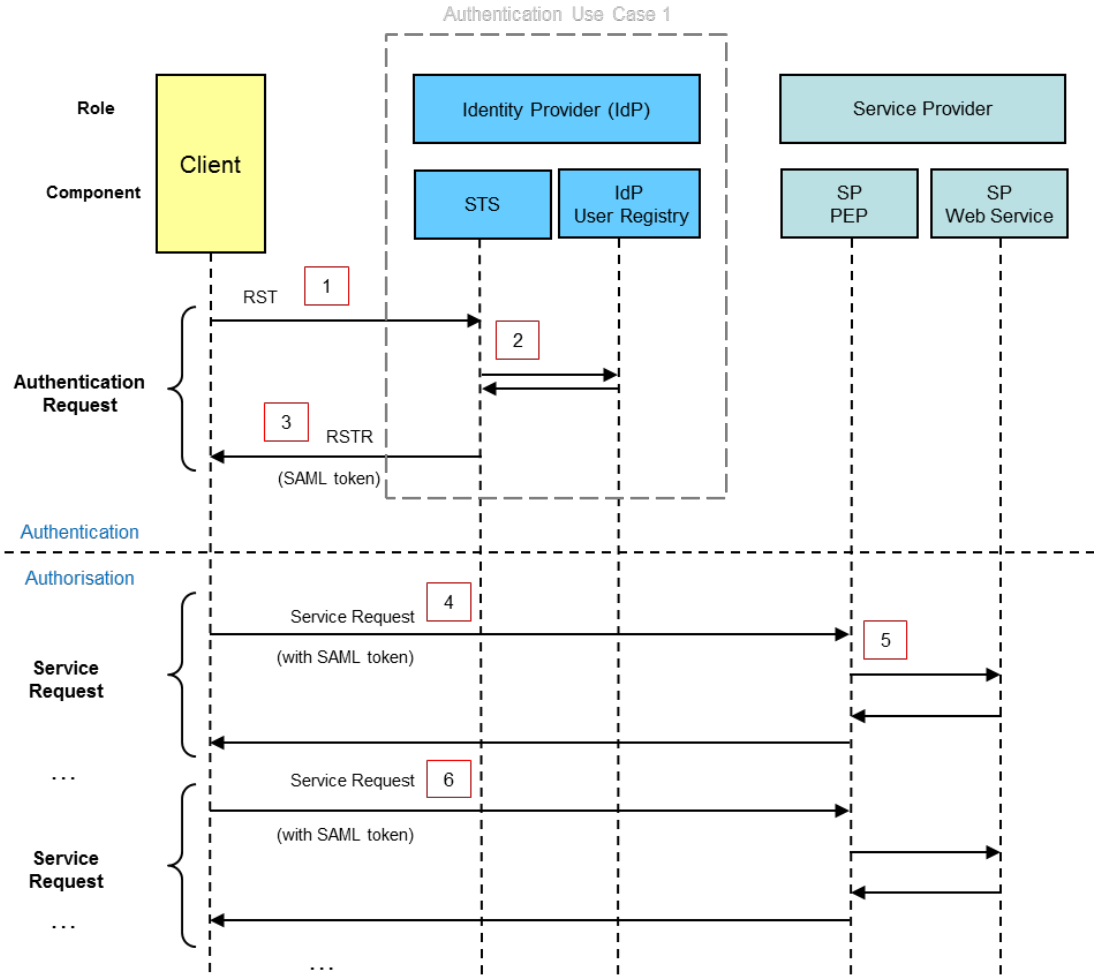


Figure 3: Authentication / Authorisation Use Case

A high level use case for authentication and authorisation is shown in the above figure. Note that the diagram has a higher level of abstraction than the other diagrams present in the remaining of the document; more precisely, the IdP depicted in the figure may either authenticate users on its own or delegate the authentication to another IdP. The same applies for the depicted SP. Following sections of this document further elaborate the detail of the authentication and authorisation.

1. The RST is sent by the client to the STS, which is directly exposed as a Web service.
2. The IdP performs user authentication checks on the RST and, if successful, retrieves user attributes; then it sends back an RST Response (RSTR) with a SAML token containing assertions on these attributes.
3. The client receives the RSTR containing the SAML token.
4. The client then sends a service request containing the SAML token.
5. The request is received by a PEP that takes the decision to authorise or refuse the request, based on the attributes present in the attached SAML token.

6. This client may send other service requests with the same SAML token (i.e. without re-issuing authentication request), provided that the validity of this token has not expired.

It is worth mentioning that the authentication request is not directly coupled with subsequent service requests. The client is just in charge of attaching a valid SAML token on each request addressed to PEP-protected SP. The same SAML token can be reused several times to successfully access services, provided[2]

- that the PEP of the targeted service is the Relying Party for which the SAML token has been encrypted,
- that the SAML token is still valid (e.g. expiry time),
- that the access policy enforced by the PEP authorises the request.

Based on these constraints, the actual sequence of authentication requests and service requests is determined by the client, depending on the token renewal algorithm, on the targeted services and on the expiry period of the SAML token defined by the STS.

Note also that, although a single STS is typically used per SP security domain, several PEPs can be used for access control to services that are part of that security domain. The mechanism described later in section 6.4.1.1 (i.e. AppliesTo element of RST) deals with the encryption aspects of this multi-PEP configuration. The same mechanism could be used by the STS to generate SAML tokens with a different set of attributes depending on the target PEP.

6.4 Security Model

The model is based on OASIS SAML [NR10, NR12], WS-Security SAML token profile [NR11] and, for the issuance of SAML token, on OASIS WS-Trust 1.3 [NR23] and OASIS Web Services Security UsernameToken Profile 1.1 [NR24].

The model uses a "*Bearer*" scenario; this means in essence that any client that bears a given valid SAML token can make use of the claims contained in that token.

For the present need of SAML token delivery, only one operation of WS-Trust 1.3 is required: the *RequestSecurityToken* (RST), limited to the "Issue" action, as it is defined in the Issuance Binding section (section 4) of [NR23]. This operation returns a RequestSecurityTokenResponse (RSTR).

The purpose of RST (with "Issue" action) in the present Best Practice is to provide a SAML token to a requester, provided that it gets proof that it can trust this requester. The actual proof of trust depends on which entity is responsible to authenticate users, i.e. which entity is the IdP. The present interface supports two kinds of IdP organisation, which entails two different RST formats:

1. *the IdP is the STS (or it can be accessed by the STS)*: in this case, the RST contains the name and password identifying the user plus an optional definition of the designated IdP; the STS checks that the user can be authenticated with these credentials or relay the authentication to the designated IdP;

[2] These elements are detailed and explained in the remaining of the document.

2. *the IdP is another system, not accessible by STS*: in this case, the RST shall contain a user id and shall be digitally signed: the STS checks that the signature corresponds to a requester that it trusts. For this purpose, the STS shall maintain a list of public keys of all the requester entities it trusts.

Case 1, based on user id/password pair, is the usual pattern that has been covered in all previous versions of the present document. Case 2 has been introduced in version 0.0.6; it allows subcontracting user identification to an external Web-SSO type system such as OpenAM, Shibboleth, or ESA EO-SSO (see section 8). The context of case 2 is typically a Portal system that assures that a given user has been authenticated and then issues to the STS that trusts this Portal a signed RST with the authenticated user id.

For the ease of description of the differences between the two cases, we shall use in the following the wording *RST with password* for case 1 and *RST with signature* for case 2.

In all cases, the returned message is a *Request Security Token Response* (RSTR), carrying a SAML token (see [NR23]), which contains *assertions*[3] about the authentication and attributes of the identified user.

The STS receives user credentials over an encrypted channel i.e. HTTPS. The signed and encrypted SAML token is returned over HTTPS and subsequently used in service requests. It is an explicit design decision that the client is unable to decrypt the content of the encrypted SAML token (confidentiality of SAML attributes).

6.4.1 Encryption

Encryption of the SAML token is performed by the STS during the processing of RST. Decryption is performed by the PEP during the processing of service request. The encryption protocol is a "hybrid cryptosystem", i.e. it uses together symmetric key encryption and public key encryption. More precisely, it is defined by

- a key encapsulation scheme, which is a public-key cryptosystem, and
- a data encapsulation scheme, which is a symmetric-key cryptosystem.

From an external point of view, the hybrid cryptosystem is itself a public-key system, which public and private keys are the same as in the key encapsulation scheme. This statement is important for the remaining of the present document where public-key cryptosystem is assumed while symmetric-key encryption aspect is left aside.

The data encryption algorithm used is the AES-128 (symmetric key) while the key encryption uses RSA (public key), as defined in [NR15]. The full encryption process is as follows:

1. The STS first creates the symmetric key using the AES-128 encryption algorithm.
2. This symmetric key is then itself encrypted with the public key of the entity that shall consume the SAML token using the RSA encryption algorithm to create a secret key.
3. The SAML token (i.e. the SAML Assertion element) is then encrypted with the generated secret key using the AES-128 encryption algorithm. Note that the encryption type is *Element*, which means that the SAML Assertion element itself is encrypted, not only its child elements; this is specified by the Type attribute of EncryptedData element:

```
<xenc:EncryptedData xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
Type="http://www.w3.org/2001/04/xmlenc#Element">
```

[3] The concept of "assertion" here is a specific instance, in the SAML context, of the concept of "claim" in WS-Trust ([NR23]).

4. The message is then built.

The rationale of step 2 is that the SAML token is encrypted for a specific target Service Provider. Only the PEP of the targeted SP service is able to decrypt the SAML token, through its private key. The criterion used by the STS to choose the "right" public key will be described in the next subsection (6.4.1.1).

An example of encrypted SAML token is given in Figure 11.

6.4.1.1 Retrieval of Encryption Public Key

The STS shall encrypt the token using the Relying Party's public key. This constraint is caused by the public-key encryption of SAML token, which entails that only one defined entity is able to decrypt the delivered SAML token (the one that owns the private key associated to the encrypting public key). In order to afford multiple Relying Parties, the STS shall be able to encrypt the SAML token with one selected public key, chosen among a set of multiple registered public keys.

The target Relying Party is known by the Client of the STS: it is the SP entity to which a service request shall be addressed. This information should be conveyed, from STS Client to STS, on the optional AppliesTo element of the RST, which contains a wsa:Address (see Annex B) as illustrated below.

```
<?xml version="1.0" encoding="UTF-8"?>
<wst:RequestSecurityToken … >
   …
   <wst:AppliesTo>
      <wsa:EndPointReference>
         <wsa:Address>
            urn:ceos:def:epr:esa-cds:1.0:cat-dail-ope
         </wsa:Address>
      </wsa:EndPointReference>
   </wst:AppliesTo>
   …
</wst:RequestSecurityToken>
```

The STS shall use a keystore containing at least one default public key and an unlimited set of public keys associated the wsa:Address of each Relying Party. The rule used by the STS to choose the public key is then based on the AppliesTo element of the received RST:

- *if the AppliesTo element is absent*, then the public key used for encryption shall be the default public key registered on the STS;
- *if the AppliesTo element is present*, then the public key used for encryption shall be the public key of the specific relying party associated to the wsa:Address specified in the AppliesTo element[4]; if the wsa:Address is unknown from STS then the RST fails and a fault shall be reported to the requester (see 7.1.1.3 or 7.1.2.3).

[4] The public key of a Relying Party (PEP) should be distributed together with an identifier, equal to what is used for the wsa:Address value in the AppliesTo element (e.g. urn:ceos:def:epr:esa-cds:1.0:cat-dail-ope). This will facilitate the STS configuration, e.g. using the identifier as the alias name in the keystore to retrieve the public key.

Note that the STS implementation could leave out the treatment of AppliesTo element but, then, it is recommended that STS reports a fault to the requester if the AppliesTo element is present (instead of silently ignoring this element).

The following figure illustrates the various keystores with their keys for the scenario described later in section 6.4.3.3.

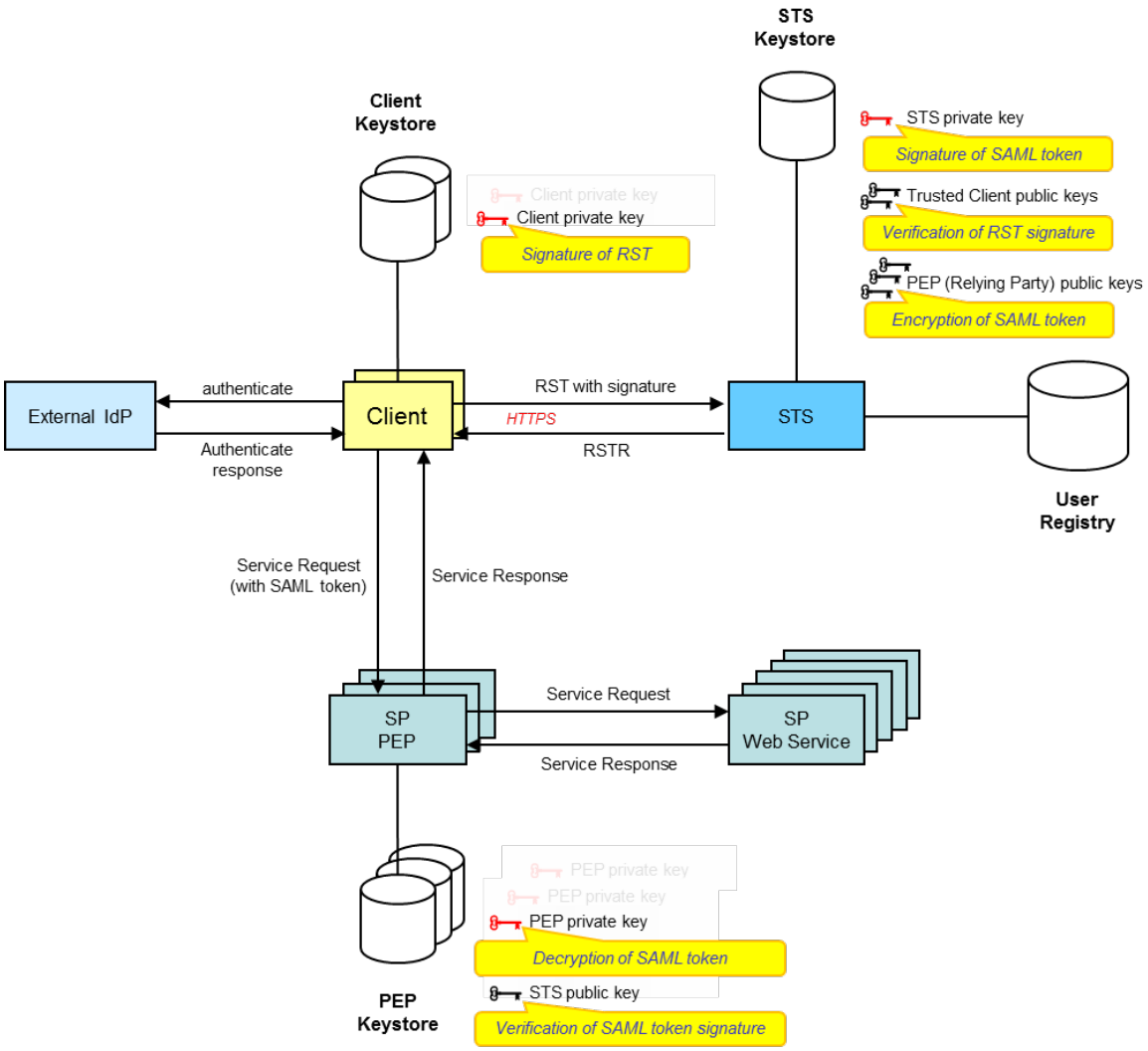


Figure 4: Example of Keys and Keystores

6.4.2 Signature / Message Digest

The SAML token is signed before it is encrypted. The signature process is characterized by the following statements:

- The secure hash SHA-1 [NR13] digital signature message digest algorithm is used, as supported by [NR15].
- The element that is signed is the top-level SAML Assertion, i.e. `<urn:oasis:names:tc:SAML:1.0:assertion:Assertion>` or `<urn:oasis:names:tc:SAML:2.0:assertion:Assertion>`.

- The signature is put as an "enveloped signature" method, which means that the signature element is embedded as a child of the afore-mentioned SAML Assertion element.
- No certificate is put in the signature. This means that the PEP verifying the signature has to know (from its keystore, for example) the public key of the STS that produced the SAML token[5].
- A canonicalization method shall be used which eliminates namespace declarations that are not visibly used within the SAML token. This shall apply for both
    - SignedInfo element, specified in `Signature/SignedInfo/CanonicalizationMethod/@Algorithm`
    - and actual element to be signed, specified in `Signature/SignedInfo/Reference/Transforms/Transform/@Algorithm`

  A suitable algorithm is “Exclusive XML Canonicalization” which is implemented through a digital signature declaration:

  `Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"`

  Note that the specified canonicalization algorithm omits the comments.
- The URI attribute of the <ds:Reference URI="..."> element shall refer to the <saml:Assertion> node being signed (using XPointer, see 4.3.3.3 in [NR18]. The XML pattern is as follows:

```
<saml:Assertion … AssertionID="xxxx" … >
   …
   <ds:Signature …">
      <ds:SignedInfo>
         …
         <ds:Reference URI="#xxxx">
            …
         </ds:Reference>
      </ds:SignedInfo>
      …
  </ds:Signature>
</saml:Assertion>
```

The XPointer format, used in AssertionID and reference URI, shall comply with [NR18]; it is not additionally constrained by the present Best Practice document.

Note that the present Best Practice only enforces the signature of SAML token. Other digital signatures on the remaining elements of the messages, which may be required by interfaces of Service Providers, are permitted but these are out of the scope of the present Best Practice.

An example of signed SAML token is given in Figure 12.

---

[5] To allow for the use of several STS services generating SAML tokens to control the access to the same service (same PEP), it is recommended that the Issuer attribute (SAML 1.1) or element (SAML 2.0) of the SAML token be set to the same STS identifier as used for RST delegation (i.e. wsa:Address in DelegateTo element) in sections 6.4.3.2.

Also, to facilitate the PEP configuration (alias names in keystore), it is recommended to distribute the STS public key together with an identifier, equal to the value used in the SAML token Issuer attribute/element (e.g. urn:ceos:def:epr:eumetsat:1.0:sts-1).

6.4.3 Authentication Use Cases

In the present section, we describe four use cases, which refine the two authentication cases that have been introduced in section 6.1. The new distinction made here covers the fact that the STS receiving a request can either handle the request by itself or delegate it. This choice is actually defined in the request itself (see optional "DelegateTo" element below); it is independent of the two authentication schemes. The final use cases are represented in the table below.

User Registry / *Authentication*	**Local Registry**	**External registry (delegation)**
STS as IdP	① Local STS as IdP	② Delegate STS as IdP
External IdP (Web-SSO)	③ Local STS with external IdP	④ Delegate STS with external IdP

The first two use cases assume that the STS *is* in charge of authentication (case 1.a in section 6.1):

1. **Local STS as IdP**: the STS performs authentication from its local user registry;
2. **Delegate STS as IdP**: the local STS relays authentication request to a delegate STS[6].

The last two use cases assume that the STS *is not* in charge of authentication (case 1.b in section 6.1); the STS delivers security tokens to trusted clients, which rely themselves on an external trusted IdP using another authentication protocol; this case intents to make the present Best Practice interoperable with Web-SSO systems like Shibboleth or OpenAM (see section 8):

3. **Local STS with external IdP**: the STS delivers SAML token to trusted clients from its local user registry;
4. **Delegate STS with external IdP**: the STS relays SAML token request to a trusted delegate STS.

These four use cases are detailed in the following subsections.

[6] Note that the decision tree presented later in section 6.4.3.2 currently only supports one level of delegation.

6.4.3.1 Local STS as IdP (Default Case)

In this use case, the **RST with password** is used; it contains no IdP identifier, so the authentication is performed locally by the STS itself.[7]

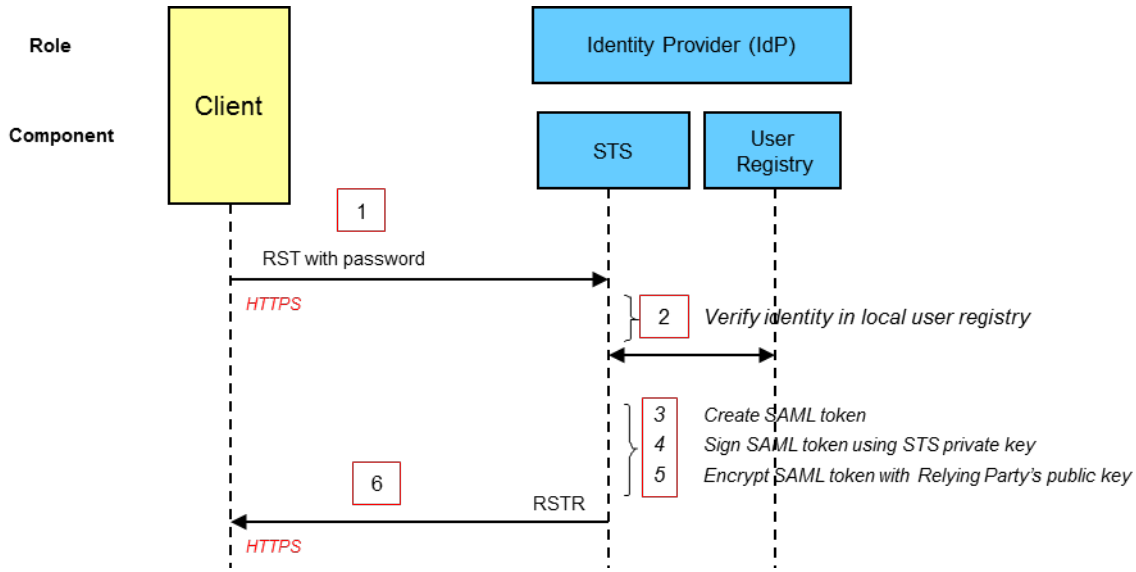


Figure 5: Local STS as IdP (Default Case)

Scenario:

1. The RST with password is sent to the STS using (SOAP over) HTTPS[8].
2. The STS verifies the identity in the local user registry.
3. The STS creates a SAML token using attributes retrieved from the user registry. The SAML token is created containing assertion of the authentication and assertions regarding the attributes of the user.
4. The STS signs the SAML token using its private key.
5. The STS encrypts the SAML token with the Relying Party's public key (see subsection 6.4.1.1 for the process of key retrieval).
6. The RSTR containing the signed and encrypted SAML token is returned to the Client using (SOAP over) HTTPS.

