

Open Geospatial Consortium Inc.

Date: 2007-05-07

Reference number of this document: OGC 06-107r1

Version: 0.9

Category: OpenGIS® Interoperability Program Report

Editor: Cristian Opincaru

Trusted Geo Services IPR

Copyright © 2007 Open Geospatial Consortium. All Rights Reserved

To obtain additional rights of use, visit <http://www.opengeospatial.org/legal>

Warning

This document is not an OGC Standard. This document is an OGC Discussion Paper and is therefore not an official position of the OGC membership. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an OGC Standard. Further, an OGC Discussion Paper should not be referenced as required or mandatory technology in procurements.

Document type: OpenGIS® Discussion Paper
Document stage: Draft
Document language: English

License Agreement

Permission is hereby granted by the Open Geospatial Consortium, ("Licensor"), free of charge and subject to the terms set forth below, to any person obtaining a copy of this Intellectual Property and any associated documentation, to deal in the Intellectual Property without restriction (except as set forth below), including without limitation the rights to implement, use, copy, modify, merge, publish, distribute, and/or sublicense copies of the Intellectual Property, and to permit persons to whom the Intellectual Property is furnished to do so, provided that all copyright notices on the intellectual property are retained intact and that each person to whom the Intellectual Property is furnished agrees to the terms of this Agreement.

If you modify the Intellectual Property, all copies of the modified Intellectual Property must include, in addition to the above copyright notice, a notice that the Intellectual Property includes modifications that have not been approved or adopted by LICENSOR.

THIS LICENSE IS A COPYRIGHT LICENSE ONLY, AND DOES NOT CONVEY ANY RIGHTS UNDER ANY PATENTS THAT MAY BE IN FORCE ANYWHERE IN THE WORLD.

THE INTELLECTUAL PROPERTY IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF THIRD PARTY RIGHTS. THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE DO NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE INTELLECTUAL PROPERTY WILL MEET YOUR REQUIREMENTS OR THAT THE OPERATION OF THE INTELLECTUAL PROPERTY WILL BE UNINTERRUPTED OR ERROR FREE. ANY USE OF THE INTELLECTUAL PROPERTY SHALL BE MADE ENTIRELY AT THE USER'S OWN RISK. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR ANY CONTRIBUTOR OF INTELLECTUAL PROPERTY RIGHTS TO THE INTELLECTUAL PROPERTY BE LIABLE FOR ANY CLAIM, OR ANY DIRECT, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM ANY ALLEGED INFRINGEMENT OR ANY LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR UNDER ANY OTHER LEGAL THEORY, ARISING OUT OF OR IN CONNECTION WITH THE IMPLEMENTATION, USE, COMMERCIALIZATION OR PERFORMANCE OF THIS INTELLECTUAL PROPERTY.

This license is effective until terminated. You may terminate it at any time by destroying the Intellectual Property together with all copies in any form. The license will also terminate if you fail to comply with any term or condition of this Agreement. Except as provided in the following sentence, no such termination of this license shall require the termination of any third party end-user sublicense to the Intellectual Property which is in force as of the date of notice of such termination. In addition, should the Intellectual Property, or the operation of the Intellectual Property, infringe, or in LICENSOR's sole opinion be likely to infringe, any patent, copyright, trademark or other right of a third party, you agree that LICENSOR, in its sole discretion, may terminate this license without any compensation or liability to you, your licensees or any other party. You agree upon termination of any kind to destroy or cause to be destroyed the Intellectual Property together with all copies in any form, whether held by you or by any third party.

Except as contained in this notice, the name of LICENSOR or of any other holder of a copyright in all or part of the Intellectual Property shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Intellectual Property without prior written authorization of LICENSOR or such copyright holder. LICENSOR is and shall at all times be the sole entity that may authorize you or any third party to use certification marks, trademarks or other special designations to indicate compliance with any LICENSOR standards or specifications.

This Agreement is governed by the laws of the Commonwealth of Massachusetts. The application to this Agreement of the United Nations Convention on Contracts for the International Sale of Goods is hereby expressly excluded. In the event any provision of this Agreement shall be deemed unenforceable, void or invalid, such provision shall be modified so as to make it valid and enforceable, and as so modified the entire Agreement shall remain in full force and effect. No decision, action or inaction by LICENSOR shall be construed to be a waiver of any rights or remedies available to it.

None of the Intellectual Property or underlying information or technology may be downloaded or otherwise exported or reexported in violation of U.S. export laws and regulations. In addition, you are responsible for complying with any local laws in your jurisdiction which may impact your right to import, export or use the Intellectual Property, and you represent that you have complied with any regulations or registration procedures required by applicable law to make this license enforceable

Contents		Page
1	Scope.....	9
1.1	Goals and Non-Goals	9
1.2	Relation with other documents developed during the OWS4.GeoDRM initiative.....	10
2	Compliance	10
3	Normative references	10
4	Terms and definitions	11
5	Conventions	13
5.1	Abbreviated terms	13
5.2	UML notation.....	13
5.3	Used parts of other documents	13
6	OWS4 and the GeoDRM thread of OWS4	14
6.1	Introduction	14
6.2	The GeoDRM Thread.....	15
6.3	GeoDRM Use Cases.....	15
6.3.1	Use Case #1: Unrestricted Use License	15
6.3.2	Use Case #2: Distributor License	16
6.3.3	Use Case #3: End User License.....	17
6.3.4	Use Case #4: WFS-T Feature Updater.....	18
7	Trust Model.....	20
7.1	WS-Trust Model.....	20
7.2	Models for Trust Brokering and Assessment	22
7.2.1	Token Acquisition.....	22
7.2.2	Out-of-Band Token Acquisition	22
7.2.3	Trust Bootstrap.....	22
7.3	WS-Trust Security Token Service Framework	23
7.4	Trust model for OpenGIS® Web Services.....	24
7.4.1	Trust Model: The Client Perspective	25
7.4.2	Trust Model: The (GeoDRM Enabled) OpenGIS® Web Service Perspective	27
7.4.2.1	License Tokens.....	29
7.4.2.2	License Token Validation	29
7.4.3	Trust model for cascading scenarios.....	30
7.4.3.1	Different types of service chaining	31
7.4.3.2	The Workflow	32
7.4.3.3	Error handling / Negotiation of Credentials	35
8	Model Implementation.....	36
8.1	Identity tokens	36
8.1.1	Username / Password.....	37
8.1.2	Kerberos.....	37

8.1.3	PKI / X509 Certificates.....	37
8.1.4	SAML	38
8.2	Licenses and License tokens	38
8.2.1	Direct-encoded license tokens	39
8.2.2	Pointer-like license tokens	39
8.3	The License Broker / Manager.....	39
9	Model Implementation in OWS4.....	41
9.1	Identity tokens in OWS4	41
9.1.1	Username / Password Tokens (UniBW)	42
9.1.1.1	Acquiring the token	42
9.1.1.2	Attaching the token to a OWS Request Message.....	42
9.1.2	X509 Certificates (UniBW)	44
9.1.2.1	Acquiring the token	44
9.1.2.2	Attaching the token to a OWS Request Message.....	44
9.1.3	SAML Assertions / The Authentication Service proposed by Fraunhofer.....	46
9.1.3.1	Acquiring and attaching the token to a OWS request	46
9.1.4	Identity Token Encoding.....	47
9.2	License Reference Tokens in OWS4.....	49
9.2.1	License Reference Tokens encoding	49
9.2.2	Acquiring a License Reference Token.....	51
9.2.3	Attaching a License Reference Token to a SOAP Message	51
9.3	Licenses	53
9.3.1	License Model.....	53
9.3.1.1	License encoding.....	55
9.4	The License Broker / Manager in OWS4.....	56
9.4.1	License Broker Service.....	56
9.4.2	License Manager Service:.....	56
10	Future Work.....	59

Figures	Page
Figure 1: The trust model from WS-Trust	21
Figure 2: The functionality of a Security Token Service.....	24
Figure 3: Trust model from the client perspective.....	25
Figure 4: Trust model from the GeoDRM Enabled OpenGIS® perspective.....	27
Figure 5: The verification steps	28
Figure 6: License token types	29
Figure 7: Distribution choices for enforcement.....	30
Figure 8: A typical service chaining scenario.....	31
Figure 9: Tokens & claims in a service chaining scenario	33
Figure 10: Tokens flow in a service chaining scenario.....	35
Figure 11: Authentication sequence.....	47
Figure 12: SAML Assertion.....	48
Figure 13: SAML AuthenticationStatement	48
Figure 14: SAML AttributeStatement	49
Figure 15: SAML Subject.....	49
Figure 16: License as used in OWS-4.....	53
Figure 17: License encoding schema.....	55

i. Preface

This document was developed as part of the OGC OWS-4 GeoDRM Thread activity as part of the OGC Interoperability Program OWS-4 initiative. The OWS4 initiative began June 2006 and finished with a demonstration in early December 2006 at the Port Authority of New York and New Jersey, in Jersey City, USA. The results were presented at the OGC San Diego TC meeting in mid December in the GeoDRM WG, Security WG and in the Architecture WG.

Suggested additions, changes, and comments on this draft report are welcome and encouraged. Such suggestions may be submitted by email message or by making suggested changes in an edited copy of this document.

The changes made in this document version, relative to the previous version, are tracked by Microsoft Word, and can be viewed if desired. If you choose to submit suggested changes by editing this document, please first accept all the current changes, and then make your suggested changes with change tracking on.

ii. Document terms and definitions

This document uses the specification terms defined in Subclause 5.3 of [OGC 05-008], which is based on the ISO/IEC Directives, Part 2. Rules for the structure and drafting of International Standards. In particular, the word “shall” (not “must”) is the verb form used to indicate a requirement to be strictly followed to conform to this specification.

iii. Submitting organizations

The following organizations submitted this document to the Open Geospatial Consortium Inc.

- con terra GmbH, Münster, Germany
- Fraunhofer ISST, Dortmund, Germany
- University of the German Armed Forces (UniBW), Munich, Germany

iv. Document contributor contact points

All questions regarding this document should be directed to the editor or the contributors:

Name	Organization	Contact
Cristian OPINCARU (CO)	University of the German Armed Forces (UniBW)	Cristian.Opincaru [at] unibw.de
Andreas MATHEUS (AM)	University of the German Armed Forces (UniBW)	Andreas.Matheus [at] unibw.de
Christian Elfers (CE)	con terra GmbH	christian.elfers [at] conterra.de
Jan Martin Senne (JMS)	Fraunhofer ISST	jan.martin.senne [at] do.isst.fraunhofer.de
Roland Wagner (RW)	con terra GmbH	roland.wagner [at] uni-muenster.de

v. Revision history

Date	Release	Editor	Primary clauses modified	Description
17.07.06	002	Opincaru	All	Initial document creation
10.01.07	004	Opincaru	All	Added chapter on OWS4, Implementation and Implementation in OWS4
12.01.07	005	Opincaru	8	Initial section content. Minor changes to the rest of the document.
15.01.07	006	Opincaru	3, 9, 10	Initial section content. Minor changes to the rest of the document.
19.01.07	007	Opincaru	9	Incorporated feedback from RW, AM, JMS & CE
May 3 2007	009	Carl Reed	Various	Prepare document for posting as DP

vi. Changes to the OGC Abstract Specification

Because this report is indented for a release as OGC Discussion paper, no Change Requests are requested currently.

However portions of this document complement the GeoDRM Reference Model (OGC #06-004r4). These should be seen as extensions to the reference model and should be taken into consideration in future reviews of this document.

vii. Future work

The future work is described in detail in chapter 10.

Foreword

The contents of this document are the result of a multi-organization collaboration during the OGC Web Services 4.0 Test Bed, GeoDRM thread. The document describes a trust model that enables the exchange of trusted messages between OGC services and clients.

The document is intended as an OGC Discussion Paper. It does neither cancel nor replace other OGC documents in whole or in part.

Two other documents are also planned as result of OWS4.GeoDRM: The GeoDRM Engineering Viewpoint and Supporting Architecture in OWS4 (OGC #06-182) and the OWS Common Change Proposal (OGC #06-177). The intention of this document is to complement the other two documents.

The model presented in this document is based on the model described in WS-Trust. Other parts of this document, especially the implementation reference other security standards & specifications from IETF, W3C, OASIS and ISO. Most notably, the following standards are referenced: WS-Security, WS-Trust, XACML, SAML.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The OGC 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 specification set forth in this document, and to provide supporting documentation.

OpenGIS[®] Trusted Geo Services IPR

1 Scope

1.1 Goals and Non-Goals

The goal of this document is to enable the exchange of trusted messages between OpenGIS[®] Web Services and clients for these services. As such, this document will describe a trust model that is based on the exchange and brokering of security tokens. This model is based on the model proposed by the WS-Trust specification. This document will address the service protocol, service request, chaining with other services and service response required to form a complete trusted services chain.

The following are goals of this document:

- The definition of trust and trust relations;
- To propose a trust model to allow trusted message exchanges between OpenGIS[®] Web Services and clients;
- To specify what the following entities and the interactions between them are: GeoDRM-enabled Client, GeoDRM-enabled Service, License Manager, License Broker and Identity Provider;
- To give indication how the model can be implemented using existing mainstream IT standards created by OASIS, IETF, W3C and ISO;
- To document how the model was implemented during the OWS4 initiative;
- To provide feedback for upcoming OGC testbeds (such as OWS5) and the OGC working groups.

The following are explicit non-goals of this document:

- Federation of identities
- Provide basic security concepts to the reader – it is assumed that the reader has some knowledge about security and also at least some knowledge about SOAP, WS-Security, XACML and SAML.
- To present in detail the implementations and how these implementations were demonstrated during the demo event in New Jersey. For this, ask the OGC staff for a DVD with the demo and / or check the OWS-4 home on the OGC web site.
- Encryption of licenses

1.2 Relation with other documents developed during the OWS4.GeoDRM initiative

In addition to this document, during the OWS4.GeoDRM initiative the following other two documents were developed:

- GeoDRM Engineering Viewpoint and Supporting Architecture in OWS4 (OGC# 06-184)
- OWS Common Change Proposal (OGC #06-177)

It is the intention of this document to complement these other two documents.

2 Compliance

This report is indented as a discussion paper. Compliance is not an issue to be taken into consideration for this document.

3 Normative references

As this document is not an implementation specification, there are no normative references. However, the following documents are considered to be relevant to this report and useful for the reader.

Other relevant interoperability program reports (IPR) from OWS4:

- [1] OWS4-IPR: GeoDRM OWS4 Common Change Proposal – OGC #06-177
- [2] OWS4-IPR: GeoDRM Engineering Viewpoint and Supporting Architecture in OWS4 – OGC #06-184

Specifications, Discussion Papers and other relevant documents from OGC:

- [3] The GeoDRM Reference Model, OGC #06-004r4
- [4] GeoXACML, OGC #05-036

Standards from OASIS, ISO, W3C and IETF relevant to this report:

- [5] Web Services Trust Language, February 2005, online at <http://specs.xmlsoap.org/ws/2005/02/trust/WS-Trust.pdf>
- [6] OASIS Web Services Security, Version 1.1, February 2006, online at <http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf>
- [7] OASIS Web Services Security, Username Token Profile, Version 1.1, online at: <http://www.oasis-open.org/committees/download.php/16782/wss-v1.1-spec-os-UsernameTokenProfile.pdf>

- [8] OASIS Web Services Security, X.509 Token Profile, Version 1.1, online at: <http://www.oasis-open.org/committees/download.php/16785/wss-v1.1-spec-os-x509TokenProfile.pdf>
- [9] OASIS Web Services Security, X.509 SAML Profile, Version 1.1, online at: <http://www.oasis-open.org/committees/download.php/16768/wss-v1.1-spec-os-SAMLTOKENProfile.pdf>
- [10] OASIS eXtensible Access Markup Language (XACML), Version 2.0, online at: <http://docs.oasis-open.org/xacml/2.0/XACML-2.0-OS-ALL.zip>
- [11] OASIS Security Assertion Markup Language (SAML), Version 1.1, online at: <http://www.oasis-open.org/committees/download.php/3400/oasis-sstc-saml-1.1-pdf-xsd.zip>
- [12] OASIS Security Assertions Markup Language (SAML), Version 2.0, online at: <http://docs.oasis-open.org/security/saml/v2.0/saml-2.0-os.zip>
- [13] W3C Web Services Architecture, February 2004, online at <http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/>

4 Terms and definitions

The purpose of this chapter is to introduce and explain important terms. Please take a moment to read these definitions before reading the rest of the document.

For the purposes of this document the following terms shall apply. For each of the terms a reference is given to the document where the respective term is defined. Most of the terminology is taken from the following sources: *[WS-Trust]*, [Web Services Architecture]

4.1

Claim

A claim is a statement made about a client, service or other resource (e.g. name, identity, key, group, privilege, capability, etc.). *Source: [WS-Trust]*

4.2

License Claim

A claim, typically about a client, referring to rights having been granted for some geospatial resource (e.g. OpenGIS® Web Service, etc.)

4.3

Identity Claim

A claim, typically about a client, referring to some identity attribute (e.g. name, etc.).

4.4

Security Token

A security token represents a collection of claims. *Source: [WS-Trust]*

4.5

Security Token Service

A security token service (STS) is a Web service that issues security tokens (see [WS-Security]). That is, it makes assertions based on evidence that it trusts, to whoever trusts it (or to specific recipients). To communicate trust, a service requires proof, such as a signature to prove knowledge of a security token or set of security tokens. A service itself can generate tokens or it can rely on a separate STS to issue a security token with its own trust statement (note that for some security token formats this can just be a re-issuance or co-signature). This forms the basis of trust brokering. *Source: [WS-Trust]*

4.6

Trust

Trust is the characteristic that one entity is willing to rely upon a second entity to execute a set of actions and/or to make set of assertions about a set of subjects and/or scopes. *Source: [WS-Trust]*

4.7

Direct Trust

Direct trust is when a relying party accepts as true all (or some subset of) the claims in the token sent by the requestor. *Source: [WS-Trust]*

4.8

Direct Brokered Trust

Direct Brokered Trust is when one party trusts a second party who, in turn, trusts or vouches for, a third party. *Source: [WS-Trust]*

4.9

Indirect Brokered Trust

Indirect Brokered Trust is a variation on direct brokered trust where the second party negotiates with the third party, or additional parties, to assess the trust of the third party. *Source: [WS-Trust]*

4.10

Software Agent

A software agent is a software program acting on behalf of a person or organization. *Source: a specialized definition of the one found in [Web Services Architecture]*

4.11

License

Representation of grants that convey to principals the rights to use specified resources subject to specified conditions *Source: [GeoDRM RM]*

5 Conventions

5.1 Abbreviated terms

LB	License Broker
LM	License Manager
STS	Security Token Service

5.2 UML notation

All diagrams that appear in this specification are presented using the Unified Modeling Language (UML) static structure diagram, as described in Subclause 5.2 of [OGC 05-008].

5.3 Used parts of other documents

This document uses significant parts of document [WS-Trust]. To reduce the need to refer to that document, this document copies some of those parts with small modifications. To indicate those parts to readers of this document, the largely copied parts are shown with a light grey background (15%).

6 OWS4 and the GeoDRM thread of OWS4

The purpose of this chapter is to introduce the reader to the activities form OWS4 and especially to the GeoDRM thread of this initiative. Parts of this chapter can also be found in [OWS4EngVP].

6.1 Introduction

OGC's Interoperability Program is a global, hands-on and collaborative prototyping program designed to rapidly develop, test and deliver proven candidate specifications into OGC's Specification Program, where they are formalized for public release. In OGC's Interoperability Initiatives, an international team of technology providers' work together to solve specific geo-processing interoperability problems posed by the initiative's sponsoring organizations. OGC Interoperability Initiatives include test beds, pilot projects, interoperability experiments, and interoperability support services - all designed to encourage rapid development, testing, validation and adoption of open, consensus based standards specifications.

In the fall of 2005, the OGC issued a call for sponsors for an OGC OWS-4 Interoperability initiative testbed activity to advance OGC's open framework for interoperability in the geospatial industry. Three meetings were conducted with potential OWS-4 sponsors to review the OGC technical baseline, to discuss OWS 3 results, and to identify OWS 4 requirements. Sponsors have expressed keen interest in advancing standards for sensor webs, geospatial digital rights management, geospatial semantics and knowledge management. After analyzing the sponsors input, the OGC Interoperability Team recommended to the sponsors that the content of the OWS-4 initiative be organized around the following 7 threads:

- Sensor Web Enablement (SWE)
- Geo Processing Workflow (GPW)
- Geo Decision Support (GeoDSS)
- Geo-Digital Rights Management (GeoDRM)
- CAD / GIS / BIM
- OGC Location Services (OpenLS)
- Compliance Testing (CITE)

The following companies sponsored this initiative: NGA, NTA, GeoConnections, NASA, ORNL, LMCO, BAE, GSA, Ordnance Survey, NATO C3, TeleAtlas.

6.2 The GeoDRM Thread

GeoDRM or Geospatial Digital Rights Management is defined in the OGC GeoDRM Reference Model as the packaging, distributing, controlling and tracking of geospatial content based on rights and licensing information. More generally, it can be taken to cover a broad spectrum of capabilities and underlying technologies supporting description, identification, trading, protecting monitoring and tracking of all forms of rights usages for both tangible and intangible (electronic) assets, including the management of rights-holders relationships. For the purpose of the OWS-4 initiative, GeoDRM consists of standards, technologies, and practices which enable interoperable trading of geospatial content to be implemented on top of OpenGIS® Web Services. GeoDRM does not include per se, but does require and connect to capabilities for establishing trust between actors in OpenGIS® Web Services interactions.

6.3 GeoDRM Use Cases

The following are the use cases as posted in the RFQ and the clarifications that followed.