[7] For people having read versions of the present document anterior to 0.3.0, the use case shown here unifies former use cases 1 and 3. The previous approach made a distinction between "Federating IdP" (use case 1) and "External IdP" (use case 3), although the RST protocol was the same. Further analysis has shown that this distinction is not relevant for the scope of the present Best Practice.

[8] The HTTPS encryption mechanism protects against password disclosure in the communication channel. It is assumed, however, that the client takes the necessary steps to protect the password on his side.

The client is unable to decrypt the content of SAML token present in the received RSTR; only the Relying Party can decrypt the SAML token (using its private key).

Example RST with password:

```
<?xml version="1.0" encoding="UTF-8"?>
<wst:RequestSecurityToken xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-
trust/200512/" xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-secext-1.0.xsd">
   <wst:TokenType>
      http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV1.1
   </wst:TokenType>
   <wst:RequestType>
      http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue
   </wst:RequestType>
   <wsse:UsernameToken>
      <wsse:Username>JohnDoe</wsse:Username>
      <wsse:Password>MyPassword</wsse:Password>
   </wsse:UsernameToken>
</wst:RequestSecurityToken>
```

6.4.3.2 Delegate STS as IdP

In the present use case, the **RST with password** is used; it contains a WS-Trust "DelegateTo" element embedding a wsa:Address with an identifier (URI) for a given external STS. The local STS acts here as a delegating STS that relies on another IdP, namely a "delegate" STS, to perform the actual authentication. A mapping table between identifiers and supported external STS URLs shall be maintained by the local STS. No delegation is performed if the identifier is not configured in this table.

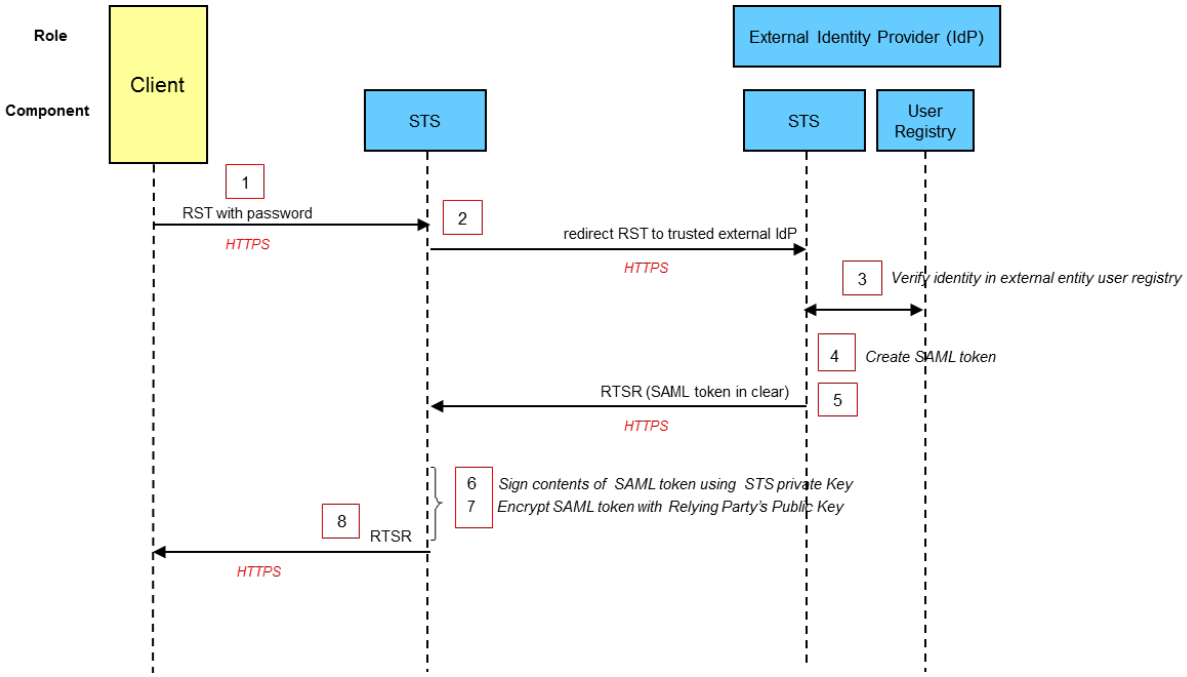


Figure 6: Delegate STS as IdP

Scenario:

1. The RST with password is sent to the local STS using (SOAP over) HTTPS.
2. The STS sees that an identifier of external STS is specified in the RST (in WS-Trust "DelegateTo" element); it redirects the RST to the designated external IdP (called *delegate STS* in the following text). The URL of this IdP is extracted from the mapping table previously described.
3. The delegate STS verifies the user in the external IdP user registry.
4. The delegate STS creates the SAML token using the attributes retrieved from the user registry.
5. The RSTR containing the SAML token, unsigned and in clear, is returned to delegating STS, through (SOAP over) HTTPS.
6. The delegating STS signs the SAML token using its private key.
7. The STS encrypts the SAML token with the Relying Party's public key (see subsection 6.4.1.1 for the process of key retrieval).
8. The RSTR containing the signed and encrypted SAML token is returned to the Client.

Notes:

1. As for the previous use case, the client is unable to decrypt the content of a SAML token present in the received RSTR; only the Relying Party can decrypt the SAML token (using its private key).

2. The confidentiality of the SAML token provided in clear by the external IdP is assured by the HTTPS protocol, which encrypts the response

3. The steps 3, 4 and 5 in the above scenario require that the STS acts as a delegate STS, to ensure that the SAML token is returned in clear and unsigned. For that purpose, any STS, whatever the role it has (delegating / delegate), identified by a given URI e.g. *urn:sss*, shall use the following decision tree for the treatment of an RST:

if the WS-Trust "DelegateTo" element is present in the RST

then if the WS-Trust "DelegateTo" element contains *urn:sss* (i.e. the local STS)

then // *Delegate STS case*
- build SAML token from local user registry
- return the SAML token, unsigned and in clear

else // *Delegating STS case*
- relay the RST to the STS indicated in "DelegateTo" (if *urn:sss* exists in mapping table)
- extract the SAML token from received RSTR
- sign SAML token with STS *urn:sss* private key and encrypt it with the Relying Party's public key
- return the signed and encrypted SAML token

else *// No STS delegation case*
- build SAML token from local user registry
- sign SAML token with STS *urn:sss* private key and encrypt it with the Relying Party's public key
- return the signed and encrypted SAML token

The rationale of this process is to support, with a common unified STS implementation, both Clients that access the delegating STS and Clients that access a delegate STS directly. Also, the system scales up seamlessly in the case of multiple delegating STS.[9]

Example RST with external IdP specified:

```
<?xml version="1.0" encoding="UTF-8"?>
<wst:RequestSecurityToken xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-
trust/200512/" xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-secext-1.0.xsd" xmlns:wsa="http://www.w3.org/2005/08/addressing">
   <wst:TokenType>
      http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV1.1
   </wst:TokenType>
   <wst:RequestType>
      http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue
   </wst:RequestType>
   <wst:DelegateTo>
      <wsa:EndPointReference>
         <wsa:Address>
            urn:ceos:def:epr:eumetsat:1.0:sts-1
         </wsa:Address>
      </wsa:EndPointReference>
   </wst:DelegateTo>
   <wsse:UsernameToken>
      <wsse:Username>JohnDoe</wsse:Username>
      <wsse:Password>MyPassword</wsse:Password>
   </wsse:UsernameToken>
</wst:RequestSecurityToken>
```

6.4.3.3 Local STS with External IdP

We cover here the case where there is an external IdP in an external security domain, which does not comply with the present Best Practice but which is trusted by the local STS. This use case is meant to make the present Best Practice interoperable with Web-SSO systems like Shibboleth or OpenAM. For instance, ESA EO-SSO, an SSO system based on Shibboleth, defines a specific security domain with its own IdP (see section 8).

In this present case, the **RST with signature** is used. (RST contains no password).

In order to integrate such external IdP, a trust relationship shall be established between the two security domains such that a service consumer (client) shall be able to get a SAML token on behalf of any user that has been authenticated by the external IdP.

[9] Note that several variant mechanisms are feasible, if we allow the inclusion of *multiple SAML tokens* in the RSTR and/or service requests. A client could then own several tokens for the same user at a given time, encrypted with different public keys and potentially carrying different contents. The PEP should then be given several "chances" (one per included SAML token) to succeed in decryption and to authorise a request. These variant mechanisms change the interfaces defined in the present version of the specification and, therefore, are no more than a subject of investigation.

In order to establish a trust relationship between the two security domains, a given Client[10] *C* of the external security domain shall provide its public key to the local STS. The trust relationship between *C* and the STS is established as soon as the STS security administrator has registered C's public key in the keystore of the STS. From that point on, client *C* can obtain a SAML token for any user authenticated on the external IdP by issuing an RST with signature.

The afore-mentioned trust relationship between *C* and the STS assumes that the STS trusts the IdP used by *C*. Let us stress that it is the duty of *C* to authenticate user on an external IdP, assumingly reliable and secure, as prerequisite before issuing the RST with signature towards the STS.

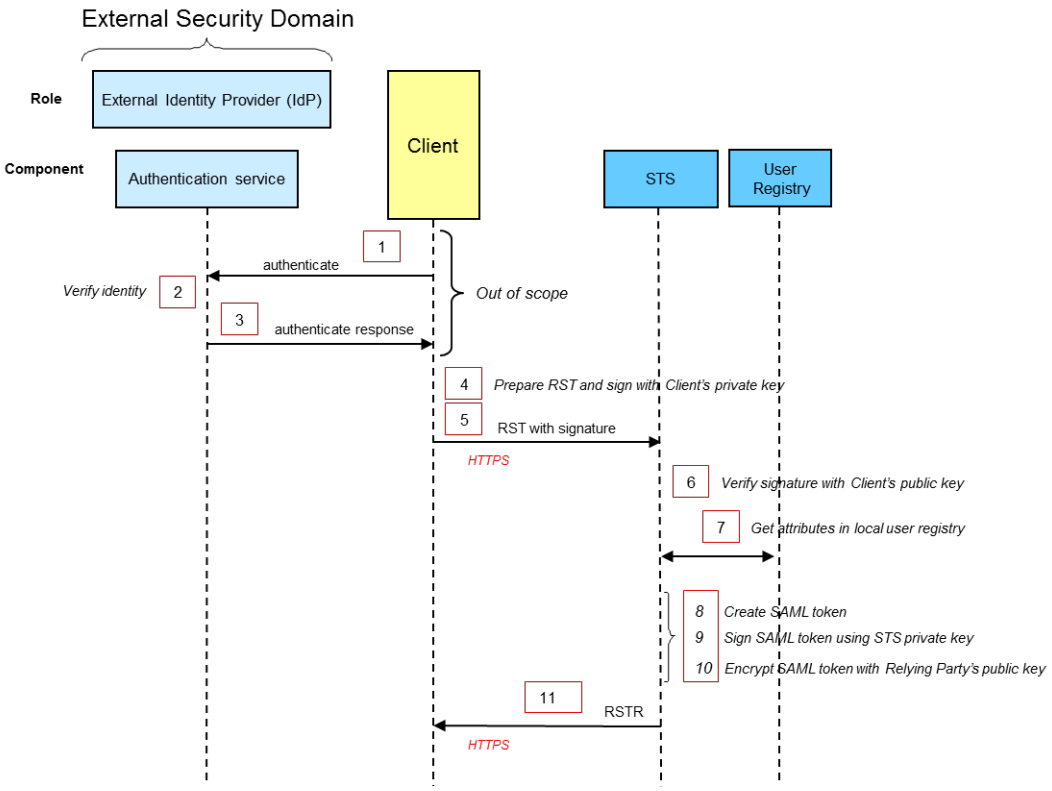


Figure 7: Local STS with External IdP

[10] It is important to remind here that the "client" meant here *is not* a front-end application, like a Web browser; it is the middle-tier service consumer that issues the RST to the STS, like a Portal server (see last paragraph of 6.3).

Scenario:

1. An authentication request is sent to the external IdP (authentication mechanism specific to Web-SSO system used, out of scope of the present Best Practice).
2. The external IdP verifies the user identity (specific to Web-SSO system used).
3. The authentication response is returned to the Client (specific to Web-SSO system used).
4. If the authentication was successful, the Client prepares an RST with the user id[11] (no password) and signs it with its private key.
5. The RST with signature is sent to the local STS using (SOAP over) HTTPS.
6. The STS verifies the signature of the RST, based on the set of registered client's public keys (stored beforehand in the STS keystore, as trusted Clients); this succeeds.
7. The STS retrieves user attributes from the local user registry.
8. The STS creates a SAML token. The SAML token is created containing an assertion of the authentication and assertions regarding the attributes of the user.
9. The STS signs the SAML token using its private key.
10. The STS encrypts the SAML token with the Relying Party's public key (see subsection 6.4.1.1 for the process of key retrieval).
11. The RSTR containing the signed and encrypted SAML token is returned to the Client using (SOAP over) HTTPS.

Example Request Security Token with signature:

In the SOAP (or HTTP) header:

```
<wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-secext-1.0.xsd">
   <ds:Signature xmlns:ds="…">
      …
   <ds:Reference URI="#body"/>
      …
   </ds:Signature>
</wsse:Security>
```

In the SOAP (or HTTP) body:

```
<wst:RequestSecurityToken>
   <wst:TokenType>
      http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV1.1
   </wst:TokenType>
```

[11] The user id is (or is derived from) any Web-SSO attribute(s) that uniquely identifies the authenticated user in the given Web-SSO security domain (e.g. JohnDoe in EO-SSO).

If Clients from several external Web-SSO security domains are allowed/trusted to access services of the same Service Provider security domain, then to remain unique, the user id should include somehow a Web-SSO security domain specific part that distinguishes it from the other Web-SSO security domains (e.g. JohnDoe@EO-SSO).

A future version of this document could consider adding an optional <wsp:URI> element in the <wsse:UsernameToken> to separately identify the Web-SSO security domain (e.g. `<wsp:URI>urn:ceos:def:epr:esa-cds:1.0:eo-sso</wsp:URI>`).

```
   <wst:RequestType>
      http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue
   </wst:RequestType>
   <wsse:UsernameToken>
      <wsse:Username>JohnDoe</wsse:Username>
   </wsse:UsernameToken>
</wst:RequestSecurityToken>
```

6.4.3.4 Delegate STS with External IdP

The present case is in essence similar to the Local STS with External IdP case for the first part and to the Delegate STS as IdP case for the second part.

For this present case, the **RST with signature** is used. It contains a "DelegateTo" element containing a wsa:Address with an identifier (URI) for the delegate STS. A mapping table between identifiers and supported external STS URLs shall be maintained by the local STS. No delegation is performed if the identifier is not configured in this table.

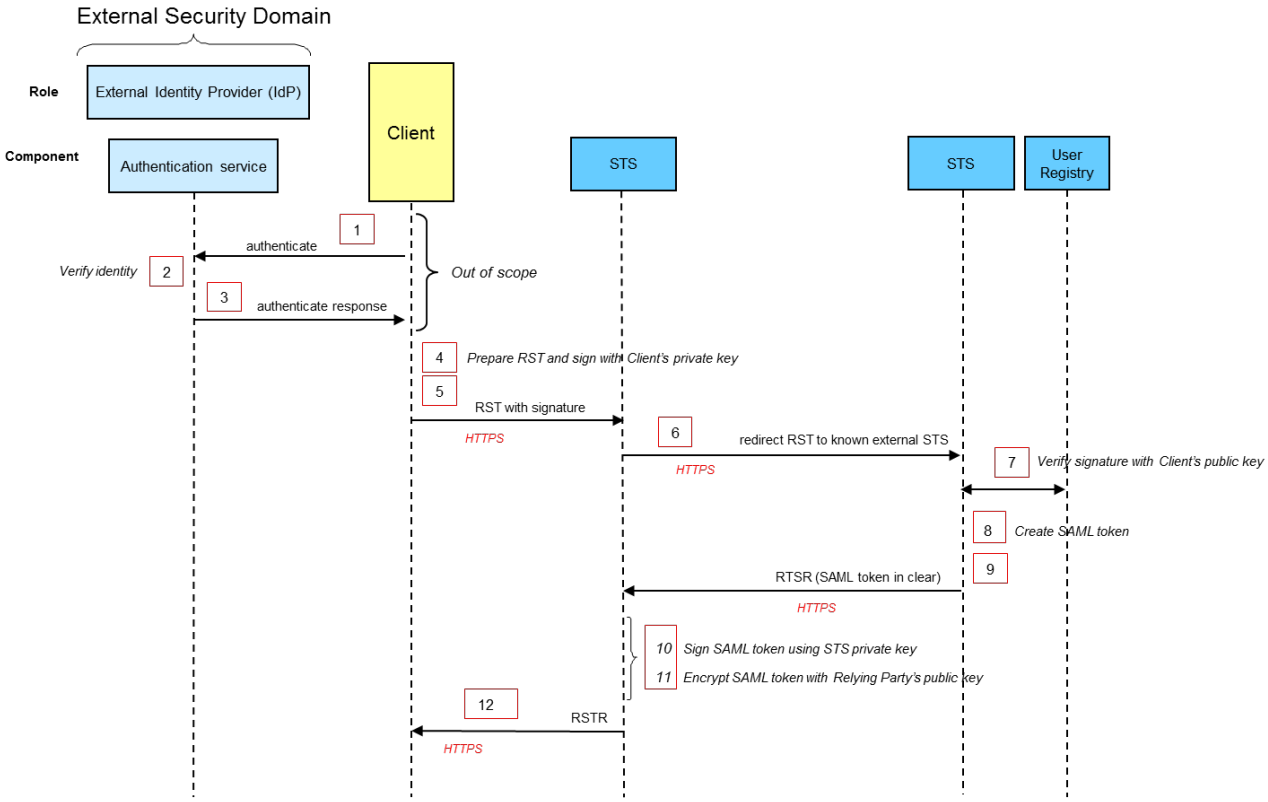


Figure 8: Delegate STS with External IdP

Scenario:

1. An authentication request is sent to the external IdP (authentication mechanism specific to Web-SSO system used, out of scope of the present Best Practice).
2. The external IdP verifies the user identity (specific to Web-SSO system used)).
3. The authentication response is returned to the Client (specific to Web-SSO system used).

4. If the authentication was successful, the Client prepares an RST with the user id (no password) and a WS-Trust “DelegateTo” element; the Client signs it with its private key.
5. The RST with signature is sent to the local delegating STS using (SOAP over) HTTPS.
6. The delegating STS sees that an identifier of external STS is specified in the RST (in the WS-Trust “DelegateTo” element); it redirects the RST to the designated external delegate STS. The URL of this IdP is extracted from the mapping table previously described.
7. The delegate STS verifies the signature of the RTS, based on the set of registered client's public keys (stored beforehand in delegate STS keystore, as trusted Clients); this succeeds.
8. The delegate STS creates the SAML token using the attributes retrieved from the user registry.
9. The RSTR containing the SAML token, unsigned and in clear, is returned to delegating STS, through (SOAP over) HTTPS.
10. The delegating STS signs the SAML token using its private key.
11. The delegating STS encrypts the SAML token with the Relying Party's public key (see subsection 6.4.1.1 for the process of key retrieval).
12. The RSTR containing the signed and encrypted SAML token is returned to the Client using (SOAP over) HTTPS.

The steps 7, 8 and 9 in the above scenario require that the external STS acts as a delegate STS, to ensure that the SAML token is returned in clear and unsigned. For that purpose, the decision tree presented in 6.4.3.2 shall be used.

6.4.4 Service Request

The service request may contain an encrypted SAML token. This SAML token is obtained from an RST as previously described and is used to control access to services (see Figure 3).

Note: It is not mandatory that the service request be preceded by an RST, as the SAML token is not mandatory in the service request. However, this should still be authorised by the PEP.

Since a specific SAML token protocol is required to access the protected Web Services, the use of WS-Policy [NR20], WS-PolicyAttachment [NR32], and WS-SecurityPolicy [NR33] is recommended for the WSDL files describing these Web services. The WS-Policy elements are used to formally specify the presence of SAML token in the request, the encryption algorithm, etc. With such dispositions, the Web services are self-describing, allowing for "discovery" by clients, hence improving the interoperability of the system. An example of WSDL using WS-Policy, WS-PolicyAttachment, and WS-SecurityPolicy is provided in Annex F.

The choice of attributes to be used in the SAML token and the access policies applied by each PEP are out of scope of the present Best Practice. However, to help understanding, several examples of authorisation rules along with their XACML counterparts are provided later in section 10.

7 Interfaces

7.1 STS Interface

The Request Security Token (RST) operation, as defined in WS-Trust 1.3 [NR23], allows clients to retrieve authentication metadata from a nominated IdP server. The response to an Authenticate request should be an XML document containing authentication metadata about the authentication and requestor.

This Best Practice supports the use of both SAML 1.1 [NR10] and SAML 2.0 [NR12] to model SAML tokens. The desired SAML version shall be specified using the standard WS-Trust TokenType element of RST (see schema in Annex B). More precisely, the format of returned token shall be SAML 1.1 or SAML 2.0 depending of the value of TokenType, respectively,

`http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV1.1`

or

`http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV2.0`

which are standard identifiers specified in SAML Token Profile [NR11].

As explained in section 6.2, two binding protocols are supported:

1. the SOAP binding (see subsection 7.1.1),
2. the HTTP binding (see subsection 7.1.2).

7.1.1 SOAP Binding

Protocol: SOAP over HTTPS

7.1.1.1 Request

As explained in section 6.4, we make a distinction between RST with password and RST with signature. The following XML-Schema fragment defines the XML encoding of the message body of the RST *with password*.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:wsse="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-trust/200512/">
   <soapenv:Body>
      <wst:RequestSecurityToken>
         <wst:TokenType>
            http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV1.1
         </wst:TokenType>
         <wst:RequestType>
            http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue
         </wst:RequestType>
         <wsse:UsernameToken>
            <wsse:Username>JohnDoe</wsse:Username>
            <wsse:Password>MyPassword</wsse:Password>
         </wsse:UsernameToken>
      </wst:RequestSecurityToken>
   </soapenv:Body>
</soapenv:Envelope>
```

Figure 9: Example of RST with password

The following XML-Schema fragment defines the XML encoding of the message body of the RST *with signature*.

```
<S11:Envelope xmlns:S11="…" xmlns:wsse="…" xmlns:xenc="…" xmlns:wst="…">
   <S11:Header>
      <wsse:Security>
         <ds:Signature xmlns:ds="…">
            …
            <ds:Reference URI="#soapbody"/>
            …
         </ds:Signature>
      </wsse:Security>
   </S11:Header>
   <S11:Body Id="soapbody">
      <wst:RequestSecurityToken>
         <wst:TokenType>
            http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV1.1
         </wst:TokenType>
         <wst:RequestType>
            http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue
         </wst:RequestType>
         <wsse:UsernameToken>
            <wsse:Username>JohnDoe</wsse:Username>
         </wsse:UsernameToken>
      </wst:RequestSecurityToken>
   </S11:Body>
</S11:Envelope>
```

Figure 10: Example of RST with signature

7.1.1.2 Response

The following XML shows an example of response, which is a Request Security Response (RSTR), as defined in WS-Trust 1.3 [NR23], containing an encrypted SAML token.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
   <soapenv:Body>
      <RequestSecurityTokenResponse xmlns="http://docs.oasis-open.org/ws-sx/ws-
trust/200512/" xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-
1.0.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
         <TokenType>http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-
1.1#SAMLV1.1</TokenType>
         <RequestedSecurityToken>
            <xenc:EncryptedData xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
Type="http://www.w3.org/2001/04/xmlenc#Element">
               <xenc:EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
               <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
                  <xenc:EncryptedKey>
                     <xenc:EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
                     <xenc:CipherData>
                        <xenc:CipherValue>
cbE8viFOmyDuxR8N4EdwS9UUKpSoUbMrWSVprW7IypMwFZLeHR9Rxd4iw5dUl4K+TffyNdRJ9Tr9PD8YIdpFLz
CvYas63g5x4/XnyA1E2AU8ZBBpM2dtbr3g4IYMywfraWrI76mHM+MERVZdHMVBWFrhqXhcS92m23m+amt14mk=
</xenc:CipherValue>
                     </xenc:CipherData>
                  </xenc:EncryptedKey>
               </ds:KeyInfo>
               <xenc:CipherData>
                  <xenc:CipherValue>
VEHlprDMQ+DqIpoPqx6TYi/mMX2dGV5JCJCrhDquZHRKqOiaIFfwqOMZvn2HW2JDFvUxJ6LRTKdNujQI7sxc6h
3IGBL7NXF7bx4jGwQ09wAA7nm6OoB4jiGdaqb8wTx0olnzn2WqOWoVeTng1lwBi0rv2+iD1HWnXAUUHfJH8ALq
4IU3hR0vjoqJH6Y21EuXPeXp/dYPUw3oIFn2FEID2u+8T+xOxbbq2ezQbU3z8n1LbgvDtN3ex5lUCo260pOOPn
92nn7nYErT682eYd+bCKoiENpQSYgHszvvyqFf9o6O0u87zk4AORWsRhQH74L2gG8wVOeHKyhEx0RsBkf4xZcQ
KBvQ9JHWQWpDEB51NZaJe1hSyaUk6T5gf9ArDnz6UwL0ZTDp6Dxgjha91u5qIMG3ECxVYKcnBv+O6Om1Q0HbL0
ecbUDR56evS+mf0U9JxduBKwFJLqta6D0wmwqYWcaF3ZrKd7SatV8Z2l0DmWTMe5R+x6O1RpbKltlduKl4bLaS
```

```
YFpaqaU758ZsmTDmjQQj8fn1qCZbDtp4SEVPWumoTg2k7RAOay2QtV5b+VA9wloSXoxVf2csLSOOH/NDE1noBI
pzgUb9Xm/YIPwikQKsxNPFM72yLrS0vjAho1Cxrg+8l7XIVcmowhPnLqSs6ZpvA01YP8EhsOFlN+0y+9EfAuoY
4jYcScfwqDehth76lER+EyAdFLi10VhVxKW14VLbmksAydndIQaw6VzGm1Qwoc3CeCaeq4q4GgFgiem1BmW9Ie
BaUBTX2wZmIKG8Z9Xhjv6MwT7hOeWH5fefipJs8JS8l6wQBo8WAczzmw6s1j8JW9YDyAWosfoTPrtOwFTaaYSi
aEXvPOnb5RgR/W4ivZ64ioA8FXyLFoWcNEJJ6AgWHDLAbCDg/zvnVwEs7OdaSxRTxVNsc7cpclGspSmk/HzGYx
PHInGhn/QPsac5iN6t6HlwnQUJgt81rI/tbFfSYqqtYqXKeNoEtw91/1DZVUi7mSc7Xj2e2Wb65h8PIoYeX3Nl
i+i4SrOoeAKaZrHtpqP6f+pI4lpkANS4RFxFDiL9Ddxv1WKD//nMck0Su0HfIbPYUYF0GGvlHsv6IiwT8dj/f0
MnCxkAgegliGageZthQiNavOcURRC/94d+1jDZGayowurzdxmJhxyiEY5REQQt3hK4aAD89wMjndzxHdtcQEuv
XA5uSm3T9qgIm4Qdvuh54PW/SKptG9fDj4paTxVVlfZ+0f/1Vxjj4pPIKOVjE3e4ChBPkJXD/nXqZ8DdR+zPxO
LWYyiqnMaxv3OInd/Iz2Lq36a09b0JEfMVz4e39sGtFzNDbxXgQnTx4L3jDYFdl5+gelUNduK9HtgklXDWfNIM
WtY5xhdTX0m3Q8hBtNgKOHeg7BcBXF+uT30mqwgJu5cbJQl/1j/QlMvromUaUQATN2ULu7mMiTWkoYoMTiijJG
AizbKIi6O5xmHvF/jicd7lBcmSz+B+BnrnqxY5DM/qQSFsnRoGmPKlJeiao2g+QuMD5x7H+pBUIq3B8lkMlUBg
5VoKx2+kCHuP2lamGFskQl80PRGygQ5adA2iWwKoIKoCdcYIc9C2sPVkJz+slExJXizl4L51GEWDlQ8VGsqNV7
CzOyIt0uXIIBQW3j0aX//7QoYVfM681TiqvtaDEY7Ip4nSV839e5xnj3s+qgzXOpoK5rw5EThDLhthPy97CJiu
SbsfcGf1DvWNE74x2E4b8HazacItRBIbx0GFDHIoqaHEih6zlhQaqwloLnUHRpL8vAQLVKiW3q94569e3GenoQ
bpjxKQ9F58VuQh3ZiZtJ+17XOxDx6ZDXcTiQDa+3nXiTgT7k9gGtpIv8vYLMuUHDEZx1zGd/rMzU3JAbbfKO8+
Cs3pMnb2KpGL4rLLgivee1P35rbHK04V8D0NbwDk0TOVnmFQWIRsgVtPwmEHXbFl3jqIUTx4xdishXmkKcDCwa
NfqYo4yzmgLUlbtchkDGF9YBA0mXv4gT7z0TiBb1jUFBhTnciL4DBkI9K8wEjklHUi7w4LjvhCB5B15y1ZG/ba
IscoRrVu41HUp7crwbsdKjcWGE//dBTXN1vrXDm0maAkConuYNPpMY0Cf+ikVITJO73UaXplFjOOtm+mkql6e5
nTd8qGQXwHZl/nGJOEO/rMsXSoVybxIn5bg+97CFCtAdsRRjAJZRQcrJIztenGJX81U0rvAX+OuoSNrgVpdxcw
uH/1x80i+CY5kdUkg3EMkU0m4FiNQ6CyXiiimVSBRB0sHfWW5/Em+qlYeRjrXCyJBYPo2mCuMtqQN42VeShEkQ
1XPx6o09NTaaxXRMpV2IHjzALLrR0Px6zqbp7CuEhPdLxTYcXetDKQJt/XHJuWdvETMgsJnyQ0cCJSPXp21xsr
K6zYLYcQ1M8rS9RHCmWvFdTZg49mX3QPANOUDPoPR0y3mOT19FWKFYOfQhHN2xPJZAPV6ZJeReeAeTBRkTvgIJ
E/3BsQpmqsSusjgEDYCrK8MfaybAC6CpE5ZKnQwV99YlTcbPx7vVKPuU13j3Aj4FjtGjkFunfCOpLX1AA0FSbB
fOOCVeYd94bCGaW8f+j3NBB+29ELYMskew2tyCBiw2HodBrMoDiYVWHbD+bWw8qMOOBurEQihVdNq5Tbi3R2fn
nX9DpfbVljJeKFjyVwCLZA5OdGIYPuJxrXGKsaBI+abTgciL4n4WbsG7LalURKCMe/HH1jVuy8VjevwJMB+u7C
HoOc9jVCwR4YSPjH9fbxcIn9UIuECm1CryEUYB6kkhBEeyxdQclP0amPYlyxVU80KN+Mxvle4//B6kwUtjS+/R
v943oXrXxaLXTCpeds49x0FWSRo/HCxynunzpkyqD4wBfUyB7hYggeRUaCb7aNVuIB1QZSY9EqF3F26Aootz1c
Ypr1CBtizZK9Q6Ez6N3iYW1dMUB7dsNp4a4emAU3CfhHYh3JNv4pD21PbPASO/t89v7uMDrsI8SOp1nHqV0hYG
2+JhxNyhYKV1oXv54mKzbW+4vwsU/ySrrexUvmkTzLCsYBI7nSZT5uVprRA+MQJBLx6dKVVuzO1x8hzTv9T2Lv
Jr7rpd6Ban94JJ8vG7OUO0OaNP9HDrz+34xmCqQRi/f0TkmfSo4uFcsIfAmdQVbd6uu22ZBoWqolazlBXjt5Oe
2AQV51Zma53dlArSBLpvbg/RoMM7cMhnGn33DkSBDYU9rN2iApw0zswa/KJ/p1r33Jrk5YTL6wTTEuaG+UxVrt
CxX4VHk7syaOjI5dshRELos3ZeIJeqKAgS45H6cK+gjCq0l3qWDDnFHcGmzYoP1665lC7c9TOs8i5OBLM6hGqg
EgcKEiTIp+tRUvDEhyN1v/YngT3izvWbsijV0QTTJcjsyFWqDSJiw8G1WH4oFqZAzF2UzE6fzEeQbMVlPPxlnp
UjipTqdtWcuayLH7tifX7diBl1fjlUOTqPK2+5vz1HckVtzJMS4g0W7rWHAbTv5nfrby/1IJBHMDutjI2dh6J7
nXbSgFOiT98TFL7upJCNc7T3AH4jRo1TzzXqODFShamQeYlooCyvStQxqj08rz74+7ery+GapNEPL4cPZ1qV0b
fKCBwOQrTV8lIZXsFtJj9TV+7lT6ZcePnCFY6pWI78u5WWePZunmI9FFhz+odZd4vfh0C3VEISmeEN28T8Xdvt
Ht8A78sr4/SmrPteZpZhByZe2n50ZHQU+uknсDgZirtz5A4LIBedcDLCgeNfonHYCQTNYOKoDA+eq5sBczKPmq
FKjPnBq1533/lptWhsgou8CZfsEaY4kZvEzK8YTVrfVt4T407A851vKxBfHIyXKxFFi17Yddr2SiqebAUjT3wa
PAoUwgdJelDYTnnKUQy0Zm25gGRDiE9LUwoOp7ys0H9m/xXJROx76gbljguU3ad9fcwQIm8RTKZvXvKrVRBUsH
utEL6/qZAb5VBQ1JHsa4tknAFTdwh7lsB1/l0lHtZ+HzBdgZ8kOvRmHiCKYb+2p26WMVNy8SRhW8EeYxx3t79L
MU3pIp9w4rCnuClwAYAXN6PP1Gf5GgsGS228ur3vwNKO8YZIdMatmKJDy8Ufkm1Ljvy4Z0/3+XcGLDWyxRx6M2
mLvMPvJIz9iGSr684PRfSydR3nq6W7gwYcOhb62cmSLVWyECoaa+cqVFFGOKHcUT3ZS7Xlx0QkniCQI9d46XDE
x64PFGeBXL/z4dj7ZYx6woX9R+F5yOAdKoILV5N9m4xzauPO4EkmKakDBtsf9tzExrArDBoT664Xc7cVJ/2jTz
X57Oms09Q7r+T8hH0JNxhcXAqhxdbMitkcFSy7t0pBgrPXRhdXohbGlhuZPAOMVkWWDMf8x7Yc4k7F3l9ua67w
5Z2QcDf8NBq5iYM3TkB+2qpmn16L7Pbp5qlAoIcB409+6VwxHiHQgBHOPGsP1xHNYGYyKcfR4VxaUUXf5G18b5
NOnx3S2VCBA9fJGXlHqW3RmtlMEP4dEQdCbhH7jw7jd5El0NabRA0fCBTAYR61vYa9Ov7SDOIefy6NpDffg9sF
ltOa36ag==
                    </xenc:CipherValue>
                </xenc:CipherData>
            </xenc:EncryptedData>
        </RequestedSecurityToken>
        <wsp:AppliesTo/>
    </RequestSecurityTokenResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Figure 11: Example of RST Response

An example is given here for completeness of the fragment before encryption:

```
<saml:Assertion xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
xmlns="urn:oasis:names:tc:SAML:1.0:assertion"
AssertionID="oracle.security.xmlsec.saml.Assertion1955a65" IssueInstant="2009-06-
25T13:34:55Z" Issuer="urn:ceos:def:epr:esa-cds:1.0:sts-ope" MajorVersion="1"
MinorVersion="1">
   <saml:Conditions NotBefore="2009-06-25T13:33:55Z" NotOnOrAfter="2009-06-
25T13:39:55Z"/>
   <saml:AuthenticationStatement AuthenticationInstant="2009-06-25T13:34:55Z"
AuthenticationMethod="urn:oasis:names:tc:SAML:1.0:am:password">
      <saml:Subject>
         <saml:NameIdentifier>JohnDoe</saml:NameIdentifier>
         <saml:SubjectConfirmation>
```

```
            <saml:ConfirmationMethod>urn:oasis:names:tc:SAML:1.0:cm:bearer
</saml:ConfirmationMethod>
         </saml:SubjectConfirmation>
      </saml:Subject>
   </saml:AuthenticationStatement>
   <saml:AttributeStatement>
      <saml:Subject>
         <saml:NameIdentifier>JohnDoe</saml:NameIdentifier>
         <saml:SubjectConfirmation>
            <saml:ConfirmationMethod>urn:oasis:names:tc:SAML:1.0:cm:bearer
</saml:ConfirmationMethod>
         </saml:SubjectConfirmation>
      </saml:Subject>
      <saml:Attribute AttributeName="Id">
         <saml:AttributeValue>JohnDoe</saml:AttributeValue>
      </saml:Attribute>
      <saml:Attribute AttributeName="c">
         <saml:AttributeValue>Italy</saml:AttributeValue>
      </saml:Attribute>
      <saml:Attribute AttributeName="o">
         <saml:AttributeValue>ESA</saml:AttributeValue>
      </saml:Attribute>
      <saml:Attribute AttributeName="ProjectName">
         <saml:AttributeValue>GSCDA</saml:AttributeValue>
      </saml:Attribute>
      <saml:Attribute AttributeName="Account">
         <saml:AttributeValue>dev</saml:AttributeValue>
      </saml:Attribute>
      <saml:Attribute AttributeName="ServiceName">
         <saml:AttributeValue>Geoland2</saml:AttributeValue>
      </saml:Attribute>
   </saml:AttributeStatement>
   <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="#oracle.security.xmlsec.saml.Assertion1955a65">
            <ds:Transforms>
               <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-
signature"/>
               <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
            </ds:Transforms>
            <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
            <ds:DigestValue>nLkuqyqDggsxQnPiGzVDDckxaA0=</ds:DigestValue>
         </ds:Reference>
      </ds:SignedInfo>
      <ds:SignatureValue>
oOkdc3KB2HwPB6YzhEa9MHx5yo1u/xqHp81wPj68uf5Ypet/5wHHEvfuNhxD+S3ejT2f4lKIGkVDcsRNyUqaAn
60CnJiN4RBpwcjcWQSUj5/XxesaR4nO4CtDylaLV6acLwwwlLN5PQ66UumASE=
      </ds:SignatureValue>
   </ds:Signature>
</saml:Assertion>
```

Figure 12: Example of Signed SAML Token

The example above uses user attributes listed in Annex D.

7.1.1.3 Exception

If the RST cannot provide the RSTR due to a failure (failed authentication, invalid signature, invalid parameters, SAML version not supported, AppliesTo not supported, resource unavailable, etc), then the SOAP Fault mechanism shall be used, following the recommendation of WS-Trust 1.3 for error handling (see section 11 of [NR23]).

The SOAP fault fields of interest defined by WS-Trust 1.3 are repeated in the following table.

Fault code (faultcode)	Error that occurred (faultstring)
wst:InvalidRequest	The request was invalid or malformed
wst:FailedAuthentication	Authentication failed (e.g. invalid password or signature)
wst:RequestFailed	The specified request failed (e.g. SAML version or AppliesTo not supported)
wst:BadRequest	The specified RequestSecurityToken is not understood.

An example is given below, for the case of a failed authentication:

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope"
xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-trust/200512/">
   <soapenv:Body>
      <soapenv:Fault>
         <soapenv:Code>
            <soapenv:Value>env:Sender</soapenv:value>
            <soapenv:Subcode>
               <soapenv:value>wst:FailedAuthentication</soapenv:value>
            </soapenv:Subcode>
         </soapenv:Code>
         <soapenv:Reason>
            <soapenv:Text xml:lang="en">Authentication failed: invalid
password</soapenv:Text>
         </soapenv:Reason>
      </soapenv:Fault>
   </soapenv:Body>
</soapenv:Envelope>
```

7.1.1.4 WSDL

The WSDL is given below for the Security Token Service, without the Bindings and Services elements. This WSDL has been obtained by updating reference files from WS-Trust 1.3: http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3.wsdl .

Note that this WSDL refers to the local schema file ws-trust.xsd, which is a restricted version of the standard WS-Trust schema http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3.xsd .

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:tns="http://docs.oasis-open.org/ws-sx/ws-trust/200512/"
xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-trust/200512/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="http://docs.oasis-
open.org/ws-sx/ws-trust/200512/">
   <wsdl:types>
      <xs:schema>
         <xs:import namespace="http://docs.oasis-open.org/ws-sx/ws-trust/200512/"
schemaLocation="ws-trust.xsd"/>
      </xs:schema>
   </wsdl:types>
   <wsdl:message name="RequestSecurityTokenMsg">
      <wsdl:part name="request" element="wst:RequestSecurityToken"/>
   </wsdl:message>
   <wsdl:message name="RequestSecurityTokenResponseMsg">
      <wsdl:part name="response" element="wst:RequestSecurityTokenResponse"/>
   </wsdl:message>
   <wsdl:portType name="SecurityTokenService">
      <wsdl:operation name="RequestSecurityToken">
         <wsdl:input message="tns:RequestSecurityTokenMsg"/>
         <wsdl:output message="tns:RequestSecurityTokenResponseMsg"/>
```

```
      </wsdl:operation>
   </wsdl:portType>
</wsdl:definitions>
```

Figure 13: Security Token Service WSDL

7.1.2 HTTP Binding

Protocol: HTTPS

7.1.2.1 Request

The WS-Trust RequestSecurityToken (RST) shall be formatted as described in 7.1.1.1.

The RST shall be put in the message body of a HTTPS POST request, using the "application/xml" media type (Content-Type value in HTTP header).

As explained in 6.4, there is a distinction between RST with password and RST with signature.

For *RST with password*, the following snippet of an RST on HTTPS POST request is provided as an example.

```
POST https://aa.bbb.ccc/sts HTTP/1.1
Host: https://aaa.bbb.ccc
Content-Type: application/xml; charset=utf-8
Content-Length: 390

<wst:RequestSecurityToken xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-
trust/200512/" xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-secext-1.0.xsd">
   <wst:TokenType>
      http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV1.1
   </wst:TokenType>
   <wst:RequestType>
      http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue
   </wst:RequestType>
   <wsse:UsernameToken>
      <wsse:Username>JohnDoe</wsse:Username>
      <wsse:Password>MyPassword</wsse:Password>
   </wsse:UsernameToken>
</wst:RequestSecurityToken>
```

For *RST with signature*, the XML signature applying on the RST shall be put in the HTTP header of the request, in a field named "Authorization". The syntax shall obey the following rules, expressed in Augmented Backus-Naur Form (ABNF) (see [NR30]):

```
authorization = "Authorization" ":" 1*SP credentials

credentials = "Bearer" 1*SP b64token
```

The `b64token` element shall be the ds:Signature element structure, calculated on the wst:RequestSecurityToken, converted by a Base64 encoding (see [NR28]). This detached signature does not contain a URI attribute in the ds:Reference element.

This is shown in the following example.