6.3.1 Use Case #1: Unrestricted Use License

Use Case Description: This use case describes “unrestricted” access to map layer resources based on a session license in which the user has read a statement of terms-of-use and agreed to them with a click-through gesture.	
Actors (Initiators): User of WMS	Actors (Receivers) Same as initiator
Pre-Conditions: <ul style="list-style-type: none"> - User requires WMS map layers. - User has access to WMS client. - User is able to discover WMS services with the needed layers through a CS/W catalog document 	Post-Conditions: <p>WMS map layers are viewable within the user’s WMS client software.</p>
System Components (may be combined) <ul style="list-style-type: none"> - GeoDRM-enabled CS/W: Catalog Service Web Profile - GeoDRM-enabled WMS: Web Map Service - GeoDRM-enabled Web WMS - GeoDRM-enabled Desktop WMS - (License) Broker: presents license offers and establishes licenses - (License) Manager: stores and matches licenses - GeoDRM Gatekeeper: decides whether a specific request is valid under a specific license - Authentication & Authorization: “security” implements authentication of license decision elements and authorization of consequences 	
Basic Course of Action:	

- | |
|---|
| <ol style="list-style-type: none"> 1. Client queries a CS-W and/or WMS to determine if needed map layers are available and under what terms 2. User selects layers of interest 3. GeoDRM Client obtains terms of use 4. User agrees to terms presented by GeoDRM Client 5. Client returns license acknowledgement to Broker Server 6. Broker Server stores established license with session identity and returns acknowledgement token 7. WMS/GeoDRM Client issues map layer request with license acknowledgement token to WMS 8. Gatekeeper Server validates identity of user and authenticity of license information, decides that license applies to request. 9. WMS returns map layer to client 10. (Alternate unrestricted use distribution) WMS Server seen by the client is cascading both the map layers and license offer / acknowledgement from one or more other servers |
|---|

6.3.2 Use Case #2: Distributor License

<p>Use Case Description: This use case describes “distributor” rights to WMS map layers. The provider of a cascading WMS operates under a license with an originating WMS to re-distribute on its own one or more map layers to clients under an unrestricted use license.</p>	
<p>Actors (Initiators): User of WMS and provider of cWMS</p>	<p>Actors (Receivers) Same as initiators</p>
<p>Pre-Conditions:</p> <ul style="list-style-type: none"> - User requires WMS map layers. - User has access to WMS client. - cWMS provider is able to cascade map layers from one or more originating WMS Servers 	<p>Post-Conditions:</p> <p>WMS map layers are viewable within the user’s WMS client software.</p>
<p>System Components</p> <ul style="list-style-type: none"> - CS/W: Catalog Service Web Profile - WMS: Web Map Service - cWMS: Cascading Web Map Service - License Broker: presents license offers and establishes licenses - License Manager: stores and matches licenses - License Gatekeeper: decides whether a specific request is valid under a specific license 	

- License Enforcer: “security” implements authentication of license decision elements and authorization of consequences
<p>Basic Course of Action:</p> <ol style="list-style-type: none"> 1. cWMS provider establishes a distributor license with an originating WMS for one or more map layers and receives a license acknowledgement token. 2. User queries a CS/W and/or the cWMS to determine if needed map layers are available and under what terms 3. User selects layers of interest 4. GeoDRM Client obtains terms of use 5. User agrees to terms 6. Broker Server stores established license and returns acknowledgement token 7. WMS/GeoDRM Client issues map layer request to cWMS with license acknowledgement token. 8. Gatekeeper Server validates identity of user and authenticity of license information, decides whether license applies to request. 9. cWMS issues map layer request to originating WMS with its own (distribution license) acknowledgement token 10. WMS returns map layer to cWMS 11. cWMS returns map layer(s) to client.

6.3.3 Use Case #3: End User License

<p>Use Case Description: This use case describes “end user” rights to WMS map layers and/or WFS feature collections for specifically identified individual users. The end user rights may be individual or may be based on an individual’s role (e.g. membership) in a licensed organization. The end user license may carry specific pre-conditions and constraints which need to be satisfied before a request can be honored.</p>	
Actors (Initiators): User of WMS	Actors (Receivers) Same as initiator
<p>Pre-Conditions:</p> <ul style="list-style-type: none"> - User requires WMS map layers. - User has access to WMS client. 	<p>Post-Conditions:</p> <p>WMS map layers are viewable within the user’s WMS client software.</p>
<p>System Components</p> <ul style="list-style-type: none"> - CS/W: Catalog Service Web Profile - WMS: Web Map Service - License Broker: presents license offers and establishes licenses - License Manager: stores and matches licenses - License Gatekeeper: decides whether a specific request is valid under a specific license 	

- Authentication & Authorization: “security” implements authentication of license decision elements and authorization of consequences
Basic Course of Action:
<ol style="list-style-type: none"> 1. User queries a CS/W and/or WMS/WFS and/or WMC to determine if needed map layers or datasets are available and under what terms 2. User selects layers and/or datasets of interest 3. User logs in and is authenticated with a specific identity (e.g. username/password) 4. Server matches identity with established individual or organization license and returns acknowledgement token 5. Client issues map layer or dataset request with license acknowledgement token 6. Server validates identity of user and authenticity of license information, decides whether license applies to request and whether any pre-conditions and constraints are met (e.g. time of request, area of request, state of daily usage quotas) 7. Server returns map layer or dataset to client

6.3.4 Use Case #4: WFS-T Feature Updater

Use Case Identifier: GeoDRM #4	Use Case Name: WFS-T Feature Updater
Use Case Domain: OWS-4 GeoDRM Feature Update	Status: Final 04/11/06
Use Case Description: This use case describes “Updater” rights to provide specific feature update transactions to a WFS-T server.	
Actors (Initiators): Remote editor / updater of feature collection	Actors (Receivers): Analyst reviewing, managing, and utilizing feature collection
Pre-Conditions: <ul style="list-style-type: none"> - Feature collection is configured through a WFS-T - Updater has new/changed features to transact - Updater has a WFS-T client - Updater has an established update licence - Analyst has a WFS-T client, authenticated session with WFS-T server, licence to review/approve feature updates, and licence to query / use feature collection 	Post-Conditions: <p>New/updated features are available for query from the WFS-T.</p>
System Components	

- WFS-T: GeoDRM-enabled Web Feature Service Transactional
- WNS: Web Notification Service
- Licence Manager: License Information Point
- GeoDRM Gatekeeper: License Decision Point
- WA2S: Authentication and Authorization Service
-

Basic Course of Action:

1. User #1 (feature updater) prepares new/updated features for transaction
2. User logs in at client and client establishes authenticated session with WFS-T / WFS-T access enforcement endpoint (client, server, and user are authenticated by WA2S to establish chain of trust)
3. User initiates a pending feature transaction against WFS-T
4. WFS-T access enforcement point requests authorization of transaction from the WA2S
5. WA2S determines that the update request requires a licence decision, WFS-T retrieves licence from Licence Manager corresponding to requested usage and requests validation from GeoDRM Gatekeeper.
6. GeoDRM Gatekeeper validates the transaction as a licenced usage and returns effect conditions, WA2S then authorizes the transaction with a constraint (notification and audit)
7. WFS-T performs (pending) transaction, responds to user with transaction id
8. WFS-T registers a transaction notification with WNS and a licenced usage with the Licence Manager according to licence effect conditions.
9. User #2 (analyst) is notified by WNS of a pending transaction.
10. User #2 retrieves usage record to verify licenced action and licencee.
11. User # 2 retrieves and reviews features in the pending transaction.
12. User # 2 approves and updates the status of the pending transaction.
13. New / updated features are available for use by other licenced users

7 Trust Model

The purpose of this chapter is to introduce a general trust model for OGC Web Services which is based on mainstream standards such as WS-Trust and WS-Security. The model leverages the concept of security tokens and describes several types of tokens and the interaction necessary to exchange these security tokens in order to enable the exchange of trusted messages between OpenGIS® Web Services and clients

The model is intentionally not specified at an implementation level – no encodings are described, although the WS-Trust and WS-Security are referenced in several places. It is the intention of the contributors to this report that this chapter be considered for the future update of the GeoDRM Reference Model.

The model described here leverages the trust model described in the WS-Trust specification. Here the trust is defined as “the characteristic that one entity is willing to rely upon a second entity to execute a set of actions and/or to make set of assertions about a set of subjects and/or scopes.” This model is based on a process in which a Web service can require that an incoming message prove a set of claims (e.g. name, keys, permissions, licenses, etc.). If a message arrives without having the required proof of claims (security tokens), the service SHOULD ignore or reject the message. A message can indicate its required claims and related information in its security policy.

A document defining the exact syntax for the security policy and the mechanisms to attach this policy to specific documents / registries (e.g. OGC capabilities, WSDL, catalog registries) is necessary. During OWS-4 experiments were conducted using WS-Policy, WS-SecurityPolicy and extending the Capabilities document. These are described in [OWS4EngVP].

7.1 WS-Trust Model

Authentication of requests is based on a combination of optional network and transport-provided security and information (claims) proven in the message. Requestors can authenticate recipients using network and transport-provided security, claims proven in messages, and encryption of the request using a key known to the recipient.

One way to demonstrate authorized use of a security token is to include a digital signature using the associated secret key (from a proof-of-possession token). This allows a requestor to prove a required set of claims by associating security tokens (e.g., PKIX, X.509 certificates) with the messages.

If the requestor does not have the necessary token(s) to prove required claims to a service, it can contact appropriate authorities (as indicated in the service's policy) and request the needed tokens with the proper claims. These "authorities", which we refer to as security token services, may in turn require their own set of claims for authenticating and authorizing the request for security tokens. Security token services form the basis of trust by issuing a range of security tokens that can be used to broker trust relationships between different trust domains.

This model is illustrated in the figure below, showing that any requestor may also be a service, and that the Security Token Service is a Web service (that is, it may express

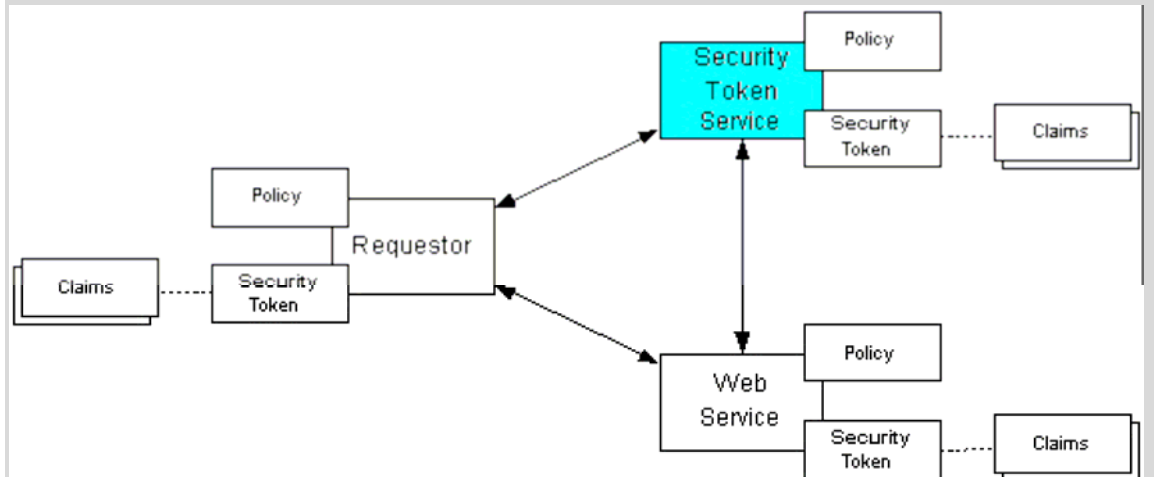


Figure 1: The trust model from WS-Trust

This general security model – claims, policies, and security tokens – subsumes and supports several more specific models such as identity-based authorization, access control lists, and capabilities-based authorization. It allows use of existing technologies such as X.509 public-key certificates, XML-based tokens, Kerberos shared-secret tickets, and even password digests. The general model in combination with the [WS-Security] and [WS-Policy] primitives is sufficient to construct higher-level key exchange, authentication, policy-based access control, auditing, and complex trust relationships.

We believe that this model can accommodate the requirements for GeoDRM and will, in the following, show how this model can be refined for this.

In the figure above the arrows represent possible communication paths; the requestor may obtain a token from the security token service, or it may have been obtained indirectly. The requestor then demonstrates authorized use of the token to the Web service. The Web service either trusts the issuing security token service or may request a token service to validate the token (or the Web service may validate the token itself).

In summary, the Web service has a policy applied to it, receives a message from a requestor that possibly includes security tokens, and may have some protection applied to it using [WS-Security] mechanisms. The following key steps are performed by the trust engine of a Web service (note that the order of processing is non-normative):

1. Verify that the claims in the token are sufficient to comply with the policy and that the message conforms to the policy.
2. Verify that the attributes of the claimant are proven by the signatures. In brokered trust models, the signature may not verify the identity of the claimant – it may verify the identity of the intermediary, who may simply assert the identity of the claimant. The claims are either proven or not based on policy.