```
POST https://aa.bbb.ccc/sts HTTP/1.1
Host: https://aaa.bbb.ccc
Content-Type: application/xml; charset=utf-8
Content-Length: 390
Authorization: Bearer oOkdc3KB2HwPB6YzhEa9MHx5yo1u/xqHp81wPj68uf5Ypet/5wHHEvfuNhxD+S3e
jT2f4lKIGkVDcsRNyUqaAn60CnJiN4RBpwcjcWQSUj5/XxesaR4nO4CtDylaLV6acLwwwlLN5PQ66UumASE=
```

```
<wst:RequestSecurityToken xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-
trust/200512/" xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-secext-1.0.xsd">
   <wst:TokenType>
      http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV1.1
   </wst:TokenType>
   <wst:RequestType>
      http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue
   </wst:RequestType>
   <wsse:UsernameToken>
      <wsse:Username>JohnDoe</wsse:Username>
   </wsse:UsernameToken>
</wst:RequestSecurityToken>
```

7.1.2.2 Response

If the RST succeeds, then the response is a WS-Trust RequestSecurityTokenResponse (RSTR) formatted as described in 7.1.1.2.

The RSTR shall be put in the message body of a HTTPS response, using the "application/xml" media type (Content-Type value in HTTP header).

```
HTTP/1.1 200 OK
Content-Type: application/xml; charset=utf-8
Content-Length: 3822

<wst:RequestSecurityTokenResponse xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-
trust/200512/" …>
   …
</wst:RequestSecurityTokenResponse>
```

7.1.2.3 Exception

If the RST on HTTP cannot provide the RSTR due to a failure (failed authentication, invalid signature, invalid parameters, SAML version not supported, AppliesTo not supported, resource unavailable, etc), then the STS server shall respond using the HTTP status code value set to "401 Unauthorized". An exception report message shall be returned as specified in section 8 of the OGC Web Services Common Standard document [NR16].

The authentication specific exception codes are the same as the fault codes defined above in section 7.1.1.3.

An example is given below, for the case of a failed authentication:

```
HTTP/1.1 401 Unauthorized
Content-Type: application/xml
…

<?xml version="1.0" encoding="UTF-8"?>
<ows:ExceptionReport xmlns:ows="http://www.opengis.net/ows/2.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/ows/2.0
http://schemas.opengis.net/ows/2.0/owsExceptionReport.xsd" version="1.0.0"
xml:lang="en">
   <ows:Exception exceptionCode="wst:FailedAuthentication" locator="o">
      <ows:ExceptionText>Authentication failed: invalid password</ows:ExceptionText>
   </ows:Exception>
</ows:ExceptionReport>
```

7.2 Service Interface

Service operations can be invoked by sending a service request to a PEP that shall authorise the access to a given service and, if authorised, shall relay this request to the end-point service (e.g. catalogue, feasibility analysis, ordering services). The request is made using WS-Security containing the SAML token previously returned in the RSTR.

As explained in section 6.2, two binding protocols are supported:

1. the SOAP binding (see subsection 7.2.1),
2. the HTTP binding (see subsection 7.2.2).

7.2.1 SOAP Binding

Protocol: SOAP plus WS-Security over HTTP/HTTPS

7.2.1.1 Request

The SAML token (i.e. a xenc:EncryptedData element extracted from the RSTR) shall be put in the SOAP header of the request, within the WS-Security element.

The following XML fragment defines the XML encoding of an example GetRecords request sent to a catalogue service.

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
   <env:Header>
      <Security xmlns="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-secext-1.0.xsd">
         <xenc:EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Element"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
            <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-
cbc"/>
            <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
               <xenc:EncryptedKey>
                  <xenc:EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
                  <xenc:CipherData>
                     <xenc:CipherValue>
k4kkm+nBkutOsmP9Lm6v4gpPvtJqxOOJLEOoKCQfE4Q7qp1yOBkKRluj9zb7Y07cNdJf8OhzGaHFGz7OIfM9Tp
l5QEntkoeOT9wg/PsYqlIaAsCRoDsYjJoQrqpHdIpv3wlCck8iysQus4LpqdKpWRk0Gk9022z/3U=
                     </xenc:CipherValue>
                  </xenc:CipherData>
               </xenc:EncryptedKey>
            </ds:KeyInfo>
            <xenc:CipherData>
               <xenc:CipherValue>
+X0OFMae+FV8zOr0pPA02icglYf4AKcaml/jNfP8gdmjIh/dB/utVICYxKtarBRSAlptozGoI92r+bwUwmAyIY
3D7gX0h6EC0P3LqhojKiMRrNbvCaotOPWoMherMp1SUbxeYgxdZVlpXa77mNHHEkjhcmNXHydgz4DJoLxzHUId
Wm9Lv+UTufH+D680JicO0SCnGdIC6KEpM68h+3x/PRRNdmzyWZ03DAADBokRmW8IbG5YlUG4NjZDhkPDRlFhaH
BTn4ZDP4LEd98KXZclwAlAB9XIICTeNFh9t0itufclexX0zu/icAM8ws/sEAh7NmLyw+k8MRI7lMXeldnqftBY
Yj5NzYZqUd87XXqTe6ytnS3SwbZXgdkgKylqdp0p0FcJVOGb4obfP/6irwzf1ujK2DMJb+9+mTQzfdNIIXimeg
V5wY2r1Wsg7XtxiVU6TFMI6VA5CH1MkmgYyYFqgI2MoiNXW3c3sgAs6+QlRoPMR3uNzvtB7NKy0m9BET+zqxCR
gPtGPstjX5ATvJ7tbcAlSKGyHubEIE9Am1Q2nGv3ChGzPPw+rwtow1D8xeSnxWOKpp/whmXcN9AEQ/z5HtDCmb
w1+ExRTM8Xy1NWp135If/6ooxcJtOf5vavo5Mx1OQto8TieF35+5FXA0rUiCn//yJzJRz2mXEMJfoO1HfwPpfG
WxwId4yuhWeylNAA4sKWt2OVDc/zkZpRTIHOoOWuut2LdsZS1fBZ+RMnpt/u8tivsRITLyd2htTILLXKIenNpd
RWeUD4d23RxcFFt+bGh+DIZJD0PfS0zxigXu+NG4wy+Plhj11h4pn2AosIP5v0ZXN/tObgsQonwKyjFwgqGH8j
sIk/96PLnu1ODRRYVIBOGlcV9K7NrHeqCkM1157HCwu/rflTXK61jzRsZ8/hzC8ADiOXQnpk8K4EOAFs/A6LY5
4A8MFQndHkAHN6NEK0nbAkqhOTur99PHrXQtYfsf26Mrd8rTKhkP5zd3pdfzvhnqOnxOSe2FuWX3WHwUTgAzMB
lJC2PRzHM4Q4q/pHFyK+UrLE2QLYsBn6VzouHfcI3dikR/0d2RQPsrQKQPB8WXMjJxK7v05jRBjZaNYpmsFk08
zweM1OWuVB8L57zSzAfb7CKpPgsgRk4ezLWrPVK7Z2UuQ+z/UH66S3a9dYsneQMaDMh7wYQtLe3fEeUhbBYrjR
BZWHriOnhxoN7Rll8bnKOXoJUJJMZqyZyvN7qHTLUxG98KUncu3HYwKSIgZEogjee0tx4MhW+t8z9Upsh7TPhW
cEvaFxpn5pfz4c704nsM7Tmcq4IijlnW8m3kX/mBR/OTFcjWh2mV8XZoa/Hro7Rj69HVjELBTlF/W+S7pNsN+h
oErRDlWxuHqC6v7KDMakaLF/Cekzozxm7z1+6a/BeiXSTNojoodOybmv7hxjObWg53RUp5H3rejnN52+7IDHJw
ik2DONIErqL5GPSoy3+adE5mSnAQklZ4zeErsl/A82ySgovaQNskuqa+6aBLvhHQ9CPeQmlddOfyU6HebMSN2m
E8OZ2yejwhujnT5ha4KoefrE21bwi5xWkNDobJbfBPXgg7Wdf4M6n3zsRTT6ixugXrkdRhnYyTzpRJ6EPpL5Cd
```

```
unh2gyQHiyJbcCh37rzTcsx0CiWHfak0ObqDRUeJ8tPyOS5PhyxcNknQ4p3RCI0QJTxUYG3jpWntAK7ZU3d0Bm
k+DAaPGGGJ44fKZ3HZTWjnFTcqWxwYOXxsiJ/kKE8ZVcDJ6IL4pf4dTnJHaa8hT09LVutVZJrqcYbI5q/QL1hc
MclPBByoaP4EmFPxX3dpbap0uf2qbX80G+jjVtsHd9rhEmyoc6tJjj27Z2B0ANPAI53AMDXGF5HHHzzficfizN
0vK48EO/xk34EEylSmCIInrf72m54f7wh8RojgoOzIWZkIU+tpCfO2HTcxRUT/rd6Wfb624YE9ov20+T1U0Yc9
nyj+tr4FkUoZGyzqm5OlYkfkKvkwkp6rH8RzUhqKewHjWb7zdlYHEH8XkMtcHneYgeQ/E8bshTuLILOELUtROD
jszaTXyrm4xlhCzJ7mWlZa+viTV4PzHdRQCVByGxOsFfhJGtrOOrxtzuTtZ9demhjAE17svhCxh5tIWg5NMJ3F
eG4HzOL9UUoah4gwjyvFa/0azdt1ZWaYc6SPufQIWk9I2HWRb1m4waiA24LkLBXvYMjWTto+DsVWPP3WQXtPaS
BNtnD/VEbM+2bbPERA1+drMTui6Gvl/iHNQ9uq+UrmXPoR9NtfSEAh/M5BdSH75zGifd369R1eFJqBuBWile2R
7ryqnPPBbVf89md0nhYe3RzdD0DKbJen5/r3eicrc3PculW8cGJDqtuEI9kC1xULyXuhWmpEACgTabmNmw3T+3
LnzEuKU97DtLpgqlJoAqXHBBImDsPmzXJ8ouh53L8ZM6jZwFXGqQBteV0HsDPwxTBSGE+tuPQHj/cWvovOBelt
ewUuBskG0EwsDtkYwmP5yNlbY4vn0ouL+4tlprZr2oNSitv3Lle/usE5ps70ALpQYvzG369DAqf2T+m3Ld5HaK
Z/N+cWtQSt/EJGjjBTTodrzbkAe7Mkz/euMxU0Pj5Gv4kyLysfivPPuvar+ZuRos/jm5N+mHUQUzWd2izk4BvB
uyWHNe3Jq45H/ELAND0OEZRoXCRbpz0io+a9C6T44BlOECJyXI1gp/m9lsbb5iSv0HpMSv4xsLpM1AuXLpKmeq
dsHO/zEnLUOhR2Dpk6hoqpnPvK+QbVO0c/YdJ+lkeGIz6C2OsrbbsiO+cUoul+7mJ6WG7VqifJWNX8mzd6RklC
ntt0W1CgBk93vzOspDJfnvBkHSZ1VmuiWLWpesttUrYCwF77lcSLDR3Rvqa29hUORrV4BRHl1LAuf9ofAyg9r3
vb2TjlrG58FOekzRxomjP/rTL0jteYiBf6YwoMEwg2chC3lhRoaTpzTpMSbAoByuI3VCsqMN81JwEAUOtXmjRF
g/C8Cnc1/Zg8tYfNtIymNbzH9S5MmoQy2oEUaM/RmAU1yhqEE+J1q4wBfJb933T8oPOQBgbNjntcDsJXwQt1xa
/QxDs7d1TaGMkzbY6YxZQxt0s6cTwyda2XwqO6Bd9pugluF4c4l218g+42PTzwHWpTcXbqOaQDLVITfM5LWK7J
vAEK5fai9Xc2dooFiNiR+QgU3SwRrZ4GZ1d4wWgOJsPexgEkYOKQuwo6S1vl0WNZtuYC7/+ct4qiHzA2tk+5HB
vMoZ1WyxUQNXo/sfhRx5xK7lT2vHqBx0k1gwoZW7l8/IPPbo0frpdKT2iOGLB/YH46GkWztp0Ft8+7uPbFebu7
6teSF04ei3utFf9h+UmcxgNmtGR1BuILEs5ERKI2KDLfr90+ltKhDZu6gOBrcOWxikx+bhojouvj0o+LdZt9zj
SaNPZTkym01zPoFv24xyAA3OUAc1WESHKcuPbwO18LIopmURoEROB3dYN8veuTfekmYPv3tHOaLdZaL66oJkla
rJBcHWYlr/ob0/gElFmn+20y/kK7oq9vEP/oOSMYgtiyCWmBEgcnm6rIQvklFxzt9FFMz06+2I5/W0OSRnr371
Pb1nukHFXHJC5bDRMbnR7JobKhPAcDibzl2iNt/uWNX3K7L3Ddh1hCHFF/Dl+won2HJsfItOvbfxVoL3fs1Rk6
+FXvO5QRqcrQVOKN/z2cn2Y8N/bLIr86AH3+J7r4fGAspyqx985VMKzzl5OHvi+DzGDmmuzgtHpB3/R0NRbWgb
W5vL3KbrrbH+QkU2DIlcp+DOYysnVNtDxlFJVSDfvHxBAeOYwm9sjzvrslpHMqkltSmiqOnuU/shfPtAdYyxoT
TDV11R+TJNQo80Mq7cJUd9NeiYi+TjorpN5qtJW9/XQIPQjOd/mv4dwGWQkS14CJ/5Em4ONdEJnJzwU4FndrLG
h76IWczBM+3grhCVWBWf5EohxV8rMEJD7m3HeP6koPo4uxTylRkhSFO8GgP0aFR5cEGsjnIhyPVcf7ad1L9t+A
3ajzpPW9m+pcdgWqvamCT47B/Uc6S/nN8VA+7bIdXVCqTsNiyyNoNSEplk8Qi97nz2ZF6/UcxdoD6aVD/HvJA5
3usmluCKuylnFbFX9eIyOF0DGppo3RsP8ka61pSt+jXrn95xkisO1u/Efmt8lb0bhrPET+NEkA==
                </xenc:CipherValue>
             </xenc:CipherData>
          </xenc:EncryptedData>
       </Security>
    </env:Header>
    <env:Body>
       <csw:GetRecords maxRecords="10" outputFormat="application/xml"
outputSchema="urn:oasis:names:tc:ebxml-regrep:xsd:rim:3.0" resultType="results"
service="CSW" startPosition="1" version="2.0.2"
xmlns:aoi="http://www.esa.int/xml/schemas/mass/aoifeatures"
xmlns:common="http://exslt.org/common"
xmlns:csw="http://www.opengis.net/cat/csw/2.0.2"
xmlns:gml="http://www.opengis.net/gml" xmlns:ogc="http://www.opengis.net/ogc"
xmlns:portal="http://www.esa.int/mass" xmlns:rim="urn:oasis:names:tc:ebxml-
regrep:xsd:rim:3.0" xmlns:serviceNs="http://www.opengis.net/cat/wrs/1.0"
xmlns:wrs="http://www.opengis.net/cat/wrs/1.0">
          <csw:Query typeNames="rim:RegistryPackage rim:ExtrinsicObject
rim:ExtrinsicObject rim:ExtrinsicObject_acquisitionPlatform
rim:ExtrinsicObject_dataLayer rim:Association_acquisitionPlatAsso
rim:Association_dataLayerAsso rim:Classification rim:ClassificationNode">
             <csw:ElementSetName
typeNames="rim:RegistryPackage">full</csw:ElementSetName>
             <csw:Constraint version="1.1.0">
                <ogc:Filter>
                   <ogc:And>
                      <ogc:BBOX>

   <ogc:PropertyName>/rim:ExtrinsicObject/rim:Slot[@name=&amp;quot;urn:ogc:def:ebRIM-
Slot:OGC-06-131:multiExtentOf&amp;quot;]/wrs:ValueList/wrs:AnyValue[1]</ogc:PropertyName>
                         <gml:Envelope srsName="EPSG:4326"
xmlns="http://www.esa.int/xml/schemas/mass/aoifeatures"
xmlns:sch="http://www.ascc.net/xml/schematron"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
                            <gml:lowerCorner>23.1368 -40.7547</gml:lowerCorner>
                            <gml:upperCorner>58.3726 32.2642</gml:upperCorner>
                         </gml:Envelope>
                      </ogc:BBOX>
                      <ogc:PropertyIsEqualTo>

   <ogc:PropertyName>/rim:ExtrinsicObject/rim:Slot[@name=&amp;quot;urn:ogc:def:ebRIM-
Slot:OGC-06-131:parentIdentifier&amp;quot;]/rim:ValueList/rim:Value[1]</ogc:PropertyName>

   <ogc:Literal>urn:ogc:def:EOP:ESA:SIMU.EECF.ENVISAT_MER_FR_xS</ogc:Literal>
                      </ogc:PropertyIsEqualTo>
```

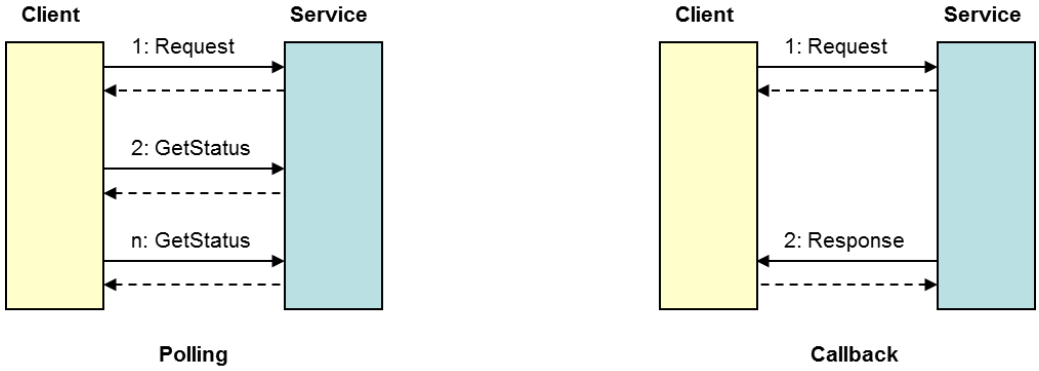
```
                    <ogc:PropertyIsEqualTo>

   <ogc:PropertyName>/rim:ExtrinsicObject/@objectType</ogc:PropertyName>
                       <ogc:Literal>urn:x-ogc:specification:csw-
ebrim:ObjectType:EO:EOProduct</ogc:Literal>
                    </ogc:PropertyIsEqualTo>
                    <ogc:PropertyIsGreaterThanOrEqualTo>

   <ogc:PropertyName>/rim:ExtrinsicObject/rim:Slot[@name=&quot;urn:ogc:def:ebRIM-
Slot:OGC-06-131:beginPosition&quot;]/rim:ValueList/rim:Value[1]</ogc:PropertyName>
                       <ogc:Literal>2009-06-26T00:00:00.000</ogc:Literal>
                    </ogc:PropertyIsGreaterThanOrEqualTo>
                    <ogc:PropertyIsLessThanOrEqualTo>

   <ogc:PropertyName>/rim:ExtrinsicObject/rim:Slot[@name=&quot;urn:ogc:def:ebRIM-
Slot:OGC-06-131:endPosition&quot;]/rim:ValueList/rim:Value[1]</ogc:PropertyName>
                       <ogc:Literal>2009-06-26T23:59:59.000</ogc:Literal>
                    </ogc:PropertyIsLessThanOrEqualTo>
                    <ogc:PropertyIsEqualTo>

   <ogc:PropertyName>$acquisitionPlatform/@objectType</ogc:PropertyName>
                       <ogc:Literal>urn:x-ogc:specification:csw-
ebrim:ObjectType:EO:EOAcquisitionPlatform</ogc:Literal>
                    </ogc:PropertyIsEqualTo>
                    <ogc:PropertyIsEqualTo>

   <ogc:PropertyName>$acquisitionPlatAsso/@sourceObject</ogc:PropertyName>
                       <ogc:PropertyName>/rim:ExtrinsicObject/@id</ogc:PropertyName>
                    </ogc:PropertyIsEqualTo>
                    <ogc:PropertyIsEqualTo>

   <ogc:PropertyName>$acquisitionPlatAsso/@associationType</ogc:PropertyName>
                       <ogc:Literal>urn:x-ogc:specification:csw-
ebrim:AssociationType:EO:AcquiredBy</ogc:Literal>
                    </ogc:PropertyIsEqualTo>
                    <ogc:PropertyIsEqualTo>

   <ogc:PropertyName>$acquisitionPlatAsso/@targetObject</ogc:PropertyName>
                       <ogc:PropertyName>$acquisitionPlatform/@id</ogc:PropertyName>
                    </ogc:PropertyIsEqualTo>
                 </ogc:And>
              </ogc:Filter>
           </csw:Constraint>
        </csw:Query>
     </csw:GetRecords>
   </env:Body>
</env:Envelope>
```

Figure 14: Example of Service Request

7.2.1.2 Response

The service response for a synchronous operation is not affected by the authorisation mechanism and remains as defined in the service interface.

For asynchronous operations, the service response is either obtained by polling (e.g. GetStatus operation) or by making a request in the opposite direction (callback) as illustrated in the figure below. In the latter case, the client endpoint is typically provided in the SOAP header of the original request using WS-Addressing.



In this Best Practice, the choice is left whether to set-up or not an authentication/authorisation layer for asynchronous service responses. This shall be decided by agreement of all the parties of the circle of trust that provide or use asynchronous services.

If the choice is made to set-up an authentication/authorisation layer for asynchronous service responses, then the response shall be protected by the same encryption and signature as defined for the service request and authentication.

- For protocols based on polling, the client and SP keep their initial roles and the use cases are exactly the same than those covered previously.
- For protocols based on WS-Addressing, the SP takes the role of the client and conversely. The sequence of steps is then as follows:

  1. The SP prepares the response to the endpoint mentioned in the WS-Addressing.
  2. This response is addressed to the PEP of the Client.
  3. The SP attaches to the asynchronous response a SAML token authenticating itself. This requires the SP to access an IdP (STS) belonging to the circle of trust, the user registry of the accessed IdP containing an entry that is a surrogate for the SP. For the ease of implementation and integration, it is recommended to have an architecture with one single IdP for the circle of trust. Other architectures with multiple IdP are possible however, including architectures where SP and IdP reside on the same entity.
  4. The asynchronous response is returned to the address provided in the WS-Addressing of the request. This will be the address of a PEP that knows the public key of the STS providing the SAML token attached to asynchronous response, for the purpose of signature verification. If multiple IdP architecture is chosen, then the PEP shall know the set of public keys associated to all these IdP.

7.2.1.3 Authorisation Exception

When a SOAP request is unauthorised on a server, this server shall respond with a SOAP fault indicating the exception reason/code (see first column of Table 1) according to the exception mechanism used by this service.

For OGC services, the exception report message shall be returned as specified in section 8 of the OGC Web Services Common Standard document [NR16]. However, the HTTP status code value shall be set to "401 Unauthorized" in all cases except when the exception code is set to AuthorisationFailed where "403 Forbidden" should be used instead.

The authorisation specific exception codes are defined in the table below.

"exceptionCode" value	Meaning of code	"locator" value
MissingToken	Request does not contain a SAML token.	Omit locator parameter
InvalidToken	Request does not contain a valid SAML token, or the token cannot be decrypted, or the token signature is invalid, or the token expired.	Omit locator parameter
TokenVersion	Request contains a SAML token with an unsupported version.	URI of SAML version supported
AuthorisationFailed	Request is for an operation that is not authorised by (the PEP of) this server.	Name of SAML attribute causing the authorisation failure

Table 1: Authorisation Exception Codes

An example is given below:

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
   <soapenv:Body>
      <soapenv:Fault>
         <soapenv:Code>
            <soapenv:Value>soap:Receiver</soapenv:Value>
         </soapenv:Code>
         <soapenv:Reason>
            <soapenv:Text xml:lang="en">A server exception was
encountered.</soapenv:Text>
         </soapenv:Reason>
         <soapenv:Detail>
            <ows:ExceptionReport xmlns:ows="http://www.opengis.net/ows/2.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/ows/2.0
http://schemas.opengis.net/ows/2.0/owsExceptionReport.xsd" version="1.0.0"
xml:lang="en">
               <ows:Exception exceptionCode="AuthorisationFailed" locator="o">
                  <ows:ExceptionText>Country of origin not
authorised</ows:ExceptionText>
               </ows:Exception>
            </ows:ExceptionReport>
         </soapenv:Detail>
      </soapenv:Fault>
   </soapenv:Body>
</soapenv:Envelope>
```

7.2.2 HTTP Binding

Protocol: HTTP/HTTPS encoding

7.2.2.1 Request

Note: the present section aims at complying with the "Authorization Request Header Field" defined in OAuth 2.0 Authorization Framework (see NR31]).

The SAML token shall be put in the HTTP header of the request, in a field named "Authorization". The syntax shall obey the following rules, expressed in Augmented Backus-Naur Form (ABNF) (see [NR30]):

```
authorization = "Authorization" ":" 1*SP credentials

credentials = "Bearer" 1*SP b64token
```

The `b64token` element shall be the SAML token, i.e. the xenc:EncryptedData element extracted from the RSTR, converted by a Base64 encoding (see [NR28]).

The following snippet of an HTTP GET request is provided as example.

```
GET http://aaa.bbb.gov/csw HTTP/1.1
Host: aaa.bbb.gov
…
Authorization: Bearer PHhlbmM6RW5jcnlwdGVkRGF0YSBUeXBlPSJodHRwOi8vd3d3LnczLm9yZy8yMDAx
LzA0L3htbGVuYyNFbGVtZW50IiB4bWxuczp4ZW5jPSJodHRwOi8vd3d3LnczLm9yZy8yMDAxLzA0L3htbGVuYy
MiPg … NEkA==
…
Content-Type: application/xml
Content-Length: …

<?xml version="1.0" encoding="UTF-8"?>
<csw:GetRecords maxRecords="10" … xmlns:wrs="http://www.opengis.net/cat/wrs/1.0">
   <csw:Query typeNames="rim:RegistryPackage rim:ExtrinsicObject
      …
   </csw:Query>
</csw:GetRecords>
```

Note that the SAML token is neither included in the request line nor in the message body of the HTTP request. Also, the syntax is independent of the HTTP method used (i.e. GET, POST, etc).

Important caution: Although the HTTP specification does not enforce a limit on the size of header, there exist practical limits imposed by the implementations of HTTP servers. Since the size of the encoded SAML token is relatively large (~5 KB, in our example), it is important to ensure that it can be handled without error by the underlying transport layer, which usually relies on several intermediate HTTP servers. As a rule of thumb, the limit of 8 KB is quoted for the total size of HTTP request line and header; in any case, the set of attributes included in the SAML token should be narrowed to the strict authorization needs[12].

[12] To improve the situation, a future version of this document could consider compression (e.g. deflate) of the SAML token before Base64 encoding. Furthermore, the SAML token could be included in the URL to allow for KVP encoding (http://…?Bearer=…&…). However, in this case, additional URL encoding (after Base64 encoding) should be performed.

7.2.2.2 Response

The service response is not affected by the authorisation mechanism and remains as defined in the service interface.

7.2.2.3 Authorisation Exception

When an HTTP request is unauthorised on a server, this server shall respond with an exception message indicating the exception reason/code (see first column of Table 1) according to the exception mechanism used by this service.

For OGC services, the exception report message shall be returned as specified in section 8 of the OGC Web Services Common Standard document [NR16]. However, the HTTP status code value shall be set to “401 Unauthorized” in all cases except when the exception code is set to AuthorisationFailed where “403 Forbidden” should be used instead.

The authorisation specific exception codes are the same as the ones defined above in Table 1.

For example:

```
HTTP/1.1 403 Forbidden
Content-Type: application/xml
…

<?xml version="1.0" encoding="UTF-8"?>
<ows:ExceptionReport xmlns:ows="http://www.opengis.net/ows/2.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/ows/2.0
http://schemas.opengis.net/ows/2.0/owsExceptionReport.xsd" version="1.0.0"
xml:lang="en">
   <ows:Exception exceptionCode="AuthorisationFailed" locator="o">
      <ows:ExceptionText>Country of origin not authorised</ows:ExceptionText>
   </ows:Exception>
</ows:ExceptionReport>
```

8 Web-SSO Integration

The present section provides specific information to use the present Best practice in the context of an integration of a Web-SSO system with a Service Provider. More specifically, it covers the integration of a C2B type authentication environment (e.g. Shibboleth [OR2] or OpenAM [OR3]) with a B2B service authorisation environment based on SAML tokens, as expressed in HL-REQ060.

As an example of Web-SSO system, EO-SSO is an operational SSO system based on Shibboleth for ESA Web-based Applications, which could be adopted by other EO Providers as well. It features typical user management functions (login-authentication, registration-account maintenance, access control, and authorization). It allows ESA Web applications to outsource the sign-on process and offers a given user access to several ESA EO Portals with one single sign-on on his browser.

A given Web-SSO system defines a specific security domain, which is separated from the security domain defined by the service provider. In concrete terms, the two security domains rely on different security tokens; the Web-SSO IdP authenticates Web Portal users without providing the SAML token defined by the present interface.

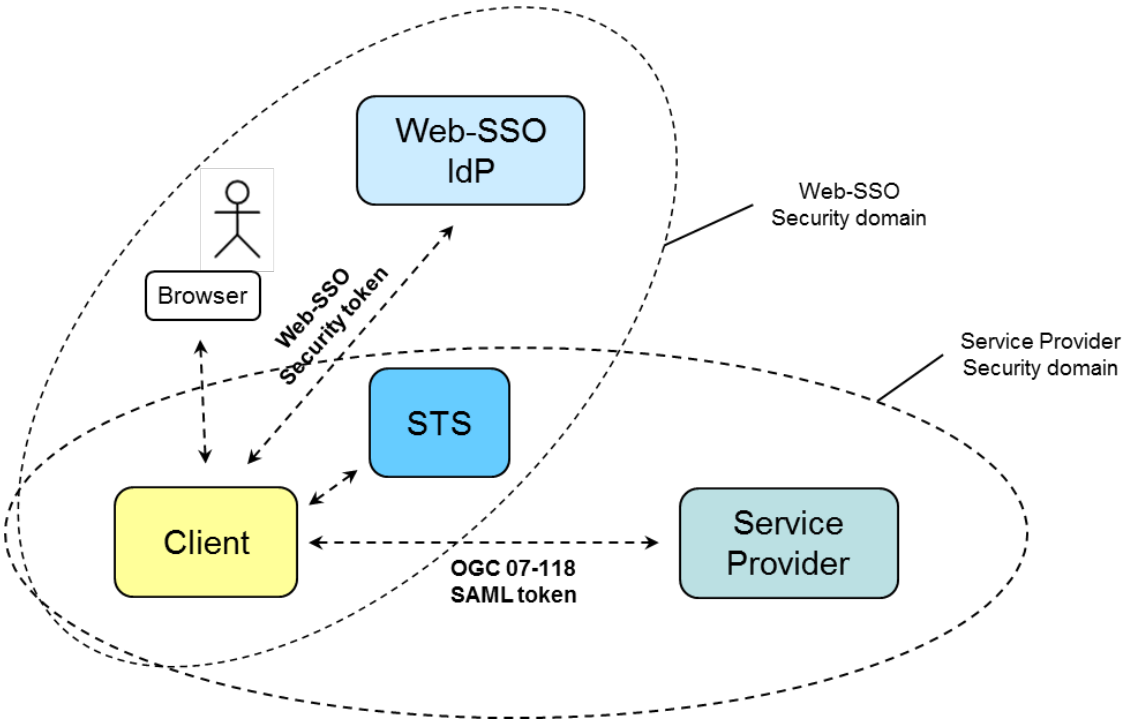


Figure 15: Web-SSO and Service Provider security domains

In this specific context, a secure gateway shall be established between the two security domains, relying on a trust relationship. RST with signature (see 6.4.3.3 and 6.4.3.4) shall be used for that purpose.

In order to establish this trust relationship, a given Client *C* of the Web-SSO security domain shall provide a certificate with its public key to the Service Provider STS. The trust relationship between *C* and service provider STS is established as soon as the service provider security administrator has put this public key in the keystore of the service provider STS.

From that point on, client *C* can obtain a SAML token for any Web-SSO authenticated users by issuing an RST with signature. The sequence of steps is as follows, for a user *U* that has not yet signed on the Web-SSO (this is largely simplified, in order to provide the most significant components and steps):

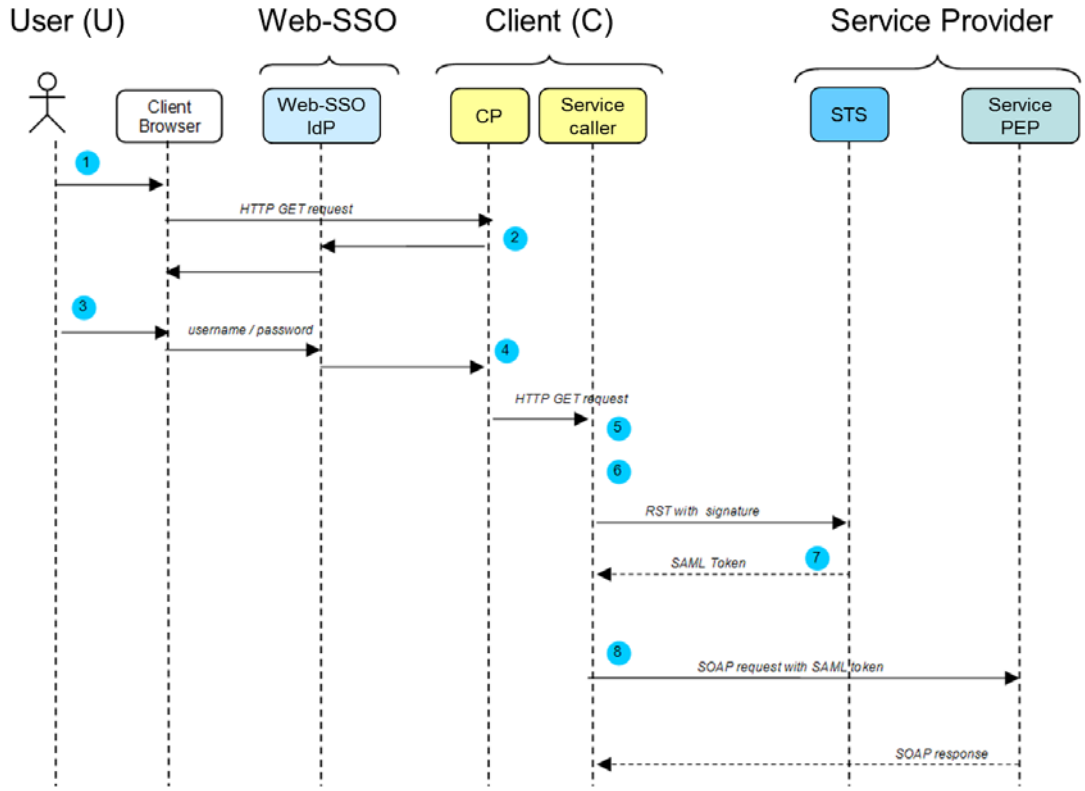


Figure 16: Web-SSO / Service Provider integration sequence diagram

1. The user *U* activates a function on client *C*, that shall use a Service Provider service
2. The Web-SSO checkpoint on *C* relays the authentication to the Web-SSO IdP
3. The user *U* enters his/her credentials and successfully signs on the Web-SSO IdP
4. The Web-SSO checkpoint (CP) on *C* sends a GET request containing Web-SSO user id in HTTP header
5. The client *C* prepares an RST with signature, by putting Web-SSO user id as username and by signing the request.
6. The client *C* issues the RST to the service provider STS.
7. The service provider STS verifies the RST signature and returns a signed and encrypted SAML token.
8. The client *C* issues service request(s) to the service provider PEP with the encrypted SAML token.

If the user has already signed-on on Web-SSO, then the system shall skip the sign-on process (steps 2 and 3) and the sequence resumes at step 4.

9 Security Considerations

The interface that is presented in the current document was designed according to a specific set of security requirements. Other application domains may want to take additional security measures, which are complementary to the minimal interface defined in the current document.

The present section identifies different types of attack or threats that are specific to the present interface; it provides for each of these types of attack or threat the answer or countermeasure, as entailed by the interface. When required, the distinction is made between RSTs and service requests, as well as on the protocol binding (SOAP or HTTP).

Type of attack / threat	Answer / countermeasures
Identity spoofing	If the IdP complies with the present Best Practice (see cases 6.4.3.1 and 6.4.3.2), then the sole artefact that conveys user identity, i.e. an evidence of authentication, is the SAML token, obtained by an RST with password. The IdP guarantees that the SAML token for user *X* is returned if and only if the credentials of *X* have been provided (see next threat topics related to password). If the IdP is an External Entity not complying with the present Best Practice (see cases 6.4.3.3 and 6.4.3.4), then the threat of identity spoofing has to be analysed at the level of this IdP, as well as the level of security gateway (Client in Figure 7, Figure 8, and Figure 15) that shall request SAML token to STS. The action of registering the public key of such External entity on the STS means that this STS *trusts* both external IdP and security gateway. If this is done, then the STS shall serve any RST secured by signature from that security gateway, with no further identity control. The signature verification shall guarantee that the RST that it has been issued by a trusted security gateway and that it has not been tampered with. For the service requests, the risk is the theft of SAML token, which could be rebound to a new service request issued by a malicious user. This risk is limited by putting short expiry time on SAML token; as the expiry time is part of the SAML token itself, it is protected from changes by signature. The expiry time and signature are both checked by the PEP. Also, HTTPS could be used to avoid (through encryption) the risk of such forged service request. Another countermeasure consists in putting an IP filter to check whether the client is authorised to make service requests.
Man-in-the-middle	For RST (with password or with signature): the transport protocol is HTTPS, which is based on SSL or TLS; SSL or TLS includes a certificate mechanism to protect against man-in-the-middle attack. For service requests (if no secured HTTP is used): the signature protocol guarantees that the emitter of SAML token is a trusted STS and that the token has not been tampered with; this is checked by the PEP. The threat is therefore located on the message payload (SOAP body or HTTP message body) or its binding with SAML token. Such threat is analysed in Identity Spoofing, Data integrity, Data confidentiality topics.

Data integrity	The signature protocol enforced on SAML Token allows for the verification of its own data integrity, at the PEP level. The data integrity of the message payload may be checked by another signature mechanism on the SOAP body or HTTP message body. Such signature should be bound in some way with the header, in order to avoid the risk of forged service request (see "identity spoofing" topic).
Data confidentiality / privacy violation	Encryption of SAML token (both for RSTs and service requests) guarantees that no entity excepting the target PEP can read conveyed user attributes. The mechanism presented for STS delegation (see 6.4.3.2 and 6.4.3.4) involves a SAML token delivered in clear but 1° it is transferred through HTTPS (ensuring point-to-point encryption), 2° it is delivered only for RST having valid credentials or signature (see "*Identity spoofing*" analysis above) and 3° the SAML token is unsigned, hence unusable for protected service requests. For service request, the data confidentiality of the message payload may be enforced by using HTTPS protocol or by encryption of the SOAP body or HTTP message body. The user registry (e.g. LDAP) is protected by password, which is known only by security officer and IdP. The IdP is a "trusted" entity.
Replay attack	For RSTs: the transport protocol is HTTPS, which is based on SSL or TLS; SSL or TLS includes a "nonce" mechanism to protect against replay attack. For service requests, the use of HTTPS provides the same protection; if unsecured HTTP is used: the risk of unauthorised access through replay of a past service request is limited by putting short expiry time on SAML token, which is checked by the PEP. Also, a replay protection may be implemented using a hashing function or digital signature which provides a unique identifier that can be used to determine if the same message is received multiple times.
Denial of Service	Web service is susceptible to message flooding denial of service attacks from message replay. "replay detection" mechanisms can be used.
Password Disclosure	If the IdP complies with the present Best Practice (see cases 6.4.3.1 and 6.4.3.2), then RST with password is used, which relies on HTTPS. The password is therefore encrypted during transmission from client to IdP. It is an implementation decision whether deployments use an LDAP registry. If LDAP is used, the LDAP registry is protected by password, which is known only by security officer and IdP. The user passwords are stored encrypted on LDAP registry. Secure LDAP (SLDAP) protocol may be used also. The risk of password disclosure is therefore limited to known and usual factors, which can be mitigated by enforcing an adequate password policy (out of scope of the present interface). If the IdP is an External Entity not complying with the present Best Practice (see case 6.4.3.3 and 6.4.3.4), then the RST with signature is used, which contains no password. The threat of password disclosure shall be analysed at the level of the external IdP, which is out of scope of the present Best Practice.

Password Cracking / Guessing	If the IdP complies with the present Best Practice (see cases 6.4.3.1 and 6.4.3.2), then RST with password is used. The risk of password cracking/guessing is limited to known and usual factors, which can be mitigated by enforcing an adequate password policy (out of scope of the present interface). If the IdP is an External Entity not complying with the present Best Practice (see case 6.4.3.3), then the RST with signature is used, which contains no password. The threat of password cracking/guessing shall be analysed at the level of the external IdP, which is out of scope of the present Best Practice.
Unauthorised access	The authorisation to Web services relies on PEP and associated access policy rules. The rules are based on asserted user attributes in the SAML token. The fact that these attributes match the actual requesting user relies on authentication.

The following table covers implementation-dependant threats.

Type of attack / threat	Answer / countermeasures
SQL injection	If a RDBMS is used for user registry, there is a risk of SQL injection for the authentication operation, i.e. a hacker enters as user id or password, some malicious character string that are interpreted by SQL engine. Such risk can be prevented by performing string validation and character escaping on input user id / password strings, before SQL lookup (out of scope of the present interface).
LDAP injection	If a LDAP registry is used for user registry, there is a risk of LDAP injection, i.e. a hacker enters as user id or password, some malicious character string that are interpreted by LDAP or JNDI API. See http://www.blackhat.com/presentations/bh-europe-08/Alonso-Parada/Whitepaper/bh-eu-08-alonso-parada-WP.pdf Such risk can be prevented by performing string validation and character escaping on input user id / password strings, before LDAP lookup (out of scope of the present interface).

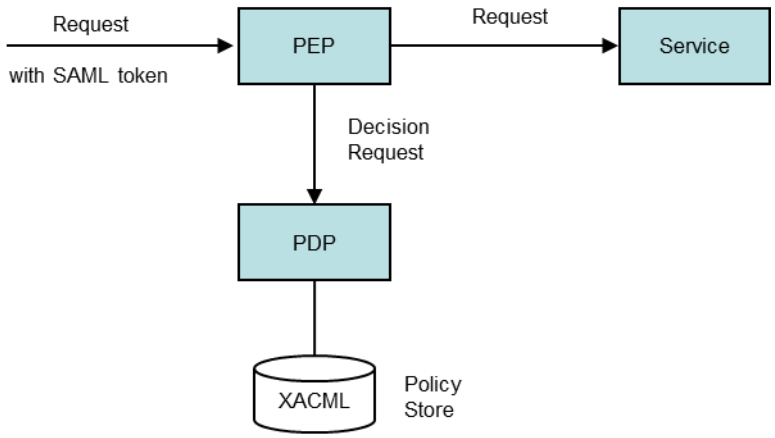
10 Authorisation Use Cases (Non-Normative)

As explained before, authorisation rules that grant access to Web services shall be evaluated by a dedicated PEP that wraps such services. However, the PEP treatments and the way access rules are stored and evaluated are not in the scope of the present document. The present section provides non-normative information about this topic.

In order to separate responsibilities, a PEP typically relies on a PDP (Policy Decision Point) that performs the actual evaluation of access rules based:

- on the request payload (i.e. the SOAP body or HTTP request line / message body),
- on the attributes of the SAML token, if any, present in the SOAP header or HTTP header, and
- on "external" elements (e.g. current time, configuration parameters).

Each PDP should have a dedicated policy store, where needed access rules or policies can easily be stored, retrieved and maintained.



The rules used by each PDP should be expressed in a standard syntax: the eXtended Access Control Markup Language (XACML) is recommended here. XACML (see [NR21]) is, in essence, a declarative access control policy language implemented in XML. It is worth mentioning here also GeoXACML (see [NR25]), which is an extension of XACML for the declaration and enforcement of geo-specific access restrictions for geographic data.

The following provides use cases and examples of policy rules, with XACML fragments implementing them. More comprehensive examples shall be found in Annex E.

10.1 Use Case: restrict access for time period

Generic policy rule:

Restrict data access for a given time period

Analysis:

XACML allows to define Rules based on “environment attributes”, such as date and time. A rich set of functions for handling date, time and dateTime values (as defined in the W3C XML Schema specification) are predefined in XACML.

Example:

Although able to access the service the user cannot access images from period t1=09:00:00 to t2=12:00:00.

The time restriction can be expressed as a Condition in an XACML rule as follows:

```
<Condition>
   <Apply FunctionId="urn:oasis:names:tc:xacml:2.0:function:time-in-range">
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-one-and-only">
         <EnvironmentAttributeDesignator
            AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-time"
            DataType="http:www.w3.org/2001/XMLSchema#time"/>
      </Apply>
      <AttributeValue
         DataType="http:www.w3.org/2001/XMLSchema#time">09:00:00</AttributeValue>
      <AttributeValue
         DataType="http:www.w3.org/2001/XMLSchema#time">12:00:00</AttributeValue>
   </Apply>
</Condition>
```

See Annex E for a more comprehensive example.

10.2 Use Case: enforce rules for specific group of users

Generic policy rule:

Enforce rules, like temporal restriction seen before, for specific group of users

Analysis:

XACML allows defining rules which target specific subjects. The rule for the current requirement can be expressed by targeting the group of users whose access shall be regulated together with a time restriction condition.

Needless to say, the group of users shall be targetable through an attribute contained in the SAML authentication token. In this way, a Rule with the following target could be defined:

Example:

Enforce rule for the users having the role "guest".