3. Verify that the issuers of the security tokens (including all related and issuing security token) are trusted to issue the claims they have made. The trust engine may need to externally verify or broker tokens (that is, send tokens to a security token service in order to exchange them for other security tokens that it can use directly in its evaluation).

If these conditions are met, and the requestor is authorized to perform the operation, then the service can process the service request.

7.2 Models for Trust Brokering and Assessment

This section outlines different models for obtaining tokens and brokering trust. These methods depend on whether the token issuance is based on explicit requests (token acquisition) or if it is external to a message flow (out-of-band and trust management).

7.2.1 Token Acquisition

As part of a message flow, a request may be made of a security token service to exchange a security token (or some proof) of one form for another. The exchange request can be made either by a requestor or by another party on the requestor's behalf. If the security token service trusts the provided security token (for example, because it trusts the issuing authority of the provided security token), and the request can prove possession of that security token, then the exchange is processed by the security token service.

The previous paragraph illustrates an example of token acquisition in a direct trust relationship. In the case of a delegated request (one in which another party provides the request on behalf of the requestor rather than the requestor presenting it themselves), the security token service generating the new token may not need to trust the authority that issued the original token provided by the original requestor since it does trust the security token service that is engaging in the exchange for a new security token. The basis of the trust is the relationship between the two security token services.

7.2.2 Out-of-Band Token Acquisition

The previous section illustrated acquisition of tokens. That is, a specific request is made and the token is obtained. Another model involves out-of-band acquisition of tokens. For example, the token may be sent from an authority to a party without the token having been explicitly requested. As well, the token may have been obtained as part of a third-party or legacy protocol. In any of these cases the token is not received in response to a direct SOAP request.

7.2.3 Trust Bootstrap

An administrator or other trusted authority may designate that all tokens of a certain type are trusted (e.g. all Kerberos tokens from a specific realm or all X.509 tokens from a specific CA). The security token service maintains this as a trust axiom and can communicate this to trust engines to make their own trust decisions (or revoke it later), or the security token service may provide this function as a service to trusting services. There are several different mechanisms that can be used to bootstrap trust for a service. These mechanisms are non-normative and are not required in any way. That is, services

are free to bootstrap trust and establish trust among a domain of services or extend this trust to other domains using any mechanism.

Fixed trust roots – The simplest mechanism is where the recipient has a fixed set of trust relationships. It will then evaluate all requests to determine if they contain security tokens from one of the trusted roots.

Trust hierarchies – Building on the trust roots mechanism, a service may choose to allow hierarchies of trust so long as the trust chain eventually leads to one of the known trust roots. In some cases the recipient may require the sender to provide the full hierarchy. In other cases, the recipient may be able to dynamically fetch the tokens for the hierarchy from a token store.

Authentication service – Another approach is to use an authentication service. This can essentially be thought of as a fixed trust root where the recipient only trusts the authentication service. Consequently, the recipient forwards tokens to the authentication service, which replies with an authoritative statement (perhaps a separate token or a signed document) attesting to the authentication.

7.3 WS-Trust Security Token Service Framework

[WS-Trust] defines a general framework to be used by security token services for token exchanges. A requestor sends a request, and if the security policy permits and the recipient's requirements are met, then the requestor receives a security token in response. The specification defines the following two XML elements for this exchange:

1. <wst:RequestSecurityToken>
2. <wst:RequestSecurityTokenResponse>

An example illustrating the syntax defined in WS-Trust is showed below:

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

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

Additionally, WS-Trust defines a set of bindings that define functions to be implemented by a Security Token Service. The bindings leverage the model framework previously and specify service interfaces (WSDL descriptions are included with the specification). The bindings are optional (an implementation does not need to implement all bindings).

Additional bindings can be specified and further profiling of these bindings is also possible.

The following are the bindings described in the specification:

1. Issuance Binding – Defines the syntax for issuing of security tokens
2. Renew Binding – Defines the syntax for the renewal of security tokens
3. Validation Binding – Defines the syntax for validating security tokens
4. Cancel Binding – Defines the syntax for canceling security tokens

The following picture illustrates the functionality of a Security Token Service, as defined in WS-Trust.

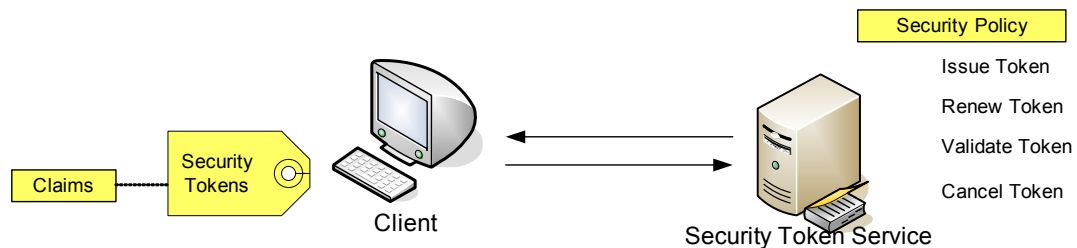


Figure 2: The functionality of a Security Token Service

Here, a Client is accessing the STS to execute one of the four operations (Issue / Renew / Validate / Cancel Token). The STS will first check to see if the requestor's claims are proved by corresponding security tokens and then check to determine whether the requestor's claims correspond to the security policy. If both conditions are met then the operation is executed.

Note that WS-Trust also specifies *negotiation and challenge extensions*. These allow the STS to challenge the client – for example a signature challenge, where the STS provides an arbitrary text and expects the client to sign it, in order to determine whether the client possesses a certain private key. Furthermore, the STS can also negotiate with the client – for example by asking the client to provide additional tokens. These challenges / negotiations can be iterated several times.

7.4 Trust model for OpenGIS® Web Services

This section describes a trust model based on the previously described framework. In this section we will show how the entities described in the OWS-4 RFQ and the GeoDRM Reference Model can be mapped on services. We will also describe the relation between these services, will suggest several architectural approaches (tight vs. loosely coupled) and for each of the proposed architectures we will show the communication paths.

7.4.1 Trust Model: The Client Perspective

The following picture shows the general trust model, picturing only the relations the client has with the different system entities:

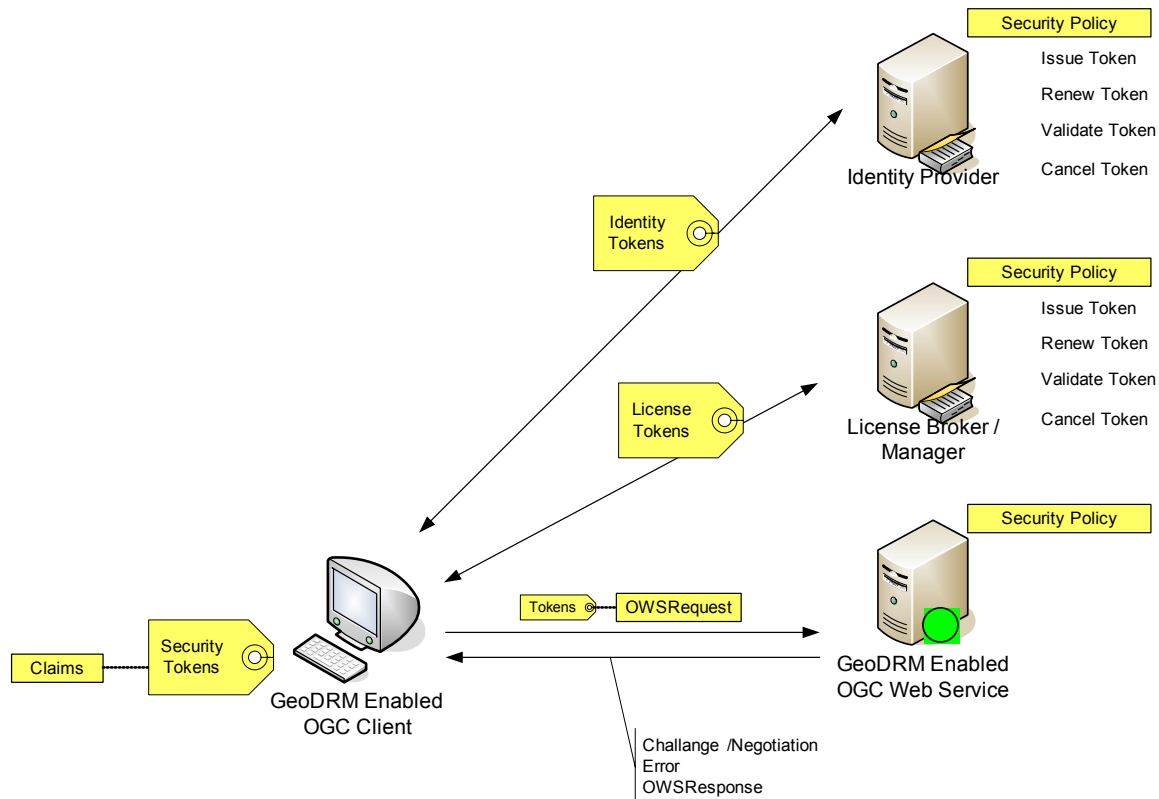


Figure 3: Trust model from the client perspective

The entities and their roles:

1. **Identity Provider (IP)** – An STS that manages identity tokens. Typically an IP will *issue identity tokens* to clients as a result of a negotiation such as asking the client to provide some other tokens or proofs of claims. A typical example can be exchanging one token for another (such as in a federation). Furthermore an IP can also *validate identity tokens* (typically on behalf of the GeoDRM Enabled Service) and *cancel such tokens*.

For the purposes of this document, identity tokens are any of the tokens described in [WS-Security] that convey identity claims.

The IP can have a Security Policy associated that describes the conditions that are placed on the client in order to establish the trust between the two (so that the client can invoke the operations of the IP). Such conditions might include the kind of tokens that the client should present, protocols constrains, etc.

Note that the IP is not a mandatory part of this trust model. A client can also obtain identity tokens by other out-of-band mechanisms (such as for example

registering to a web site where at the end of the registration process it receives a username and a password).

2. **License Broker / License Manager (LB/LM)** – An STS that manages license tokens. Typically a client will contact the LB/LM in order to procure license tokens that will be used when interacting with the GeoDRM Enabled Service. The LB/LM will typically *issue such license tokens* at the end of some interaction / negotiation (such as showing a proof of some claims – like having read and accepted a disclaimer or having paid a certain amount of money). Furthermore the LB/LM can also validate license tokens (typically on behalf of the GeoDRM Enabled Service), *renew / upgrade* such licenses and *cancel* them.

Similar to the IP, the LB/LM can have a Security Policy associated that describes the constraints that are placed on the client in order to establish trust between the two. Such conditions might include the kind of tokens that the client should present, protocols constrains, etc.

At this point no details are given as to how the LB&LM is implemented or how it communicates with the other services (IP / GeoDRM Enabled Service). These will be specified in the next section.

3. **GeoDRM Enabled OpenGIS® Web Service** – An OpenGIS® Web Service that enforces access restrictions based on the security tokens that are presented by the client together with its request. In order to enable such access control, this component might contact Identity Providers and License Managers in order to verify the tokens and / or retrieve additional information (identity attributes / license attributes / rights).

Similar to the IP and LB/LM, the GeoDRM Enabled Service can have a Security Policy associated that describes the constraints that are placed on the client in order to establish trust between the two. One important constraint is the kind of security tokens it accepts and the issuers of these tokens. For example, a service might choose to only accept certificate tokens that are issued by Verisign or Entrust, and license tokens that are issued by company A, B or C.

[WS-Trust] also enables challenges / negotiations. If such capabilities are supported, the service might ask the client to provide additional proofs of claims, which might force the client to revisit the IP or LB/LM in order to procure additional security tokens.

4. **GeoDRM Enabled OGC Client** – A client for the GeoDRM Enabled Service that is able to interact with the IP and LB/LM in order to obtain security tokens. The client should be able to manage these tokens locally, and to attach the appropriate token(s) with the OWSRequests.

Remarks:

The service entities above need not be separate entities. For example the IP and LB/LM can be one and the same entity – in this case the service will issue both

identity and license tokens. Alternatively, the GeoDRM Enabled Service might issue security tokens himself.

7.4.2 Trust Model: The (GeoDRM Enabled) OpenGIS® Web Service Perspective

The following picture shows the relations that the GeoDRM Enabled OpenGIS® Web Service has with the Identity Provider and the License Broker / License Manager. In the picture only one Identity Provider and one LB/LM are shown (for simplicity reasons). In reality a service might choose to trust several identity providers and / or several LB/LM.

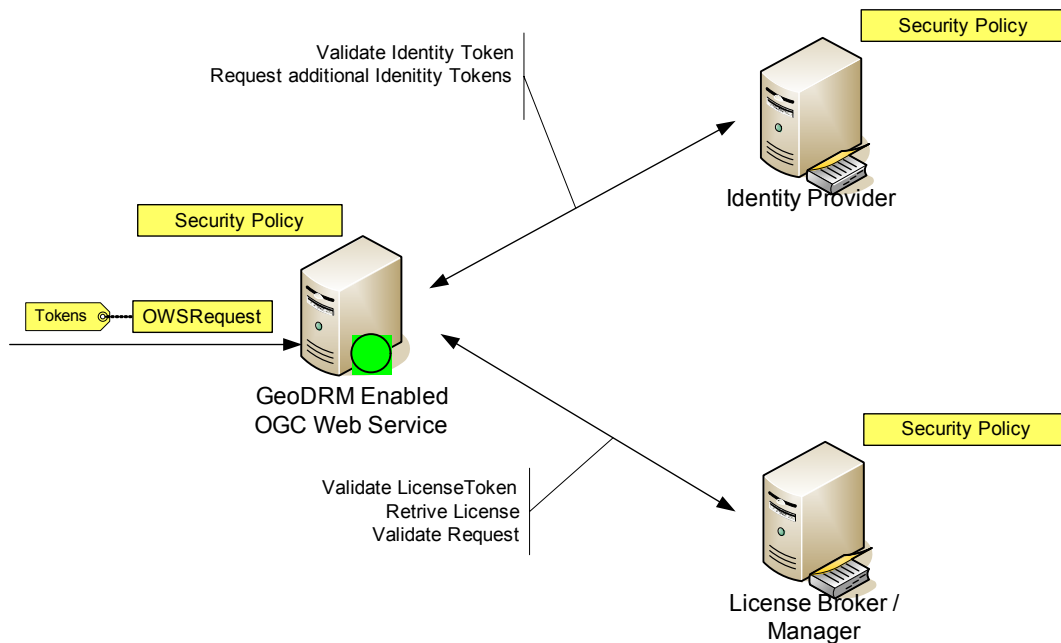


Figure 4: Trust model from the GeoDRM Enabled OpenGIS® perspective

As previously mentioned, the GeoDRM Enabled Service enforces access restrictions based on the security tokens it receives from the client. In order to assess the trust relation and to enforce the access control, it needs to make several verifications. These verifications are shown in the next figure, and detailed below.

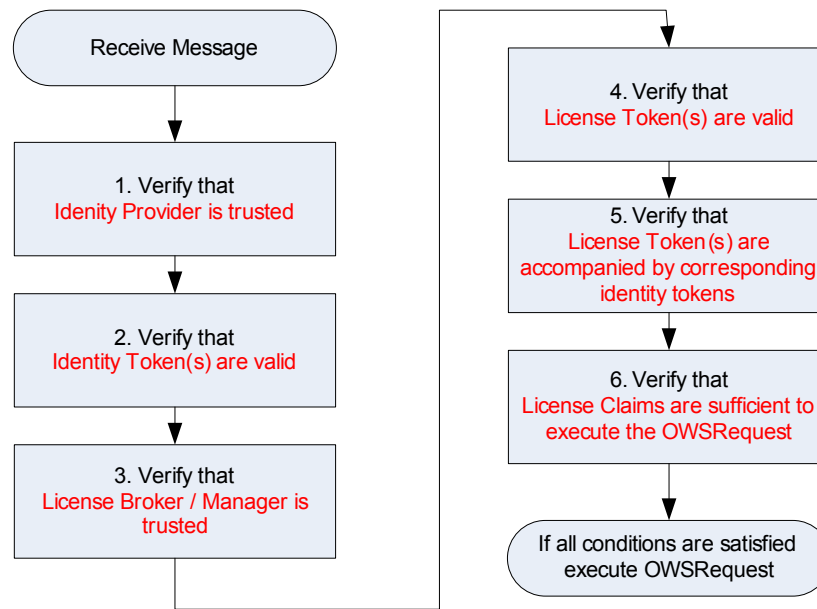


Figure 5: The verification steps

1. **Verify that the Identity Provider is trusted.** If the client presents an identity token, the service needs to determine whether it trusts the identity provider that issued the token – if applicable. For example, in the case of an X509, the certificate authority might be inspected. For some security tokens – such as username / password – this is not applicable.

This kind of information, e.g. which identity providers are trusted, should be described in the service security policy.

2. **Verify that the provided identity tokens are valid.** Depending on the type of identity token, this can be either done locally, by the service or alternatively, the service might contact the identity provider and request that the token be validated (as previously mentioned), [WS-Trust] provides a validation binding.
3. **Verify that the LB/LM is trusted.** If a license token is presented, the service needs to determine whether it trusts the issuing service. This kind of information (which LB/LM services are trusted) should be described in the service security policy.
4. **Verify that the license tokens are valid.** Depending on the type of license token, this can either be done locally, e.g. by checking the validity of digital signatures or alternatively, the service can contact the issuing authority (LB/LM) and request that the token be validated.
5. **Verify that the license tokens are accompanied by corresponding identity tokens.** It is expected that license be bounded to identity attributes (e.g. “John is allowed to view the map of Germany”) and therefore the client needs to prove these identity claims. This will typically be done by presenting identity tokens.

The service will check that the identity requirements from the license tokens are satisfied by the identity tokens, provided by the client.

6. **Verify that the license claims are sufficient to execute the OWS Request.** At this point, the service needs to figure out if the client has sufficient permissions to execute the OWS Request. This process depends very much on the type and content of the license token and will be detailed in the next section.

7.4.2.1 License Tokens^[1]

Depending on the architectural approach, two types of license tokens can be used, as illustrated in the picture below:

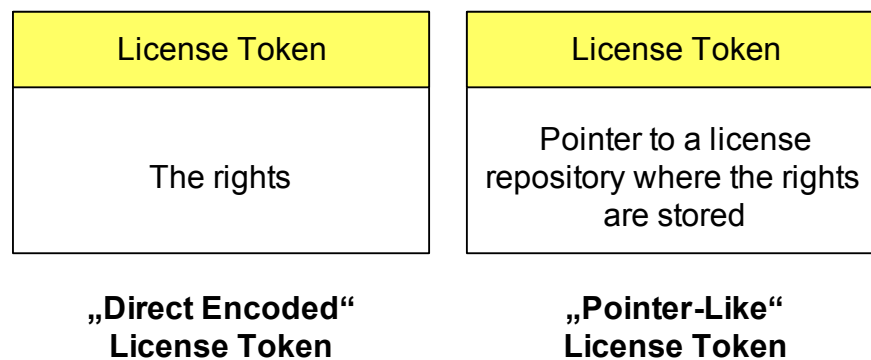


Figure 6: License token types

1. **“Direct Encoded” License Token Type.** The license token contains the permissions encoded in some machine readable format. An example of such a license token is an ISO REL token.
2. **“Pointer-Like” License Token Type.** The license token contains only a description of where the permissions can be found, but not an encoding of the permissions themselves. An example can be a SAML assertion stating that the permissions can be retrieved from some LB/LM service.

7.4.2.2 License Token Validation

Depending on the type of license token, the verification process is as follows:

1. **“Direct Encoded” License Token.** The LB/LM needs not be contacted in order to do the verification; this can be done locally by the service. For this, the service will compare the permissions encoded in the license with the OWS Request.
2. **“Pointer Like” License Token.** Because the service does not know the permissions, the LB/LM needs to be contacted for this. In this case, the following two architectural approaches can be envisioned (see next picture):
 - a. **Tight coupling.** The GeoDRM Enabled Service will contact the LB/LM for every client request, sending the OWSRequest and the license tokens and expecting in return an “authorized / not authorized” response.

If we have the XACML model in mind, then the LB/LM will act as the Policy Decision Point (PDP), while the GeoDRM Enabled Service will contain / represent a Policy Enforcement Point (PEP).

- b. Loosely coupling.** The GeoDRM Enabled Service requests the license permissions that correspond to the license token. After receiving the license, the service can make the checks locally – compare the permissions with the OWS Request and determine if the request is authorized or not.

If we have the XACML model in mind, then the LB/LM will act as the Policy Administration Point (PAP), while the GeoDRM Enabled Service will act as both Policy Decision Point (PDP) and Policy Enforcement Point (PEP).

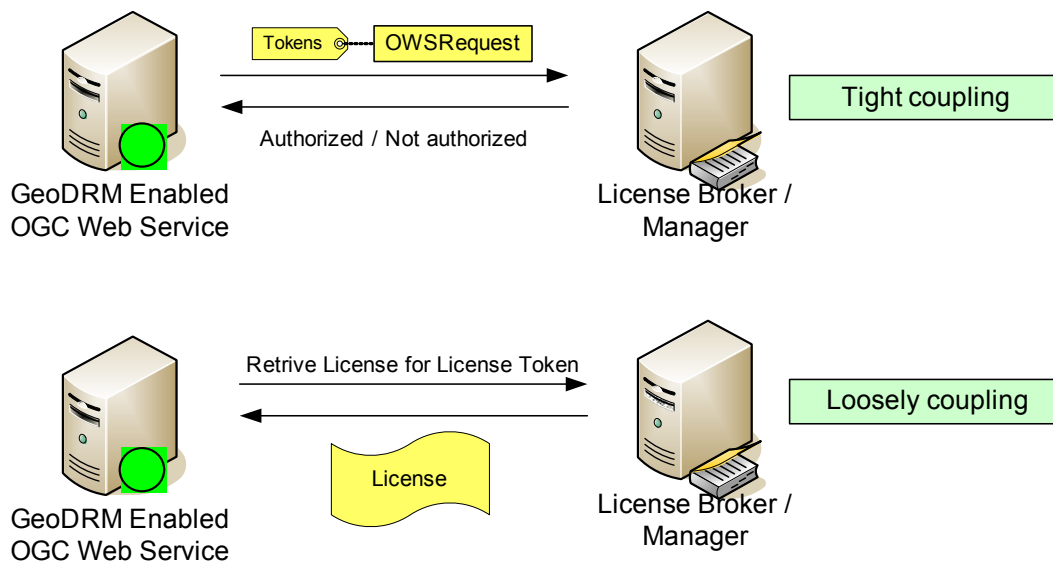


Figure 7: Distribution choices for enforcement

7.4.3 Trust model for cascading scenarios

A cascading web service is a service which is not able to respond to service requests alone. Instead it first makes requests to one or more services (“cascaded services”), waits for the responses, and then combines the results (possibly adding local data) in order to generate the response. Typical examples for cascading services are the Cascading WMS where a part of the layers are not stored locally but retrieved from other WMS services and the Feature Portrayal Service (FPS) which renders GML data that is retrieved from a (third party) WFS. Service chaining is described in the OGC Abstract Specification Topic 12: OpenGIS Service Architecture [Topic12].