```
<Target>
   <Subjects>
      <Subject>
         <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
               guest
            </AttributeValue>
            <SubjectAttributeDesignator
               AttributeId="urn:ogc:um:eop:0.0.4:saml:role"
               DataType="http://www.w3.org/2001/XMLSchema#string"/>
         </SubjectMatch>
      </Subject>
   </Subjects>
</Target>
```

where `AttributeId="urn:ogc:um:eop:0.0.4:saml:role"` is a user-defined attribute contained in the XACML decision request which holds the suitable SAML Token attribute value identifying the group of users subjects to the Rule.

Notice that a Rule Target can match more than one Subject.

See Annex E for a more comprehensive example.

10.3 Use Case: restrict access to the type of data

Generic policy rule:

Restrict access to the type of data e.g. high or low resolution data

Analysis:

XACML allows to define Rules which target specific attributes of the resource to access. However, we assume that this information is either contained in the client request to the Service, or in a configuration file.

Notice that, building a Rule restricting access for certain data values but these data values are not provided in input, can result in an Indeterminate Policy (Indeterminate means that an error occurred or some required value was missing, so a decision cannot be made).

Example:

See Annex E.

10.4 Use Case: restrict access to data based on the age of the data

Generic policy rule:

Restrict access to data based on the age of the data

The age of data is an essential parameter to be considered for some products within EUMETSAT data policy (for instance at the moment Meteosat data are only accessible for retrieval from the archive 24 hours after sensing time).

Analysis:

If the age of data is a piece of information contained in the service request, it is possible to define a rule which sets restrictions on the access to the data based on their age.

Example:

For example, the following Condition evaluates to true if the current dateTime is greater than the acquisition end time of the data + 24 hours.

```
<Condition>
   <Apply FunctionId="urn:oasis:names:tc:xacml:2.0:function:dateTime-greater-than-or-
equal">
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:dateTime-one-and-only">
         <EnvironmentAttributeDesignator
            AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-dateTime"
            DataType="http://www.w3.org/2001/XMLSchema#dateTime"/>
      </Apply>
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:dateTime-add-
dayTimeDuration">
         <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:dateTime-one-and-
only">
            <ResourceAttributeDesignator
               AttributeId="urn:ogc:def:ebRIM-Slot:OGC-06-131:endPosition"
               DataType="http://www.w3.org/2001/XMLSchema#dateTime"/>
         </Apply>
         <AttributeValue DataType="http://www.w3.org/TR/2002/WD-xquery-operators-
20020816#dayTimeDuration">
            <xf:dt-dayTimeDuration>PT24H</xf:dt-dayTimeDuration>
         </AttributeValue>
```

```
      </Apply>
   </Apply>
</Condition>
```

where `AttributeId="urn:ogc:def:ebRIM-Slot:OGC-06-131:endPosition"` is a user-defined attribute contained in the XACML decision request which holds the corresponding value of the service request.

10.5 Use Case: imposing geographical constraints

Generic policy rule:

Imposing geographical constraints, i.e. area of interest (AOI), allowing some users to access more areas than others.

Analysis:

XACML is a general-purpose access control policy language and does not include specific functions and attributes to handle geographical rules. Given that it is also an extensible language, the user can add his/her own attributes and functions, or, better, in this case, he/she can integrate the XACML rules with GeoXACML [NR25], which specifically addresses geographical constraints.

10.6 Use Case: access and check source, content, user credentials and time

Generic policy rule:

Access and check source, content, user credentials and time

Analysis:

XACML rules targets the following groups of attributes:

- Subject
- Resource
- Action
- Environment

A rich set of attributes are predefined for each group together with functions to handle them, according to their types. Additionally, XACML can be extended with user defined attributes and functions.

10.7 Use Case: restricting access to users from certain geographic locations.

Generic policy rule:

Restricting access to users from certain geographic locations.

Analysis:

An XACML Rule can be defined to restrict access to users from geographical locations provided that this information is contained in the request to the Service Provider.

For example, if the authentication is performed according to the present "User Management Interfaces for Earth Observation" Best Practice, then the request may contain a SAML Token with attributes defined according to the "GMES Minimum Profile"; one of these attribute is the "country of origin" of the subject requesting access. Consequently, this attribute will be embedded in the XAML decision request and a Rule can be defined accordingly.

Example:

See Annex E.

10.8 Use Case: route service access based on user type

Generic policy rule:

Route a service access based on user type.

Note: This would for example allow a "scientific" user request to be routed to service 1 (e.g. DLR catalogue/ordering service) and a "commercial" user request to be routed to service 2 (e.g. Infoterra catalogue/ordering service).

Analysis:

This requirement could be met using the XACML Obligations; the Obligation is defined as follows:

"Obligation - An operation specified in a policy or policy set that should be performed by the PEP in conjunction with the enforcement of an authorisation decision"

In our case, the operation to be carried out is sending the request to the suitable provider; for each user type value, a policy can be defined with the following features:

- a rule matching a target subject type and having effect "permit";
- an obligation to send the request to the suitable Service Provider if the policy evaluates to "permit";

Annex A: Abstract Test Suite (Normative)

1 Conformance Test Class: The core

1.1 Test Module M.1 Basic requirements

This Test Module is made up of Abstract Test Cases which establishes preliminaries conditions to the actual test cases, such as the protocol bindings, messaging framework, adoption of specification and algorithms to encrypt and sign the messages.

1.1.1 ATC-1.1 SOAP Binding of the authentication request/response messages

Test case identifier	"urn:ogc:cite:ats:um:07-118:soap-binding:sts"
Test assertion [purpose]	STS operations shall support the embedding of requests and responses in SOAP messages. Only SOAP messaging (via HTTPS/POST) with document/literal style shall be used. Messages should conform to SOAP 1.2. The following assertions shall hold: • The SOAP Header holds a detached signature [if applicable] • The SOAP Body holds the RST (Request Security Token)
Test method	Send an RST embedded in a SOAP Envelope over the HTTPS protocol; verify that a response is returned (even in case of failure) embedded in a SOAP Envelope over the HTTPS protocol. The response is either an RSTR (Request Security Token Response) or a SOAP Fault. The SOAP Envelope shall be compliant with version 1.2 of SOAP (namespace http://www.w3.org/2003/05/soap-envelope)
Reference	Clause 6.2
Test type	Capability

1.1.2 ATC-1.2 HTTP Binding of the authentication request/response messages

Test case identifier	"urn:ogc:cite:ats:um:07-118:http-binding:sts"
Test assertion [purpose]	STS operations shall support the embedding of requests and responses in HTTP messages (via HTTPS/POST). The following assertions shall hold: • The HTTP Header holds a detached signature [if applicable]. • The HTTP Body holds the RST (Request Security Token).
Test method	Send an RST as the message body over the HTTPS protocol; verify that a response is returned (even in case of failure) as the message body over the HTTPS protocol. The response is either an RSTR (Request Security Token Response) or an OWS Exception Report message
Reference	Clause 6.2
Test type	Capability

1.1.3 ATC-1.3 SAML token encoding for authentication information

Test case identifier	"urn:ogc:cite:ats:um:07-118:saml-token"
Test assertion [purpose]	SAML is proposed to encode the user authentication token. A SAML token is made of the following statements: • Authentication statements: a typical authentication statement asserts Subject S authenticated at time t using authentication method m. • Attribute statements: a typical attribute statement asserts Subject S is associated with attributes X,Y, Z having values v1,v2,v3. The set of attribute statements returned in a SAML token shall be defined arbitrarily.
Test method	1. Send a valid RST (either with password or with signature) to the STS. The RST shall not contain information in addition to: • username/password if an RST with password is sent • username/signature if an RST with signature is sent 2. Check that the Body of the HTTP/SOAP response holds an encrypted SAML token. 3. Decrypt the SAML token using the default Relying Party's private key, and verify that the SAML token has the expected statements covering the (arbitrarily) defined set of attributes. **Pre-condition:** For carrying out this test, the client needs a copy of the default Relying Party's private key. The default Relying Party's private key is the private key corresponding to the default public key used by the STS in the encryption process when no reference to a Relying Party is given in the RST request. For testing purposes, a couple of private/public keys can be generated using available tools (for example, 'keytool' on JRE), where the certificate with the public key is self-signed by the STS itself.
Reference	Clauses 6.3
Test type	Capability

1.1.4 ATC-1.4 Encryption algorithm for SAML token

Encryption of the SAML token is performed by the STS when creating a RSTR.

Decryption of SAML token is performed by [the PEP of] the Service Provider.

Test case identifier	"urn:ogc:cite:ats:um:07-118:encryption"
Test assertion [purpose]	The encryption algorithm used for the SAML token is the AES-128. The symmetric AES-128 key used for encryption is made available to the recipient as follows: • The key is encrypted using the asymmetric RSA encryption algorithm with the public key of the recipient. • The resulting value is added to the encrypted message, using the XML Encryption [NR17] and XML Signature [NR18] specifications

Test method	1. Send a valid RST (either with password or with signature) to the STS. The RST shall not contain information in addition to:
	  • username/password if an RST with password is sent
	  • username/signature if an RST with signature is sent
	2. Decrypt the AES-128 symmetric key contained in the response using the default Relying Party's private key.
	3. Decrypt the SAML token using the AES-128 symmetric key and check that the result contains a valid SAML Assertion
	
	Pre-condition:
	For carrying out this test, the client needs a copy of the default Relying Party's private key. The default Relying Party's private key is the private key corresponding to the default public key used by the STS in the encryption process when no reference to a Relying Party is given in the RST request.
	For testing purposes, a couple of private/public keys can be generated using available tools (for example, 'keytool' on JRE), where the certificate with the public key is self-signed by the STS itself.

Reference	Clauses 6.4.1
Test type	Basic

1.1.5 ATC-1.5 Encryption key retrieval

The STS uses a default public key for encrypting the SAML Token, unless the client provides a reference to an STS configured public key, specific for a SP.

Test case identifier	"urn:ogc:cite:ats:um:07-118:encryption-sp"
Test assertion [purpose]	The Relying Party public key used in the encryption process is indicated by the client. The STS shall be able to encrypt the SAML token using the public key reference provided in the client request.
Test method	1. Send a valid RST (either with password or with signature) which includes the "AppliesTo" element; the "Address" field shall contain a reference to the public key of the Relying Party to be used for encryption 2. Check that the Body of the HTTP/SOAP response holds encrypted data. 3. Decrypt the AES-128 symmetric key contained in the response using the Relying Party's private key. 4. Decrypt the SAML token using the AES-128 symmetric key and check that the result contains a valid SAML Assertion **Pre-condition:** For carrying out this test: • the client needs a copy of the Relying Party's private key. • the STS is configured with the Relying Party's public key For testing purposes, a couple of private/public keys can be generated using available tools (for example, 'keytool' on JRE), where the certificate with the public key is self-signed by the STS itself.
Reference	Clauses 6.4.1.1
Test type	Basic

1.1.6 ATC-1.6 Digest algorithm for signing SAML tokens

Test case identifier	"urn:ogc:cite:ats:um:07-118:digest"
Test assertion [purpose]	The secure hash SHA-1 digital signature message digest algorithm is proposed. The SAML Token is signed before encryption. The XML signature <ds:Signature> element of can be used for signature, according to WS-Security specification.
Test method	1. Send an RST (either with password or with signature) to the STS. The RST shall not contain information in addition to: • username/password if an RST with password is sent • username/signature if an RST with signature is sent 2. Check that the response contains an encrypted SAML token and decrypt it following the process specified in ATC 1.4 3. Digest the SAML token using the SHA-1 algorithm 4. Decrypt the signature using the private key of the STS. 5. Compare the digest obtained at step 3 with the value resulting from step 4. The two values shall match. **Pre-condition:** For carrying out this test, the client needs: • a copy of the STS private key. • a copy of the default Relying Party's private key. For testing purposes, couples of private/public keys can be generated using available tools (for example, 'keytool' on JRE), where the certificate with the public key is self-signed by the STS itself.
Reference	Clause 6.4.2
Test type	Basic

1.2 Test Module M.2 Authentication

1.2.1 Test Module M.2.1 RST with password

This Test Module is made up of Abstract Test Cases related to the management of RSTs with password and responses.

- The first test case is related to the following scenario: the client issues an RST with password to the STS without indicating the Identity Provider in charge of fulfilling the request; this is the default case, and implies that the recipient STS shall fulfil the request locally.

- The second test case is related to the following scenario: the client issues an RST with password to the STS explicitly indicating the Identity Provider in charge of fulfilling the request. In this case, the recipient STS delegates to a different Identity Provider the authentication and SAML token generation.

1.2.1.1 ATC-2.1.1 Local STS as IdP

Test case identifier	"urn:ogc:cite:ats:um:07-118:rst-passw-1"
Test assertion [purpose]	The STS is assumed to be the request designated IdP. In this use case the RST contains only the user credentials (username, password).
Test method	The client issues an RST with: • mandatory username/password information; Verify that the client receives a valid SAML token which is signed and encrypted The protocol to be used for the message exchange is (SOAP over) HTTPS. The SAML token shall be returned in the (SOAP) HTTPS Body of the response.
Reference	Clause 6.4.3.1
Test type	Capability

1.2.1.2 ATC-2.1.2 Delegate STS as IdP

Test case identifier	"urn:ogc:cite:ats:um:07-118:rst-passw-2"
Test assertion [purpose]	The STS is not the request designated IdP. In this use case the RST contains an identifier for the delegate STS, in addition to user credentials (username, password).
Test method	The client issues an RST with: • mandatory username/password information; • an identifier for the addressed (delegate) STS (in the "DelegateTo" sub tree) Verify that the client receives a valid SAML token which is signed and encrypted. The protocol to be used for the message exchange is (SOAP over) HTTPS. The SAML token shall be returned in the (SOAP) HTTPS Body of the response.
Reference	Clause 6.4.3.2
Test type	Capability

1.2.2 Test Module M.2.2 RST with signature

This Test Module is made up of Abstract Test Cases related to the management of RST with signature and responses. The scenarios covered are very similar to those depicted for module M.2.1, with the only difference that the STS involved are only responsible for the generation of the SAML Token (the client is already authenticated).

1.2.2.1 ATC-2.2.1 Local STS with external IdP

Test case identifier	"urn:ogc:cite:ats:um:07-118:rst-sign-1"
Test assertion [purpose]	The STS is responsible for the SAML Token generation In this use case the RST contains only username and a valid signature
Test method	The client issues an RST with: • mandatory username information and signature Verify that the client receives a valid SAML token which is signed and encrypted. The protocol to be used for the message exchange is (SOAP over) HTTPS. The SAML token shall be returned in the (SOAP) HTTPS Body of the response.
Reference	Clause 6.4.3.3
Test type	Capability

1.2.2.2 ATC-2.2.2 Delegate STS with external IdP

Test case identifier	“urn:ogc:cite:ats:um:07-118:rst-sign-2”
Test assertion [purpose]	The STS delegates the SAML Token generation. In this use case the RST contains an identifier of the delegate STS, in addition to username and a valid signature
Test method	The client issues an RST with: • mandatory username information and signature • an identifier for the addressed (delegate) STS (in the “DelegateTo” sub tree) Verify that the client receives a valid SAML token which is signed and encrypted. The protocol to be used for the message exchange is (SOAP over) HTTPS. The SAML token shall be returned in the (SOAP) HTTPS Body of the response.
Reference	Clause 6.4.3.4
Test type	Capability

1.2.3 Test Module M.2.3 RST failure

This Test Module is made up of Abstract Test Cases related to the HTTP and SOAP binding for RST failures

1.2.3.1 ATC-2.3.1 RST failure with SOAP binding

Test case identifier	"urn:ogc:cite:ats:um:07-118:rst-failure:soap"
Test assertion [purpose]	The STS shall return a SOAP Fault with proper content when the SAML token cannot be delivered
Test method	The client issues an incorrect RST (either with password or with signature). Verify that the client receives a SOAP Fault such that: • the value of the Subcode element is: o "wst:InvalidRequest", if the request was invalid or malformed o "wst:FailedAuthentication", if the authentication failed (e.g. invalid password or signature) o "wst:RequestFailed", if the specified request failed (e.g. SAML version or AppliesTo not supported) o "wst:BadRequest", if the specified RequestSecurityToken is not understood. • the text of the Reason element provides details on the fault
Reference	Clause 7.1.1
Test type	Capability

1.2.3.2 ATC-2.3.2 RST failure with HTTP binding

Test case identifier	"urn:ogc:cite:ats:um:07-118:rst-failure:http"
Test assertion [purpose]	The STS shall return an exception message if a service request cannot be fulfilled. The exception shall clearly indicate raison of failure
Test method	The client issues an incorrect RST (either with password or with signature). Verify that an exception message is returned. The excepted response is an OGC OWS ExceptionReport tree containing an "exceptionCode" attribute set to: • "wst:InvalidRequest", if the request was invalid or malformed • "wst:FailedAuthentication", if the authentication failed (e.g. invalid password or signature) • "wst:RequestFailed", if the specified request failed (e.g. SAML version or AppliesTo not supported) • "wst:BadRequest", if the specified RequestSecurityToken is not understood. Additionally, the HTTP status code shall be set to: • "401 Unauthorized"
Reference	Clause 7.1.2
Test type	Capability

1.3 Test Module M.3 Authorisation

This Test Module is made up of Abstract Test Cases related to the management of service requests and responses.

The module is made up of two sub-modules, the former for services supporting the SOAP binding, the latter for service supporting the HTTP binding.

1.3.1 Test Module M.3.1 Autorisation for SOAP binding

1.3.1.1 ATC-3.1.1 SOAP Authorisation with synchronous response

Test case identifier	"urn:ogc:cite:ats:um:07-118:synchronous-authorisation:soap"
Test assertion [purpose]	Only an authorised client can access a requested protected service. The service request contains a SAML Token returned by a previous successful RST.
Test method	Verify that the service to be invoked is protected, i.e. its WSDL specifies WS-Security policies. The client issues a request containing a SAML token previously obtained through authentication; the SAML Token is included in the SOAP Header of the request, wrapped in a WS-Security element. Verify that the client is authorised to access the protected service, that is a successful response shall be returned.
Reference	Clauses 7.2.1
Test type	Capability

1.3.1.2 ATC-3.1.2 SOAP Authorisation with asynchronous response

NOTE: This abstract test case refers to an asynchronous service based on the "call-back" mechanism.

Test case identifier	"urn:ogc:cite:ats:um:07-118:asynchronous-authorisation:soap"
Test assertion [purpose]	Only an authorised client can access a requested protected service. The service request header contains a SAML Token returned by a previous successful RST and WS-Addressing information to allow dispatching of the response.
Test method	Verify that the service to be invoked is protected, i.e. its WSDL specifies WS-Security policies. The client issues a request containing a SAML token, previously obtained through authentication; the SAML Token is included in the SOAP Header of the request, wrapped in a WS-Security element. The Service Provider shall return a service response according to the following format: • The SOAP Header contains a SAML Token which authenticates the Service Provider, signed with the private key of the STS where the Service Provider is assumed to be registered • The SOAP Body contains the actual response of the service. **Pre-condition:** The client shall support the asynchronous communication for the requested service. The PEP of the client knows the public key of the STS authenticating the Service Provider.
Reference	Clauses 7.2.1
Test type	Capability

1.3.1.3 ATC-3.1.3 SOAP Service request failure

Test case identifier	"urn:ogc:cite:ats:um:07-118:authorisation-failure:soap"
Test assertion [purpose]	The Service Provider shall return a SOAP fault message if a service request cannot be fulfilled. The SOAP fault shall clearly indicate raison of failure
Test method	The client issues an invalid service request. Verify that a SOAP fault response is returned, such that: • the <Detail> element holds application specific information about the reason of failure. For an OGC service, the <Detail> element shall contain an OGC OWS ExceptionReport tree containing an "exceptionCode" attribute set to: • "MissingToken", if the SAML Token is missing • "InvalidToken", if the SAML Token is invalid (wrong encryption key/invalid signature/expired time) • "TokenVersion", if the SAML Token version is not supported • "AuthorisationFailed", if the SAML Token is valid, but the user is not authorized to access the resources
Reference	Clause 7.2.1
Test type	Capability

1.3.2 Test Module M.3.2 Autorisation for HTTP binding

1.3.2.1 ATC-3.2.1 HTTP Authorisation with synchronous response

Test case identifier	"urn:ogc:cite:ats:um:07-118:synchronous-authorisation:http"
Test assertion [purpose]	Only an authorised client can access a requested protected service. The service request contains a SAML Token returned by a previous successful RST, added as an HTTP Header.
Test method	Verify that the service to be invoked is protected, i.e. its WSDL specifies WS-Security policies. The client issues a request containing a SAML token previously obtained through authentication; the SAML token is added (base64 encoded) to an HTTP Header with the following format: "Authorization: Bearer *b64token*" Verify that the client is authorised to access the protected service, that is a successful response shall be returned.
Reference	Clauses 7.2.2
Test type	Capability

1.3.2.2 ATC-3.2.2 HTTP Service request failure

Test case identifier	"urn:ogc:cite:ats:um:07-118:authorisation-failure:http"
Test assertion [purpose]	The Service provider shall return an exception message if a service request cannot be fulfilled. The exception shall clearly indicate raison of failure
Test method	The client issues an invalid request. Verify that an exception message is returned. For an OGC service, the excepted response is an OGC OWS ExceptionReport tree containing an "exceptionCode" attribute set to: • "MissingToken", if the SAML Token is missing • "InvalidToken", if the SAML Token is invalid (wrong encryption key/invalid signature/expired time) • "TokenVersion", if the SAML Token version is not supported • "AuthorisationFailed", if the SAML Token is valid, but the user is not authorized to access the resources Additionally, the HTTP status code shall be set to: • "403 Forbidden" if the exceptionCode is set to "AuthorisationFailed" • "401 Unauthorized" in all the remaining cases
Reference	Clause 7.2.2
Test type	Capability

Annex B: Schemas (Normative)

Since the schemas of WS-Trust have many optional elements, we provided here a narrower schema that limits the degree of freedom of the standard schemas, focusing on RST and RSTR. When the underlying child schemas cannot be changed, English annotations are used to specify specific constraints. The constrained schema has been obtained by updating reference files from WS-Trust 1.3:
http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3.xsd

The constrained schema is compatible with the standard WS-Trust 1.3 (i.e. any service implementation conforming to constrained files shall also conform to the standard ones).

In the following, we provide, as support to the WS-Trust 1.3 schema, information on the structure of RST, RSTR, then the constrained `ws-trust.xsd` schema, then references to the SAML 1.1 and SAML 2.0 schemas and a reference to the WS-Security schema.

For the following two subsections, namespace prefixes are defined in the following table:

Prefix	Namespace
ds	http://www.w3.org/2000/09/xmldsig#
xenc	http://www.w3.org/2001/04/xmlenc#
wsa	http://www.w3.org/2005/08/addressing
wsp	http://schemas.xmlsoap.org/ws/2004/09/policy
wsse	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd
wst	http://docs.oasis-open.org/ws-sx/ws-trust/200512/

RequestSecurityToken (RST)

The schema for RequestSecurityToken is illustrated in the following diagram.

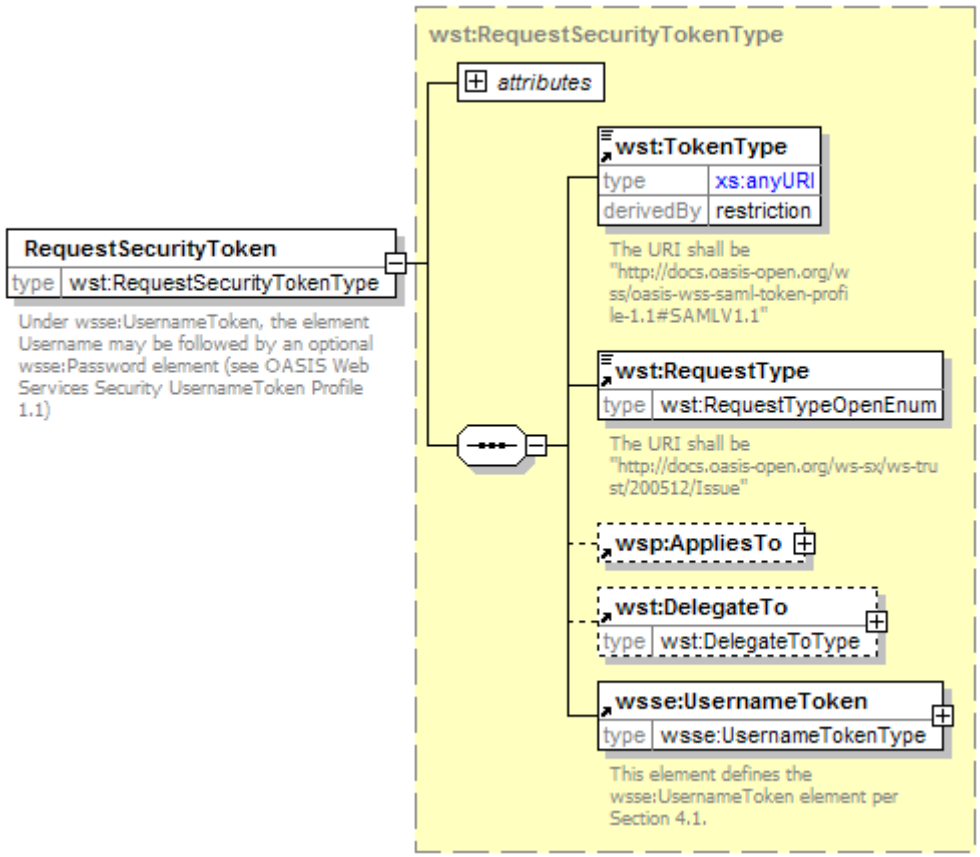


Figure 17: RequestSecurityToken schema

Refer to WS-Trust 1.3 (section 4.1 in [NR23]), with the following constraints:

wst:RequestSecurityToken/wst:TokenType

is REQUIRED and shall have the following URI, defined in [NR11]:
`http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV1.1`
or
`http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV2.0`

wst:RequestSecurityToken/wst:RequestType

is REQUIRED and shall have the following URI, (only Issue action is supported by the RST, for the moment):
`http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue`

wst:RequestSecurityToken/wsp:AppliesTo

is OPTIONAL. It shall contain a wsa:EndPointReference, itself containing a wsa:Address. This element (URI) is used to inform the STS about which Relying Party, if not the default one, is supposed to consume the SAML token; the STS can then use the associated public key to encrypt this token.

wst:RequestSecurityToken/wst:DelegateTo

is OPTIONAL. It shall contain a wsa:EndPointReference, itself containing a wsa:Address. It is used to require the STS to delegate user identification to an external trusted IdP. The wsa:Address shall contain an identifier (URI) known by the STS; from this identifier, the STS is supposed to retrieve the URL of the external IdP. If the DelegateTo element is absent, then the user identification is performed locally on the STS.

wst:RequestSecurityToken/wsse:UsernameToken

is REQUIRED. It contains the mandatory element Username, with the user id for which a SAML token is requested. In case of *RST with password*, a wsse:Password element is REQUIRED after Username. In case of *RST with signature*, it is REQUIRED to NOT put wsse:Password element.

Other elements defined in [NR23] are allowed in the RST but they shall be ignored by the STS complying with the present Best Practice.

In case of *RST with signature*, it is REQUIRED to put in the SOAP header a wsse:Security element containing a ds:Signature element. The ds:Signature shall contain the digital signature of the SOAP body (that contains the wst:RequestSecurityToken element), as a detached signature. The following

- The secure hash SHA-1 digital signature message digest algorithm is used, as supported by [NR15].
- The element that is signed is SOAP Body. The URI attribute of the <ds:Reference URI="..."> element shall refer to the <soap:Body> node being signed (using XPointer, see 4.3.3.3 in [NR18].
- The signature is “detached”.
- No certificate is put in the signature. This means that the STS verifying the signature has to know (from its keystore, for example) the public key of the requester, as an evidence of the trust it commits on this requester.
- A canonicalization method shall be used which eliminates namespace declarations that are not visibly used within the SAML token. A suitable algorithm is ”Exclusive XML Canonicalization” which is implemented through a digital signature declaration:

  ```
  <ds:CanonicalizationMethod
  Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
  ```

  Note that the specified canonicalization algorithm omits the comments.

RequestSecurityTokenResponse (RSTR)

The schema for RequestSecurityTokenResponse is illustrated in the following diagram.

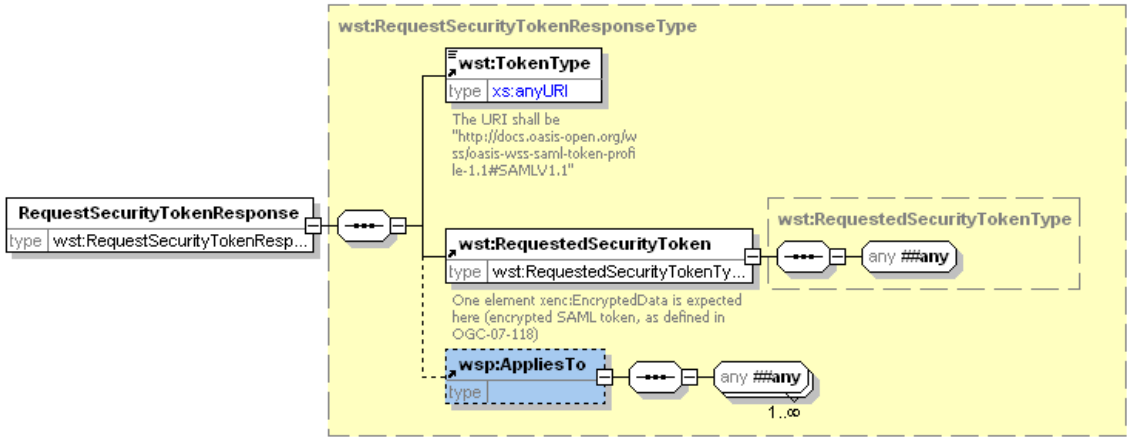


Figure 18: RequestSecurityTokenResponse schema

Refer to WS-Trust 1.3 (section 4.1 in [NR23]), with the following constraints:

wst:RequestSecurityToken/wst:TokenType

is REQUIRED and shall have the following URI, defined in [R11]:
`http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV1.1`
or
`http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV2.0`

wst:RequestSecurityToken/wst:RequestedSecurityToken

is REQUIRED an shall contains one `<xenc:EncryptedData>` element; once decrypted, it shall be a SAML 1.1 assertion as defined in `oasis-sstc-saml-schema-assertion-1.1.xsd` or a SAML 2.0 assertion as defined in `saml-schema-assertion-2.0.xsd` (see below). Specific requirements concerning the encryption and signature of SAML assertion are provided in sections 6.4.1 and 6.4.2, respectively.

WS-Trust Schema

The following schema file defines the types for RST and RSTR.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-trust/200512/"
xmlns:wsa="http://www.w3.org/2005/08/addressing"
xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-
1.0.xsd" xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="http://docs.oasis-
open.org/ws-sx/ws-trust/200512/" elementFormDefault="qualified">
   <xs:import namespace="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-secext-1.0.xsd" schemaLocation="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"/>
   <xs:import namespace="http://schemas.xmlsoap.org/ws/2004/09/policy"
schemaLocation="http://schemas.xmlsoap.org/ws/2004/09/policy/ws-policy.xsd"/>
   <xs:import namespace="http://www.w3.org/2005/08/addressing"
schemaLocation="http://www.w3.org/2006/03/addressing/ws-addr.xsd"/>
   <!-- WS-Trust Section 3.1 -->
   <xs:element name="RequestSecurityToken" type="wst:RequestSecurityTokenType">
      <xs:annotation>
         <xs:documentation>Under wsse:UsernameToken, the element Username may be
followed by an optional wsse:Password element (see OASIS Web Services Security
UsernameToken Profile 1.1)</xs:documentation>
      </xs:annotation>
   </xs:element>
   <xs:complexType name="RequestSecurityTokenType">
      <xs:sequence>
         <xs:element ref="wst:TokenType"/>
         <xs:element ref="wst:RequestType"/>
         <xs:element ref="wsp:AppliesTo" minOccurs="0"/>
         <xs:element ref="wst:DelegateTo" minOccurs="0"/>
         <xs:element ref="wsse:UsernameToken"/>
      </xs:sequence>
      <xs:attribute name="Context" type="xs:anyURI" use="optional"/>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
   </xs:complexType>
   <xs:element name="TokenType">
      <xs:annotation>
         <xs:documentation>The URI shall be "http://docs.oasis-open.org/wss/oasis-wss-
saml-token-profile-1.1#SAMLV1.1" or "http://docs.oasis-open.org/wss/oasis-wss-saml-
token-profile-1.1#SAMLV2.0"</xs:documentation>
      </xs:annotation>
      <xs:simpleType>
         <xs:restriction base="xs:anyURI">
            <xs:enumeration value="http://docs.oasis-open.org/wss/oasis-wss-saml-
token-profile-1.1#SAMLV1.1"/>
            <xs:enumeration value="http://docs.oasis-open.org/wss/oasis-wss-saml-
token-profile-1.1#SAMLV2.0"/>
         </xs:restriction>
      </xs:simpleType>
   </xs:element>
   <xs:element name="RequestType" type="wst:RequestTypeOpenEnum">
      <xs:annotation>
         <xs:documentation>The URI shall be "http://docs.oasis-open.org/ws-sx/ws-
trust/200512/Issue"</xs:documentation>
      </xs:annotation>
   </xs:element>
   <xs:simpleType name="RequestTypeOpenEnum">
      <xs:union memberTypes="wst:RequestTypeEnum xs:anyURI"/>
   </xs:simpleType>
   <xs:simpleType name="RequestTypeEnum">
      <xs:restriction base="xs:anyURI">
         <xs:enumeration value="http://docs.oasis-open.org/ws-sx/ws-
trust/200512/Issue"/>
      </xs:restriction>
   </xs:simpleType>
   <!-- WS-Trust Section 3.2 -->
   <xs:element name="RequestSecurityTokenResponse"
type="wst:RequestSecurityTokenResponseType"/>
   <xs:complexType name="RequestSecurityTokenResponseType">
      <xs:sequence>
```

```
         <xs:element ref="wst:TokenType"/>
         <xs:element ref="wst:RequestedSecurityToken"/>
         <xs:element ref="wsp:AppliesTo" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute name="Context" type="xs:anyURI" use="optional"/>
      <xs:anyAttribute namespace="##other" processContents="lax"/>
   </xs:complexType>
   <xs:element name="RequestedSecurityToken" type="wst:RequestedSecurityTokenType">
      <xs:annotation>
         <xs:documentation>One element xenc:EncryptedData is expected here (encrypted
SAML token, as defined in OGC-07-118)</xs:documentation>
      </xs:annotation>
   </xs:element>
   <xs:complexType name="RequestedSecurityTokenType">
      <xs:sequence>
         <xs:any namespace="##any" processContents="lax"/>
      </xs:sequence>
   </xs:complexType>
   <!-- WS-Trust Section 9.3 -->
   <xs:element name="DelegateTo" type="wst:DelegateToType"/>
   <xs:complexType name="DelegateToType">
      <xs:sequence>
         <xs:any namespace="##any" processContents="lax"/>
      </xs:sequence>
   </xs:complexType>
</xs:schema>
```

SAML Assertion Schema

The schema for SAML assertions 1.1 is defined at the following URL:

http://www.oasis-open.org/committees/download.php/3408/oasis-sstc-saml-schema-assertion-1.1.xsd

The schema for SAML assertions 2.0 is defined at the following URL:

http://docs.oasis-open.org/security/saml/v2.0/saml-schema-assertion-2.0.xsd

WS-Security Schema

Each SOAP service request may include, if required, the encrypted SAML token returned in the RSTR. In such situation, the SOAP header shall contain a `<wsse:Security>` element (WS-Security 1.1) having a `<xenc:EncryptedData>` (the SAML token) as child.

The schema defining the `<wsse:Security>` element is defined at the following URL:

http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd

Annex C: SOAP 1.1 Implementation (Normative)

The normative SOAP protocol binding is SOAP 1.2 (see section 6.2). The support of SOAP 1.1 is optional.

If SOAP 1.1 is used, only SOAP messaging (via HTTP/POST) with document/literal style shall be used. Also, for the RST request, the expected SOAP action is:

`http://docs.oasis-open.org/ws-sx/ws-trust/200512#RequestSecurityToken`

The following information (non-normative) explains the main differences between the above two versions of SOAP and is illustrated in the following figures.

At HTTP header level, SOAP 1.1 uses a Content-Type of text/xml and requires a SOAPAction, whereas SOAP 1.2 uses a Content-Type of application/soap+xml and has no SOAPAction. In SOAP 1.2, an optional action parameter could be added to the Content-Type header field.

At SOAP level, SOAP 1.1 and SOAP 1.2 use a different namespace and a different Fault structure.

```
POST http://aaa.bbb.ccc/csw HTTP/1.1
Host: aaa.bbb.ccc
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://www.opengis.net/cat/csw/2.0.2/requests#GetRecords"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 <soap:Body>
   <GetRecords xmlns="http://www.opengis.net/cat/csw/2.0.2" … >
     <Query> … </Query>
   </GetRecords>
 </soap:Body>
</soap:Envelope>
```

SOAP 1.1

```
POST http://aaa.bbb.ccc/csw HTTP/1.1
Host: aaa.bbb.ccc
Content-Type: application/soap+xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
 <soap12:Body>
   <GetRecords xmlns="http://www.opengis.net/cat/csw/2.0.2" … >
     <Query> … </Query>
   </GetRecords>
 </soap12:Body>
</soap12:Envelope>
```

SOAP 1.2

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:wst="http://schemas.xmlsoap.org/ws/2005/02/trust">
   <soapenv:Body>
      <soapenv:Fault>
         <faultcode>wst:FailedAuthentication</faultcode>
         <faultstring>Authentication failed: invalid password</faultstring>
      </soapenv:Fault>
   </soapenv:Body>
</soapenv:Envelope>
```

SOAP 1.1

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope"
xmlns:wst="http://schemas.xmlsoap.org/ws/2005/02/trust">
   <soapenv:Body>
      <soapenv:Fault>
         <soapenv:Code>
            <soapenv:Value>env:Sender</soapenv:value>
            <soapenv:Subcode>
               <soapenv:value>wst:FailedAuthentication</soapenv:value>
            </soapenv:Subcode>
         </soapenv:Code>
         <soapenv:Reason>
            <soapenv:Text xml:lang="en">Authentication failed: invalid
password</soapenv:Text>
         </soapenv:Reason>
      </soapenv:Fault>
   </soapenv:Body>
</soapenv:Envelope>
```

SOAP 1.2

Annex D: Example of SAML Token Attributes (Non-Normative)

In the GMES Space Component Data Access (GSCDA) community, the following attributes are included in the SAML token to implement basic policy steps:

SAML Token attribute name	Description
Id	Unambiguous federated identity
c	Country of origin
o	Organisation
ProjectName	Names of projects with which user is affiliated
Account	The account number
ServiceName	Associated services
UserProfile	Type of user (Commercial/GMES/Scientific)

Table 2: Example of Attributes in SAML Token

Annex E: XACML Examples (Non-Normative)

The following examples of PDP decision requests and policies are based on XACML 2.0.