The following picture shows a typical cascading scenario. The Client, the Cascading Service and the Cascaded Services (A and B) are considered to be GeoDRM enabled.

This is necessary because they all need to understand requests accompanied by tokens and need to process the tokens.

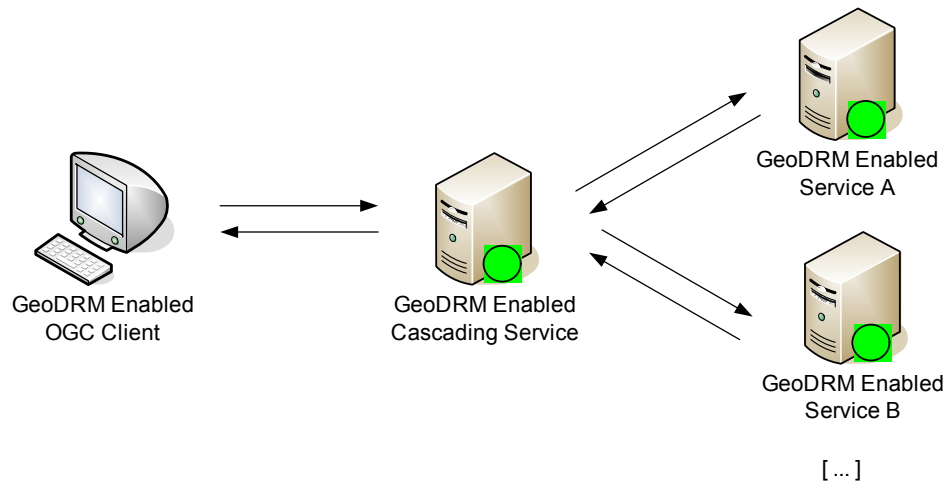


Figure 8: A typical service chaining scenario

Before going any further, it is necessary to make a couple of remarks / assumption with regard to the actions of each of the software agents:

- i) From a client – server perspective, the Cascading Service is acting as a client to the cascaded services (A and B)
- ii) Since the Cascading Service is requesting information from services A and B in order to fulfil the request it got from the Client, we will consider that the Cascading Service is requesting information from services A and B on behalf of the Client
- iii) The Client and the Cascading Service have different identities, which are backed by different security tokens and these are not transmittable. Under no circumstances will the Client forward credentials to the Cascading Service such that the Cascading Service is able to make the services believe he is the Client. For example, if username and password authentication is used, the Client shall never communicate the password to the Cascading Service.

7.4.3.1 Different types of service chaining

The OGC Abstract Specification, Topic 12 (see [topic12]) enumerates 3 architectural patterns for service chaining. These are the following:

- i) **User defined (transparent) chaining**, where the user defines and controls the order of execution of the individual services. Here the details of the services are not hidden from the user and therefore the user is able to provide input to the services along the chain.

In this case, the user invokes each of the services from the chain. As such, from the trust point of view, the interactions are similar with the ones where

no chaining is involved. A client will need to procure the necessary identity and license tokens and then invoke one by one each of the services. When invoking a service, the client will send the necessary tokens together with his request.

- ii) **Workflow-Managed (translucent) chaining**, where the chain is managed by a workflow service. The user's involvement in the steps of the chain is mostly one of watching the individual services. The chain exists prior to the user executing the request. The user has knowledge of the services along the chain and therefore is able to provide input to the services along the chain.

In this case, the user will not invoke the services himself, but instead will invoke the workflow manager, which will in turn invoke the services on his behalf. From a trust point of view, the user needs to provide the necessary tokens for all services along the chain when invoking the service manager. The service manager shall then make sure that he sends appropriate tokens to the services along the chain.

This scenario can be seen as a typical delegation problem (the user delegates the workflow manager to invoke the other services on his behalf). This scenario is the most complicated and is further described in the next sections.

- iii) **Aggregate Service (opaque chaining)**, where the service chain appears as a single aggregate service which handles all requests. The user has no awareness that there is a set of services behind the aggregate service. Because the user has no knowledge of the chain, he is not able to provide input to the services along the chain.

In this case, the user has no knowledge of the chain, therefore he will not provide any tokens for the services along the chain. When making request, the aggregator will make the requests on his own behalf and not on behalf of the user. This is because the client can not delegate the aggregator to execute some actions he is not aware of.

7.4.3.2 The Workflow

The following picture shows the interactions between the Client, Cascading Service and the cascaded services together with the security elements. The same Trust model is used, where services have an associated Security Policy which describes what tokens are necessary in order for the request to be processed and clients have an associated set of Claims which are backed up by Security Tokens. Since the Cascading Service is both a client and a service it has both of these elements.

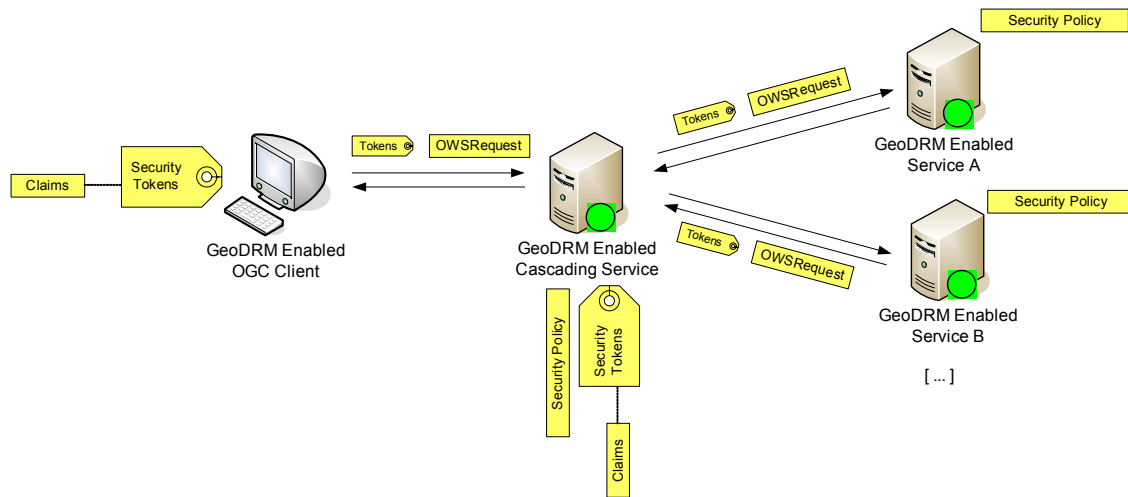


Figure 9: Tokens & claims in a service chaining scenario

The following sections detail the communication between the software components. The requirements which are set on the messages are described (what tokens are necessary, what information should these tokens convey).

7.4.3.2.1 The interaction Client – Cascading Service

As mentioned in the previous section this scenario is an example of translucent chaining and the client will need to supply security information which is destined to both the Cascading Service and the services which the Cascading Service will later invoke.

The Cascading Service is just like all other OGC Services, and therefore the trust model described in this document will apply it. This means that the service may require that the client presents an identity token and a license token in order for the service to further process the request.

Furthermore, since the cascaded services (in our example A and B) also require license tokens, the client will need to provide these license tokens to the cascading service. The cascading service will use these tokens when making the cascading requests. Because a cascading service may invoke several services in order to fulfill one request from the client, it would be useful to have information about the service to which the license token refers stored in the license token.

To give an example, for the picture above, the client would need to send one identity token and three license tokens with his request (one license token for the Cascading Service, one license token for service A and one license token for service B).

7.4.3.2.2 The Cascading Service – cascaded services

At this point, the Cascading Service is interacting with the cascaded services as if no service chaining were involved. The Cascading Service will act just like a regular client which contacts a GeoDRM enabled service.

The difference though is where it gets the tokens. When making a request, the Cascading Service will attach a token claiming its own identity. The license token will, on the other hand side, come from the client as discussed in the previous section. The license token should have an associated license stating that the Cascading Service is permitted to access the services when acting on behalf of the user.

Furthermore, the Cascading Service shall prove that it is acting on behalf of the client. This is not a trivial problem. To some extents this may be solved by having the client send an additional client which will allow the Cascading Service to prove it's acting on his behalf. Additionally, the client can license the Cascading Service access to services only for a limited period of time (for examples a couple of seconds) ~ while this will not prove that the Cascading Service is acting on the client's behalf, this would not allow the Cascading Service to abusive behavior. This problem will not be further detailed in this document. Other documents shall specify how this shall be achieved and in which conditions this is necessary. For the purpose of this document we will consider that the Cascading Service is trusted to act on behalf of the client.

Because one Cascading Service may invoke several services which may be in different administrative domains and therefore different licenses may be required, the Cascading Service will need to choose from the tokens provided by the client the ones that are valid for each service that is invoked. This may be done for example by inspecting the Security Policy of each service. Alternatively (and perhaps better), the license token should contain information regarding the services where it is accepted and valid.

To summarize, when invoking a cascaded service, the Cascading Service shall provide the following, in addition to the OWS request:

1. Identity Token (claiming his own identity)
1. License Token (from client) claiming that the Cascading Service is allowed to access the service when acting on the client's behalf
2. (Optional) Proof that the Cascading Service is acting on behalf of the client

The following picture illustrates this:

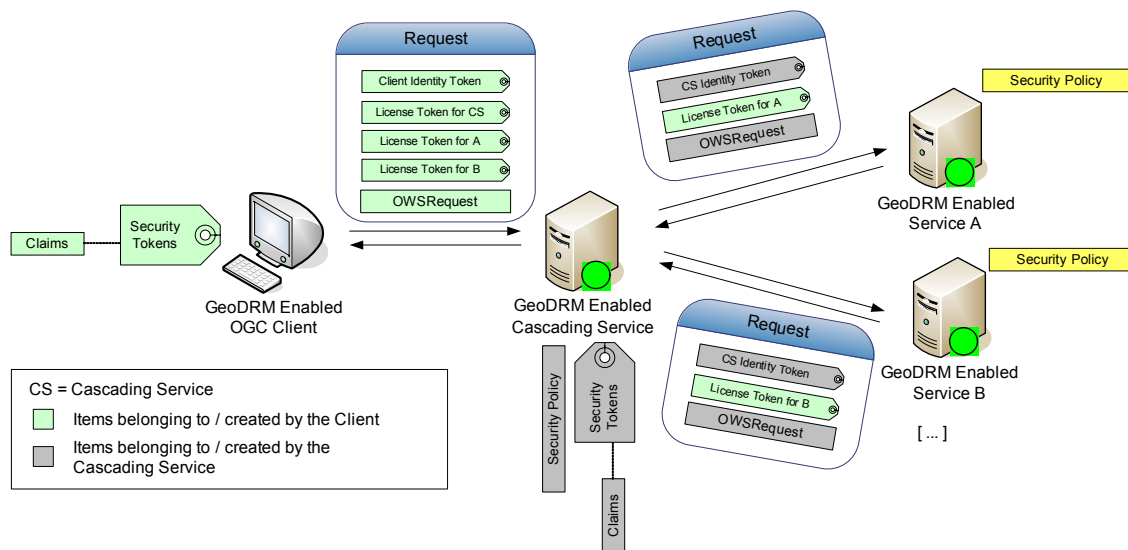


Figure 10: Tokens flow in a service chaining scenario

7.4.3.3 Error handling / Negotiation of Credentials

Because the Cascading Service is situated between the Client and the cascaded services, it should mediate error handling and negotiation of credentials.

If an error occurs when invoking one of the cascaded services, the Cascading Service will not be able to fulfil the client request and will therefore return an error to the client. The error should inform the client about the original error.

If negotiation is supported as described in this document, section 7.3, the negotiation should be mediated by the Cascading Service. For example, if the Cascading Service invokes service A, and this one asks for additional tokens (for example another license token is required), the Cascading Service should then forward this information to the client, requiring that it passes additional tokens with his request.

8 Model Implementation

The purpose of this chapter is to give guidance as to how the model presented in the previous chapter could be implemented. This section will present how existing standards such as WS-Security, WS-Trust, XACML, SAML can be used to implement the concepts previously described.

The intention of this chapter is to provide guidance as to what is possible to achieve using and / or adapting existing mainstream standards proposed by organizations such as OASIS, IETF and W3C. The chapter will not describe in depth how these standards are to be applied, but only provide guidance for this. The following should be taken as an input for the following OGC testbeds (such as OWS5) and for working groups within the OGC – especially the GeoDRM WG and the Security WG.

8.1 Identity tokens

Identity tokens are fragments of information that are used by the client to prove its identity to the service provider. Depending on the authentication method being used, these tokens convey different information. Furthermore, the authentication method being used also has implication on how these tokens are procured and the type of trust relation (see 7.2 for more information about trust brokering and token acquisition).

Identity tokens, as far as this document is concerned, are sent together with a request to the service provider. That is, every request to the service provider that needs to be authenticated (each message exchanged) will contain one (or more) identity token. These tokens are attached to the message in a manner that would not allow eavesdroppers to capture the token and use it in reply attacks.

During OWS4 the group only focused on SOAP as the transport protocol. SOAP was chosen because a lot of the standardization effort in the field of security was invested in it during the past years. Future initiatives of the OGC should investigate how the Get and Post bindings that use HTTP as transport protocol can accommodate security tokens.

One way to implement security tokens is via the WS-Security specification from OASIS. This specification proposes a standard set of SOAP [SOAP11, SOAP12] extensions that can be used when building secure Web services to implement message content integrity and confidentiality. This specification is flexible and is designed to be used as the basis for securing Web services within a wide variety of security models including PKI, Kerberos, and SSL. Specifically, this specification provides support for multiple security token formats, multiple trust domains, multiple signature formats, and multiple encryption technologies. The token formats and semantics for using these are defined in the associated profile documents.

It is not the purpose of this document to describe in detail the syntax and semantics for identity tokens and how each token type should be attached to SOAP messages. This information is found in WS-Security and its accompanying token profiles. In the

following we will only enumerate the authentication method and point to the WSS token profile

8.1.1 Username / Password

Username / Password authentication is described in the Username Token Profile v1.1 available at the following address:

<http://www.oasis-open.org/committees/download.php/16782/wss-v1.1-spec-os-UsernameTokenProfile.pdf>

Although this authentication method is quite trivial, the above mentioned specification does a good job by adding features that help protecting the password and avoiding replay attacks. These features include the use of hashed passwords, nonces and / or creation dates. Furthermore a key derivation algorithm is presented which can be used when computing Message Authentication Codes (MACs) or as a symmetric key for encryption.

This authentication method was used during OWS4, see section 9.1.1 for details.

8.1.2 Kerberos

Kerberos is a computer network authentication protocol which allows individuals communicating in an insecure network to prove their identity to one another in a secure fashion. Kerberos is a project originating from MIT, which has various implementations, one of them being used in Microsoft Windows networks.

The Kerberos Token Profile v1.1 (with errata) is available at the following address:

<http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-errata-os-KerberosTokenProfile.pdf>

The document describes the use of Kerberos tokens with respect to WS-Security. It specifies how to encode Kerberos tickets in SOAP messages and how use these tokens for digital signatures and encryption.

This authentication method was not used during OWS4.

8.1.3 PKI / X509 Certificates

An X.509 certificate specifies a binding between a public key and a set of attributes that includes (at least) a subject name, issuer name, serial number and validity interval. Public Key Infrastructures based on X.509 certificates are widely deployed today for authentication (usually done by means of digital signatures) and encryption.

The use of X.509 authentication framework in the context of SOAP web services is described in X.509 Token Profile v1.1 (with errata), available at the following address:

<http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-errata-os-x509TokenProfile.pdf>

The document describes how X.509 certificates (or references to them) can be attached to SOAP messages. Furthermore the document describes how digital signatures and encryption blocks can reference keys attached in this fashion.

If a digital signature is verified and the key used for the digital signature is associated to an identity, a service provider can assert, using the X.509 certificate attached to the message (and assuming that this certificate is valid) the identity of the message emitter.

Authentication via X.509 certificates has been used during OWS4, see section 9.1.2 for details.

8.1.4 SAML

Security Assertion Markup Language (SAML) is an XML standard from OASIS designed for exchanging security information. SAML is a very flexible specification and can be used in a multitude of scenarios. One of the problems that it tries to solve is the Single Sign On problem and in this way it can be used for authentication in different scenarios.

SAML Token Profile v1.1 (with errata) describes how SAML assertions can be attached to SOAP messages. The document can be found at the following address:

<http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-errata-os-SAMLTokenProfile.pdf>

Because it addresses Single Sign On, SAML is of course ideal for federation scenarios where users come from a different security domain than the one of the service provider.

SAML has been used for authentication during OWS4, see section 9.1.3 for more information.

Please note, that SAML was also used as a way to encode license tokens (see sections 8.2 and 9.2 for more information).

8.2 Licenses and License tokens

A GeoLicense is the expression of the rights and constraints on those rights to be performed against a geospatial resource. It is the container expressing the rights to use a specified geospatial resource, for a given geographical space, over a specific period of time – subject to other conditions (from [GeoDRM RM]).

License tokens are the encoding of licenses in a machine readable language. That is, license tokens are fragments of information that assert permissions / rights on (in our case) a geospatial resource. Section 7.4 describes the lifecycle of license tokens in more details and presents the two choices for implementation: direct-encoded license tokens and pointer-like license tokens.

During OWS4 we investigated several ways of encoding license tokens and concluded that SAML is a good choice because of several reasons: it offers an interoperable framework, it is XML based, it allows for extension and the definition of specific data models. Also, there are specifications describing how SAML assertions can be attached to SOAP messages.

8.2.1 Direct-encoded license tokens

As mentioned direct-encoded license tokens convey information regarding permissions / rights on one or more specific geospatial resources. During the OWS-4 initiative the GeoDRM group has chosen XACML as the encoding choice for licenses. XACML v2.0 is available at the following URL:

<http://docs.oasis-open.org/xacml/2.0/XACML-2.0-OS-ALL.zip>

Because geospatial restrictions are useful, GeoXACML OGC #05-036 (currently discussion paper) is an important point of start. It can be found at the following URL:

http://portal.opengeospatial.org/files/index.php?artifact_id=10471

Specifically, the SAML profile of XACML describes how XACML policies can be exchanged by means of SAML. This document is an ideal point of start for the encoding of license tokens. The document can be found at the following URL:

http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-saml-profile-spec-os.pdf

Furthermore, depending on the requirements, there are other options for the encoding of permissions. The following standards could be suited as license encodings, however these were not tested during the OWS-4 initiative:

- ISO REL / GeoREL OGC #06-172r1 – A geospatial extension of ISO REL
- ODRL (Open Digital Rights Management) – For details see <http://odrl.net/>

8.2.2 Pointer-like license tokens

Pointer-like license tokens only contain information about where the real license is stored and about how it can be retrieved. During OWS-4 the group decided that the SAML framework is a good choice for encoding such tokens. SAML is a specification from OASIS and can be found at the following internet address:

<http://docs.oasis-open.org/security/saml/v2.0/saml-2.0-os.zip> (SAML v2.0)

<http://www.oasis-open.org/committees/download.php/3400/oasis-sstc-saml-1.1-pdf-xsd.zip> (SAML v1.1)

Specifically, the GeoDRM group work in OWS-4 found that SAML attribute assertions are a good choice for encoding pointer-like license tokens. The exact implementation is described in section 9.2 and in the document GeoDRM Engineering Viewpoint and Supporting Architecture in OWS4.

8.3 The License Broker / Manager

The license manager / license broker are responsible for the management of the license tokens for the entire life of the tokens. This includes the following phases:

- Issuing / Creation

06-107r1

- Update / Renewal
- Revocation / Cancellation

As already mentioned in chapter 7, the license manager / license broker is, from an implementation point of view, a Security Token Service.

9 Model Implementation in OWS4

The purpose of this chapter is to show how the trust model was implemented during the OWS4 initiative. As seen in the previous chapter there is quite an array of standards and specifications that could be used when implementing the trust model described in chapter 7. Furthermore some of the specifications (such as WS-Security for example) have several profiles and investigating all these would have been a tremendous amount of work. Because of the limited amount of resources and time available within the OWS-4 initiative it was not possible to experiment with all of them.

This chapter describes the implementations choices of the GeoDRM group within OWS-4.

9.1 Identity tokens in OWS4

The following 3 topics show how identity tokens were used in OWS-4.

Attention is drawn that in order to have an implementation running one single type of identity token is enough.

However depending on the exact requirements of the application scenario, one type of authentication would prove more appropriate than the others. In short, the following are the advantages and disadvantages of each of the 3 authentication methods tested in OWS-4:

- Username / Password
 - Easy to implement, easy to use, users are familiar with the authentication process (everybody knows what a username / password is), registration process is very simple, widely deployed on the internet
 - Not very secure, passwords can be forgotten, lost, stolen
- Public Key Infrastructure / X.509 Certificates
 - Secure way of authentication, can be combined with data encryption, hierarchic trust roots allow for simple user management, widely deployed on the internet
 - Key management, key exchange can become complicated
- The Fraunhofer authentication service
 - Useful in federation scenarios, where a client accesses several services using the same credentials. It removes the burden of user management from each service

- An additional “log-in” step needs to be made before making requests to the OGC service, an interface needs to be defined for this service (which need to be implemented in the clients)

9.1.1 Username / Password Tokens (UniBW)

Username / password authentication was implemented during OWS4 by the University of the German Armed Forces (UniBW).

9.1.1.1 Acquiring the token

Authentication by username / password is a form of direct trust, where there is a direct trust relation between the client making the request and the GeoDRM enabled service.

Token acquisition for username / password tokens was not an issue to be investigated during OWS4. The mechanisms by which the token is acquired by the client are considered to be out-of-band (see 7.2).

The usual way of acquiring a username and password is a registration process where a user goes to a web site where he fills out a form. The server can then verify the information provided by the user (for example by checking if the email address is correct or by asking the user to provide credit card information). At the end of the registration process, the user receives a username and password that he can use when accessing the GeoDRM enabled service.

This process of registration was investigated and also implemented during the OWS-3 initiative. Here the user was required to register to a web site where he was presented with the terms of use for a specific WMS / WFS server. After accepting the terms and conditions he was able to use the service unrestricted.