Uses Case: restrict access for time period

Decision request:

```
<?xml version="1.0" encoding="UTF-8"?>
<Request xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="
urn:oasis:names:tc:xacml:2.0:context:schema:os http://docs.oasis-
open.org/xacml/access_control-xacml-2.0-context-schema-os.xsd">
   <Subject>
      <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
DataType="xs:string">
         <AttributeValue>anonymous</AttributeValue>
      </Attribute>
   </Subject>
   <Resource>
      <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
DataType="xs:string">
         <AttributeValue>WEB_Map_Server</AttributeValue>
      </Attribute>
   </Resource>
   <Action>
      <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
DataType="xs:string">
         <AttributeValue>GetMap</AttributeValue>
      </Attribute>
   </Action>
   <Environment/>
</Request>
```

Policy:

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os" xmlns:xacml-
context="urn:oasis:names:tc:xacml:2.0:context:schema:os"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="
urn:oasis:names:tc:xacml:2.0:policy:schema:os http://docs.oasis-
open.org/xacml/access_control-xacml-2.0-policy-schema-os.xsd"
PolicyId="urn:oasis:names:tc:xacml:2.0:example:policyid:HL-IDM-480"
RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-
overrides">
   <PolicyDefaults>
      <XPathVersion>http://www.w3.org/TR/1999/Rec-xpath-19991116</XPathVersion>
   </PolicyDefaults>
   <Target>
      <Resources>
         <Resource>
            <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
equal">
               <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">WEB_Map_Server</AttributeValue>
               <ResourceAttributeDesignator
AttributeId="urn:oasis:names:tc:xacml:2.0:resource:resource-id"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
            </ResourceMatch>
         </Resource>
      </Resources>
   </Target>
   <Rule RuleId="urn:oasis:names:tc:xacml:2.0:example:ruleid:HL-IDM-480"
Effect="Deny">
      <Description>
User cannot access the service for getting maps in the time range 9:00 AM - 12:00 AM
      </Description>
      <Target>
```

```
         <Actions>
            <Action>
               <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
equal">
                  <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">GetMap</AttributeValue>
                  <ActionAttributeDesignator
AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
               </ActionMatch>
            </Action>
         </Actions>
      </Target>
      <Condition>
         <Apply FunctionId="urn:oasis:names:tc:xacml:2.0:function:time-in-range">
            <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-one-and-
only">
               <EnvironmentAttributeDesignator
AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-time"
DataType="http:www.w3.org/2001/XMLSchema#time"/>
            </Apply>
            <AttributeValue
DataType="http:www.w3.org/2001/XMLSchema#time">09:00:00</AttributeValue>
            <AttributeValue
DataType="http:www.w3.org/2001/XMLSchema#time">12:00:00</AttributeValue>
         </Apply>
      </Condition>
   </Rule>
   <Rule RuleId="urn:oasis:names:tc:xacml:2.0:example:ruleid:HL-IDM-480-OTHER"
Effect="Permit"/>
</Policy>
```

Uses Case: enforce rules for specific group of users

Decision request:

```
<?xml version="1.0" encoding="UTF-8"?>
<Request xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="
urn:oasis:names:tc:xacml:2.0:context:schema:os http://docs.oasis-
open.org/xacml/access_control-xacml-2.0-context-schema-os.xsd">
   <Subject>
      <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
DataType="xs:string">
         <AttributeValue>dail_user_1</AttributeValue>
      </Attribute>
      <Attribute AttributeId="urn:ogc:um:eop:0.0.4:saml:role" DataType="xs:string">
         <AttributeValue>guest</AttributeValue>
      </Attribute>
   </Subject>
   <Resource>
      <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
DataType="xs:string">
         <AttributeValue>csw-ebrim_catalogue</AttributeValue>
      </Attribute>
   </Resource>
   <Action>
      <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
DataType="xs:string">
         <AttributeValue>GetRecords</AttributeValue>
      </Attribute>
   </Action>
   <Environment/>
</Request>
```

Policy:

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os" xmlns:xacml-
context="urn:oasis:names:tc:xacml:2.0:context:schema:os"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="
urn:oasis:names:tc:xacml:2.0:policy:schema:os http://docs.oasis-
```

```
open.org/xacml/access_control-xacml-2.0-policy-schema-os.xsd"
PolicyId="urn:oasis:names:tc:xacml:2.0:example:policyid:HL-IDM-490"
RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-
overrides">
   <PolicyDefaults>
      <XPathVersion>http://www.w3.org/TR/1999/Rec-xpath-19991116</XPathVersion>
   </PolicyDefaults>
   <Target>
      <Resources>
         <Resource>
            <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
equal">
               <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">ebrim_catalogue
               </AttributeValue>
               <ResourceAttributeDesignator
AttributeId="urn:oasis:names:tc:xacml:2.0:resource:resource-id"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
            </ResourceMatch>
         </Resource>
      </Resources>
   </Target>
   <Rule RuleId="urn:oasis:names:tc:xacml:2.0:example:ruleid:HL-IDM-490"
Effect="Deny">
      <Description>
User with "guest" role cannot access the service in the time range 9:00 AM - 12:00 AM
      </Description>
      <Target>
         <Subjects>
            <Subject>
               <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
equal">
                  <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">guest</AttributeValue>
                  <SubjectAttributeDesignator
AttributeId="urn:ogc:um:eop:0.0.4:saml:role"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
               </SubjectMatch>
            </Subject>
         </Subjects>
      </Target>
      <Condition>
         <Apply FunctionId="urn:oasis:names:tc:xacml:2.0:function:time-in-range">
            <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-one-and-
only">
               <EnvironmentAttributeDesignator
AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-time"
DataType="http:www.w3.org/2001/XMLSchema#time"/>
            </Apply>
            <AttributeValue
DataType="http:www.w3.org/2001/XMLSchema#time">09:00:00</AttributeValue>
            <AttributeValue
DataType="http:www.w3.org/2001/XMLSchema#time">12:00:00</AttributeValue>
         </Apply>
      </Condition>
   </Rule>
   <Rule RuleId="urn:oasis:names:tc:xacml:2.0:example:ruleid:HL-IDM-490-OTHER"
Effect="Permit"/>
</Policy>
```

Uses Case: restrict access to the type of data

Decision request:

```
<?xml version="1.0" encoding="UTF-8"?>
<Request xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="
urn:oasis:names:tc:xacml:2.0:context:schema:os http://docs.oasis-
open.org/xacml/access_control-xacml-2.0-context-schema-os.xsd">
   <Subject>
      <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
DataType="xs:string">
```

```
         <AttributeValue>dail_user_1</AttributeValue>
      </Attribute>
      <Attribute AttributeId="urn:ogc:um:eop:0.0.4:saml:role" DataType="xs:string">
         <AttributeValue>guest</AttributeValue>
      </Attribute>
   </Subject>
   <Resource>
      <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
DataType="xs:string">
         <AttributeValue>csw-ebrim_catalogue</AttributeValue>
      </Attribute>
   </Resource>
   <Action>
      <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
DataType="xs:string">
         <AttributeValue>GetRecords</AttributeValue>
      </Attribute>
   </Action>
   <Environment/>
</Request>
```

Policy:

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os" xmlns:xacml-
context="urn:oasis:names:tc:xacml:2.0:context:schema:os"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="
urn:oasis:names:tc:xacml:2.0:policy:schema:os http://docs.oasis-
open.org/xacml/access_control-xacml-2.0-policy-schema-os.xsd"
PolicyId="urn:oasis:names:tc:xacml:2.0:example:policyid:HL-IDM-500"
RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-
overrides">
   <PolicyDefaults>
      <XPathVersion>http://www.w3.org/TR/1999/Rec-xpath-19991116</XPathVersion>
   </PolicyDefaults>
   <Target>
      <Resources>
         <Resource>
            <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
equal">
               <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">ebrim_catalogue
               </AttributeValue>
               <ResourceAttributeDesignator
AttributeId="urn:oasis:names:tc:xacml:2.0:resource:resource-id"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
            </ResourceMatch>
         </Resource>
      </Resources>
   </Target>
   <Rule RuleId="urn:oasis:names:tc:xacml:2.0:example:ruleid:HL-IDM-500"
Effect="Deny">
      <Description>
User with the "guest" role cannot access high-resolution data
      </Description>
      <Target>
         <Subjects>
            <Subject>
               <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
equal">
                  <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">guest</AttributeValue>
                  <SubjectAttributeDesignator
AttributeId="urn:ogc:um:eop:0.0.4:saml:role"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
               </SubjectMatch>
            </Subject>
         </Subjects>
      </Target>
      <Condition>
         <Apply FunctionId="urn:oasis:names:tc:xacml:2.0:function:double-greater-
than">
            <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:double-one-and-
only">
```

```
            <ResourceAttributeDesignator
DataType="http://www.w3.org/2001/XMLSchema#double" AttributeId="urn:ogc:def:ebRIM-
Slot:OGC-06-131:sensorResolution"/>
            </Apply>
            <AttributeValue
DataType="http:www.w3.org/2001/XMLSchema#double">_threshold
            </AttributeValue>
         </Apply>
      </Condition>
   </Rule>
   <Rule RuleId="urn:oasis:names:tc:xacml:2.0:example:ruleid:HL-IDM-500-OTHER"
Effect="Permit"/>
</Policy>
```

Uses Case: restricting access to users from certain geographic locations

Decision request:

```
<?xml version="1.0" encoding="UTF-8"?>
<Request xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="
urn:oasis:names:tc:xacml:2.0:context:schema:os http://docs.oasis-
open.org/xacml/access_control-xacml-2.0-context-schema-os.xsd">
   <Subject>
      <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
DataType="xs:string">
         <AttributeValue>dail_user_1</AttributeValue>
      </Attribute>
      <Attribute AttributeId="urn:ogc:um:eop:0.0.4:saml:country" DataType="xs:string">
         <AttributeValue>France</AttributeValue>
      </Attribute>
   </Subject>
   <Resource>
      <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
DataType="xs:string">
         <AttributeValue>csw-ebrim_catalogue</AttributeValue>
      </Attribute>
   </Resource>
   <Action>
      <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
DataType="xs:string">
         <AttributeValue>GetRecords</AttributeValue>
      </Attribute>
   </Action>
   <Environment/>
</Request>
```

Policy:

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os" xmlns:xacml-
context="urn:oasis:names:tc:xacml:2.0:context:schema:os"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="
urn:oasis:names:tc:xacml:2.0:policy:schema:os http://docs.oasis-
open.org/xacml/access_control-xacml-2.0-policy-schema-os.xsd"
PolicyId="urn:oasis:names:tc:xacml:2.0:example:policyid:HL-IDM-550"
RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-
overrides">
   <PolicyDefaults>
      <XPathVersion>http://www.w3.org/TR/1999/Rec-xpath-19991116</XPathVersion>
   </PolicyDefaults>
   <Target>
      <Resources>
         <Resource>
            <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
equal">
               <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">ebrim_catalogue
               </AttributeValue>
```

```
               <ResourceAttributeDesignator
AttributeId="urn:oasis:names:tc:xacml:2.0:resource:resource-id"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
            </ResourceMatch>
         </Resource>
      </Resources>
   </Target>
   <Rule RuleId="urn:oasis:names:tc:xacml:2.0:example:ruleid:HL-IDM-550"
Effect="Deny">
      <Description>
User from the "France" country cannot access the service
      </Description>
      <Target>
         <Subjects>
            <Subject>
               <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
equal">
                  <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">France</AttributeValue>
                  <SubjectAttributeDesignator
AttributeId="urn:ogc:um:eop:0.0.4:saml:country"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
               </SubjectMatch>
            </Subject>
         </Subjects>
      </Target>
   </Rule>
   <Rule RuleId="urn:oasis:names:tc:xacml:2.0:example:ruleid:HL-IDM-550-OTHER"
Effect="Permit"/>
</Policy>
```

Annex F: Example of WSDL using WS-Policy (Non-Normative)

The following WSDL example illustrates the use of WS-Policy, WS-PolicyAttachment, and WS-SecurityPolicy, where the policy is applied on a per-operation basis.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:tns="http://www.opengis.net/cat/csw/2.0.2/soap"
xmlns:csw="http://www.opengis.net/cat/csw/2.0.2"
xmlns:ows="http://www.opengis.net/ows"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:wsp="http://www.w3.org/ns/ws-policy"
xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702"
xmlns:wsa=" http://www.w3.org/2005/08/addressing"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
1.0.xsd" targetNamespace="http://www.opengis.net/cat/csw/2.0.2/soap">

   <wsdl:types>
      <xsd:schema elementFormDefault="qualified"
targetNamespace="http://www.opengis.net/cat/csw/2.0.2" version="2.0.2">
         <xsd:include schemaLocation="http://schemas.opengis.net/csw/2.0.2/CSW-
discovery.xsd"/>
         <xsd:include schemaLocation="http://schemas.opengis.net/csw/2.0.2/CSW-
publication.xsd"/>
         <xsd:import namespace="http://www.opengis.net/ows"
schemaLocation="http://schemas.opengis.net/ows/1.0.0/owsExceptionReport.xsd"/>
      </xsd:schema>
   </wsdl:types>

   <wsdl:message name="GetCapabilitiesRequest">
      <wsdl:part name="Body" element="csw:GetCapabilities"/>
   </wsdl:message>
   <wsdl:message name="GetCapabilitiesResponse">
      <wsdl:part name="Body" element="csw:Capabilities"/>
   </wsdl:message>

   …

   <wsdl:message name="GetRecordsRequest">
      <wsdl:part name="Body" element="csw:GetRecords"/>
   </wsdl:message>
   <wsdl:message name="GetRecordsResponse">
      <wsdl:part name="Body" element="csw:GetRecordsResponse"/>
   </wsdl:message>

   …

   <wsdl:message name="ServiceExceptionReport">
      <wsdl:part name="Body" element="ows:ExceptionReport"/>
   </wsdl:message>

   <wsdl:portType name="csw">
      <wsdl:operation name="csw.getCapabilities">
         <wsdl:input message="tns:GetCapabilitiesRequest"/>
         <wsdl:output message="tns:GetCapabilitiesResponse"/>
         <wsdl:fault name="ServiceExceptionReport"
message="tns:ServiceExceptionReport"/>
      </wsdl:operation>

   …

      <wsdl:operation name="csw.getRecords">
         <wsdl:input message="tns:GetRecordsRequest"/>
         <wsdl:output message="tns:GetRecordsResponse"/>
         <wsdl:fault name="ServiceExceptionReport"
message="tns:ServiceExceptionReport"/>
      </wsdl:operation>

   …
```

```
   </wsdl:portType>

   <wsdl:binding name="csw-SOAP" type="tns:csw">
      <wsdl:documentation>CSW interface bound to SOAP over HTTP/1.1.
</wsdl:documentation>
      <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
      <wsdl:operation name="csw.getCapabilities">
         <soap:operation
soapAction="http://www.opengis.net/cat/csw/2.0.2/requests#GetCapabilities"/>
         <wsdl:input>
            <soap:body use="literal"/>
         </wsdl:input>
         <wsdl:output>
            <soap:body use="literal"/>
         </wsdl:output>
         <wsdl:fault name="ServiceExceptionReport">
            <soap:fault use="literal" name="ServiceExceptionReport"/>
         </wsdl:fault>
      </wsdl:operation>

   …

      <wsdl:operation name="csw.getRecords">
         <soap:operation
soapAction="http://www.opengis.net/cat/csw/2.0.2/requests#GetRecords"/>
         <wsdl:input>
            <soap:body use="literal"/>
         </wsdl:input>
         <wsdl:output>
            <soap:body use="literal"/>
         </wsdl:output>
         <wsdl:fault name="ServiceExceptionReport">
            <soap:fault use="literal" name="ServiceExceptionReport"/>
         </wsdl:fault>
         <wsp:PolicyReference URI="#HMApolicy" wsdl:required="true"/>
      </wsdl:operation>

   …

   </wsdl:binding>

   <wsdl:service name="EbrimCatalogServiceSoap">
      <wsdl:port name="Ebrim-Discovery-PortSoap" binding="tns:csw-SOAP">
         <soap:address location="http://varchive.eumetsat.org/axis2/services/csw202-
wrs-eoproduct"/>
      </wsdl:port>
   </wsdl:service>

   <wsp:Policy wsu:Id="HMApolicy">
      <wsp:All>
         <sp:AsymmetricBinding>
            <wsp:Policy>
               <sp:InitiatorToken>
                  <wsp:Policy>
                     <sp:X509Token sp:IncludeToken="http://docs.oasis-open.org/ws-
sx/ws-securitypolicy/200702/IncludeToken/Never">
                        <wsp:Policy>
                           <sp:RequireThumbprintReference/>
                           <sp:WssX509V3Token10/>
                        </wsp:Policy>
                     </sp:X509Token>
                  </wsp:Policy>
               </sp:InitiatorToken>
               <sp:RecipientToken>
                  <wsp:Policy>
                     <sp:X509Token sp:IncludeToken="http://docs.oasis-open.org/ws-
sx/ws-securitypolicy/200702/IncludeToken/Never">
                        <wsp:Policy>
                           <sp:RequireThumbprintReference/>
                           <sp:WssX509V3Token10/>
                        </wsp:Policy>
                     </sp:X509Token>
                  </wsp:Policy>
               </sp:RecipientToken>
               <sp:AlgorithmSuite>
```

```
                  <wsp:Policy>
                     <sp:Basic128Rsa15/>
                  </wsp:Policy>
               </sp:AlgorithmSuite>
               <sp:Layout>
                  <wsp:Policy>
                     <sp:Strict/>
                  </wsp:Policy>
               </sp:Layout>
            </wsp:Policy>
         </sp:AsymmetricBinding>
         <sp:EncryptedSupportingTokens>
            <wsp:Policy>
               <sp:SamlToken sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-
securitypolicy/200702/IncludeToken/AlwaysToRecipient">
                  <sp:Issuer>
                     <wsa:Address>
                        urn:ceos:def:epr:esa-cds:1.0:sts-ope
                     </wsa:Address>
                     <!– or simply
                     <wsa:Address/>
                         to accept SAML tokens from different issuers (IdPs) -->
                  </sp:Issuer>
                  <sp:WssSamlV11Token11/>
               </sp:SamlToken>
            </wsp:Policy>
         </sp:EncryptedSupportingTokens>
      </wsp:All>
   </wsp:Policy>

</wsdl:definitions>
```

Annex G: Revision history

Date	Version	Editor	Sections modified	Description
15 Sep 2007	0.0.1 Draft	R.Smillie	All	Initialised Draft Document.
23 Apr 2008	0.0.2	R.Smillie		Updated in line with EO DAIL implementation project
07 Feb 2009	0.0.3	R.Smillie		Updated in line with EO DAIL implementation project SOAP version changed to 1.1 Authentication request does not use WS-Security Message examples added Encryption and signature descriptions improved
30 Jun 2009	0.0.4	R.Smillie		Updated in line with EO DAIL RID PRE-AR2#34: • Namespace in encrypted message example corrected to http://earth.esa.int/um/eop/saml <http://earth.esa.int/um/eop/saml> • decryptandCheckSignature removed from STS • authenticating identity correctly asserted in examples • authenticate and authenticateFederated merged into one operation • Attribute assertions updated in examples • WSDL provided for STS • Clarification made for the assertion element and schema attached • All schemas and references given in annex.

30 Oct 2009	0.0.5	P. Denis	All	Updates following RIDS of EUMETSAT/con terra, analysis by FP-7 GENESIS and HMA-T projects • Removed references to DAIL and GS, for the sake of generality • Added conformance (chapter 2) and Abstract Test Suite (annex A) • Put SOAP 1.2 as baseline protocol binding and put SOAP 1.1 support in annex C • Demote SAML token attributes specification as an example (annex D) • Added authorisation use cases (chapter 10) and XACML examples (annex E) • Added WS-Policy recommendation • Removed non-standard Assertion element • Remove certificate from SAML token • Changed protocol for authentication through external IdP • Added "future work" section • Clarify roles of each entity in the system • Added threats / countermeasures analysis (chapter 9) • Misc corrections and clarifications
29 Jan 2010	0.0.6	P. Denis	All	• Alignment on OASIS WS-Trust 1.3, i.e. Security Token Service (STS), providing Request Security Token operation (RST) • Precisions and changes on signature of security tokens • Created annex G for ESA UM-SSO / EO-DAIL Integration • Misc corrections and clarifications

5 Mar 2010	0.1.0	P.Denis	All	Updates following RIDS of ESA, EUMETSAT/con terra, misc corrections and precisions • Removal of references to LDAP, for the sake of generality • Improved description of encryption protocol ("hybrid cryptosystem"). • Ability to have multiple relying parties (hence multiple Federating Entities) through "AppliesTo" element described in "Extension Points" section. • Removal of sections about too general topics on SAML, encryption and signature. • Added test module for RST with signature • Correction on "Federated IdP - external identification" use case and associated RST schema: it shall use "DelegateTo" element of WS-Trust, instead of "AppliesTo" element. • Reference to GeoXACML • Misc corrections and clarifications • Corrections of typos and wrong section numbering

5 Jul 2010	0.3.0	P.Denis		"Friendly amendments" following OGC TC 9th March 2010: • Described the re-use mechanism for secured tokens • Detail Single-Sign-On • Removed PEP in front of IdP • Made explicit that an external IdP can be based on Shibboleth • Defined the scope of the document "as specific as possible" Updates following RIDS of ESA, EUMETSAT/con terra, misc corrections and precisions • Generalisation to architectures without a Federating Entity; clarification of roles of STS and Relying Parties; unification of use cases 1 and 3. • Description of the general mechanism used by STS to get the encryption public key. • Clarification on the meaning of the "Client" actor in use cases • Replaced "authorisation request" by "service request" for uniformity • Annex G promoted as section 8 and updated to be more general (Web Portal / Web Service Broker Integration) • Updated document type as "Candidate Best Practice" (Carl Reed) • Precisions made on the scope of the document (P.G. Marchetti)
8 Jul 2010	0.3.1	P.Denis		Update following HMA AWG Meeting: • Put "Shibboleth" as an example of Web-SSO (not more)

3 Sep 2010	0.3.2	P.Denis		Updates following issues found in activities of integration based on the present best practice: • Precisions made on XML canonicalization method for signature of SAML tokens and corrections of XML examples • Precisions made on signature reference in SAML tokens and corrections of XML examples • In figure 6, step 10 corrected to be in line with the text that follows • New options allowed for asynchronous service responses
3 Sep 2010	1.0	P.Denis	All	Promotion to “Best Practice” document
10 May 2013	1.1 Draft	P.Denis P. Jacques	All	Misc editorial corrections and clarifications. Convergence towards new OGC template. Added HTTP protocol binding, for both authentication requests and service requests. Split “STS with trusted IdP” use case into two cases, for the sake of uniformity on STS delegation. Updates following issues / recommendations given by con terra on HMA Forum: • Correction on type of WS-Trust “DelegateTo” element • Mechanism for identification of delegate STS, based on WS-Trust “DelegateTo” element • Added missing WS-Trust “DelegateTo” in the ws-trust.xsd schema • Clarification on the concept of “STS with trusted IdP” • Removal of obsolete text in 7.1.3 General improvement of document structure and figures for enhanced clarity and readability. Alignment of all examples with SOAP 1.2. Alignment with OWS Common for exceptions on Service requests.

21 May 2013	1.1 Draft	P. Jacques	All	Added example of RST with AppliesTo in section 6.4.1.1. Added footnote in section 6.4.2 to make recommendations about the use of the Issuer attribute/element of the SAML token. Modified URN in RST example in section 6.4.3.2. Updated Figure 7 and Figure 8 to indicate that the RTS signature is verified with the Client's public key. Updated footnote on user id in section 6.4.3.3 to take into account multiple Web-SSO security domains. Added WS-PolicyAttachment and WS-SecurityPolicy in section 3.1 and section 6.4.4. Corrected SOAP namespace in Figure 11. Modified Issuer attribute value in Figure 12. Changed xmlsignature to b64token in RST authorization HTTP header in section 7.1.2.1. Replaced URI encoding by Base64 encoding. Changed the example accordingly. Added the use of the "403 Forbidden" HTTP status in sections 7.2.1.3 and 7.2.2.3. Allow for non-OGC services provided that the same exception code is used. Removed the need for URI encoding in section 7.2.2.1. Updated/improved the related example. Added note on possible future use of compression and KVP encoding. Updated Figure 15 and Figure 16 to align components and colours. Corrected third paragraph of Annex B with references to schemas. Added non-normative text in Annex C to show the main differences between SOAP 1.1 and SOAP 1.2.
12 Jun 2013	1.1 Draft	P. Jacques	All	Editorial changes made in sections i (Abstract) and iv (Submitting organisations). A footnote was added in section 6.4.1.1 to recommend the value to use for the wsa:Address element of AppliesTo.

28 Jun 2013	1.1 Draft	P. Jacques	All	Changed [NR16] to point to a newer version of the OWS Common specification. Added note in section 6.4.3.1 with assumption on proper password protection by the Client. Updated the value of a few elements in Figure 12. Updated Annex A to match the changes made in sections 6 and 7. Added the wsa namespace prefix in table in Annex B. Added WSDL example in Annex F.