9.1.1.2 Attaching the token to a OWS Request Message

When attaching a username token to a SOAP message, the Username Token Profile v.1.0 was used. An example is shown below. The example shows the SOAP message for a WFS request (GetFeature). Important parts are marked with bold and italics.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  <SOAP-ENV:Header>
    <wsse:Security
      xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-wssecurity-secext-1.0.xsd">
      <Assertion xmlns="urn:oasis:names:tc:SAML:1.0:assertion"
        xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
        xmlns:samlp="urn:oasis:names:tc:SAML:1.0:protocol"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        AssertionID="_8dbf0b675e095567bb6e2e8fc81144fc"
        IssueInstant="2006-10-31T10:52:53.703Z"
        Issuer="http://iisdemo.informatik.unibw-
muenchen.de/ows4/DummyLicenseManager"
        MajorVersion="1" MinorVersion="1">
```

```

.....
    </Assertion>
    <wsse:UsernameToken
      xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-wssecurity-utility-1.0.xsd"
      wsu:Id="UsernameToken-12864175"
      xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-wssecurity-secext-1.0.xsd">
      <wsse:Username
        xmlns:wsse="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
        NGA-Officer
      </wsse:Username>
      <wsse:Password
        Type="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-username-token-profile-1.0#PasswordDigest"
        xmlns:wsse="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
        2W3Yp3WX21CGM4nX0V+1lU9vb3o=
      </wsse:Password>
      <wsse:Nonce
        xmlns:wsse="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
        u9ue3VrobUibsYCPI5Sv4Q==
      </wsse:Nonce>
      <wsu:Created
        xmlns:wsu="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
        2007-01-15T13:55:50.328Z
      </wsu:Created>
    </wsse:UsernameToken>
  </wsse:Security>

</SOAP-ENV:Header>
<SOAP-ENV:Body>
  <ns1:GetFeature xmlns:ns1="http://www.opengis.net/wfs"
    outputFormat="GML2" service="WFS" version="1.0.0">
    <ns1:Query typeName="tiger:poi">
      <ogc:PropertyName
        xmlns:ogc="http://www.opengis.net/ogc">
        the_geom
      </ogc:PropertyName>
      <ogc:Filter xmlns:ogc="http://www.opengis.net/ogc">
        <ogc:BBOX>
          <ogc:PropertyName>the_geom</ogc:PropertyName>
          <gml:Box xmlns:gml="http://www.opengis.net/gml"
            srsName="EPSG:4326">
            <gml:coordinates cs="," decimal="."
              ts=" ">
              -73.95533279299448,40.82414581720157
              -73.93855481003192,40.832408895114554
            </gml:coordinates>
          </gml:Box>
        </ogc:BBOX>
      </ogc:Filter>
    </ns1:Query>
  </ns1:GetFeature>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

The username is in clear, here “NGA-Officer”. The password is hashed. To prevent replay attacks, a nonce and creation timestamp are attached to the token. The hash is done on the password + nonce + creation date (this is described in the above mentioned specification).

The implementation uses the WSS4J library from Apache:

<http://ws.apache.org/wss4j/>

9.1.2 X509 Certificates (UniBW)

Authentication using X.509 certificates was implemented during OWS-4 by the University of the German Armed Forces.

9.1.2.1 Acquiring the token

Authentication via digital signature with X.509 certificates is a form of direct brokered trust (see 7.2 for more details).

Token acquisition for X.509 certificates and the exchange of public / private keys was not an issue for OWS-4. For the implementation, we considered that the client possesses a private key and that the server possesses the public key of the certificate authority and is therefore able to prove if the client certificate is valid or not. The UniBW implementation shows how certificates and private keys could be either stored in a file (a java keystore) or on a smartcard.

9.1.2.2 Attaching the token to a OWS Request Message

Authentication via digital signature via X.509 is done by:

1. Attaching the certificate or other information that can be used by the service to uniquely identify the client’s X.509 certificate
2. Proofing to the service that the client possesses the private key which corresponds to the certificate. For this, a digital signature is done on the contents of the SOAP body.

Below there is an example that shows a WFS DescribeFeatureType request. The SOAP message contains a binary security token (this exact content of the security token is omitted for clarity) and a signature block. Both are shown using bold and italics.

The digital signature references the SOAP body (#id-29876954).

```
<?xml version="1.0" encoding="utf-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  <SOAP-ENV:Header>
    <wsse:Security
      xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-wssecurity-secext-1.0.xsd">
```

```

    <Assertion AssertionID=" _b21f6dae7ce0a2e9cf040dbfe5186804"
      IssueInstant="2006-11-27T12:52:27.171Z"
      Issuer="http://iisdemo.informatik.unibw-
muenchen.de/ows4/DummyLicenseManager"
      MajorVersion="1" MinorVersion="1"
      xmlns="urn:oasis:names:tc:SAML:1.0:assertion">
..... SKIPPED .....
    </Assertion>
    <wsse:BinarySecurityToken
      EncodingType="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-soap-message-security-
1.0#Base64Binary"
      ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-x509-token-profile-1.0#X509v3"
      wsu:Id="CertId-2039344"
      xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-wssecurity-utility-1.0.xsd">
..... SKIPPED .....
    </wsse:BinarySecurityToken>
    <ds:Signature Id="Signature-26977856"
      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#dsa-sha1" />
        <ds:Reference URI="#id-29876954">
          <ds:Transforms>
            <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>
            DwpivREhH/w97zHhH0BT7/htwz0=
          </ds:DigestValue>
        </ds:Reference>
      </ds:SignedInfo>
      <ds:SignatureValue>
        La9TDltuyzaXrX70SQQEPzR62YtHo7iarGh0DUUVN41qOfb9hFvjzw==
      </ds:SignatureValue>
      <ds:KeyInfo Id="KeyId-29345020">
        <wsse:SecurityTokenReference wsu:Id="STRId-18724539"
          xmlns:wsu="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
          <wsse:Reference URI="#CertId-2039344"
            ValueType="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3" />
          </wsse:SecurityTokenReference>
        </ds:KeyInfo>
      </ds:Signature>
    </wsse:Security>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body wsu:Id="id-29876954"
    xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-wssecurity-utility-1.0.xsd">
    <ns1:DescribeFeatureType outputFormat="XMLSCHEMA" service="WFS"
      version="1.0.0"

```

```

        xsi:schemaLocation="http://www.opengis.net/wfs
        ../wfs/1.1.0/WFS.xsd"
        xmlns:ns1="http://www.opengis.net/wfs"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    </ns1:DescribeFeatureType>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>


```

9.1.3 SAML Assertions / The Authentication Service proposed by Fraunhofer

9.1.3.1 Acquiring and attaching the token to a OWS request

Authentication via SAML Assertions as implemented by Fraunhofer is a form of indirect brokered trust (see 4.9, 7.2 for more details).

If the GeoDRM enabled client evaluates the preconditions to enforce user authentication for usage of the protected service, the user has to obtain an account at the AuthenticationService. The creation and verification of user accounts by an identity provider and the enablement of trust between the AuthenticationService (identity provider) and the GeoDRM gatekeeper (service provider) is out-of-bands in this description. The AuthenticationService interface is described as follows.

 Interface ^[e2]
<ul style="list-style-type: none"> ▪ GetSAMLResponse <ul style="list-style-type: none"> ▪ Description: Get an authentication SAML artifact ▪ Parameters: Credentials ▪ Returns: SAMLAssertion with AuthenticationStatement, Exception

Starting from a user getting authenticated by the AuthenticationService, the GeoDRM client obtains identity information for further proceeding. The obtained information artifact has to be attached to a request, e.g. by using [WS-S] [SAMLToken]. If identity information is signed by a trusted party [XMLSig] the GeoDRM gatekeeper (service provider) itself is able to decide whether the request was submitted by an authenticated subject or not. In case of receiving an identity reference the GeoDRM gatekeeper (service provider) has to call back to an appropriate identity provider (AuthenticationService) for authentication.

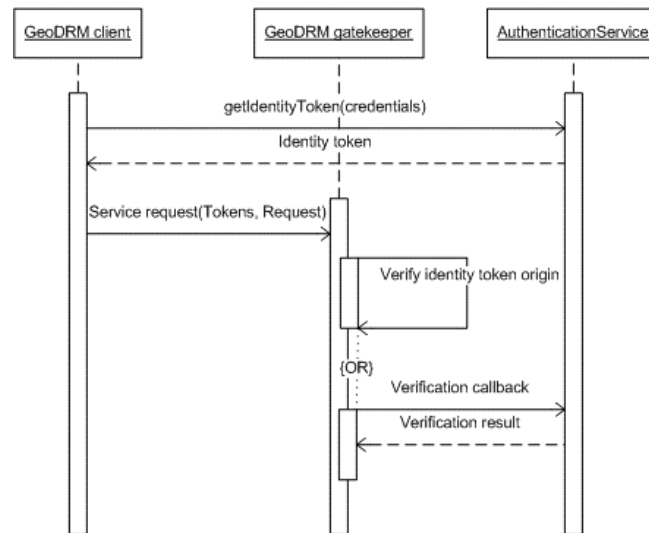


Figure 11: Authentication sequence

The assertion, is digitally signed and attached to the SOAP WS-Security header of the DRM-enabled OWS request. Thus the service provider is able to check whether the submitted request was issued by a party known to be a trusted identity provider.

The implementation used the WSS4J library from Apache:

<http://ws.apache.org/wss4j/>

9.1.4 Identity Token Encoding

Security Assertion Markup Language makes use of statements which assert certain characteristics of a subject (claims), e.g. a subject's authentication, name and role. SAML can be used to encode qualified identity tokens and may be combined with XML Digital Signature.

SAML denotes a various types of statements for expressing identity information, which all are referring to an included subject. The structure of an assertion object is shown below.

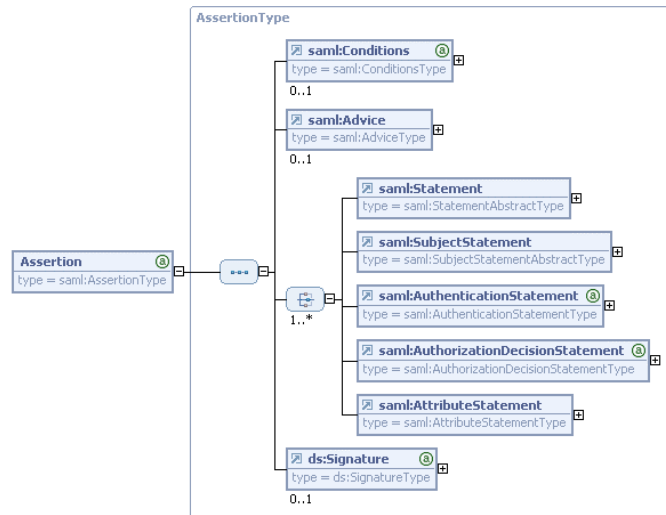


Figure 12: SAML Assertion

An AuthenticationStatement, which is used to assert that a subject did indeed authenticate with the identity provider at a particular time using a particular method of authentication is depicted below.

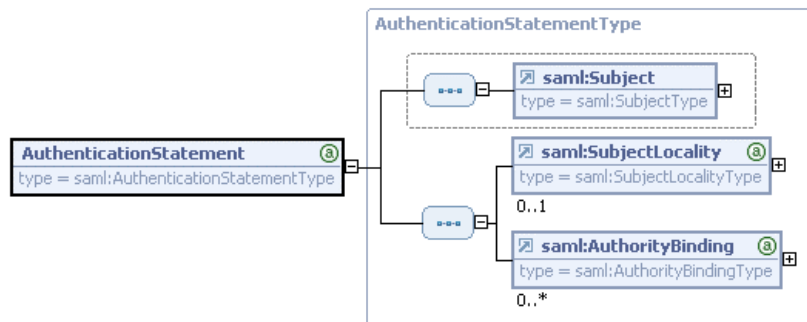


Figure 13: SAML AuthenticationStatement

Other characteristics applying to the given subject can be expressed by SAML's AttributeStatements.

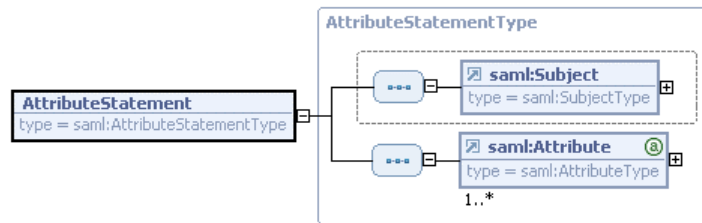


Figure 14: SAML AttributeStatement

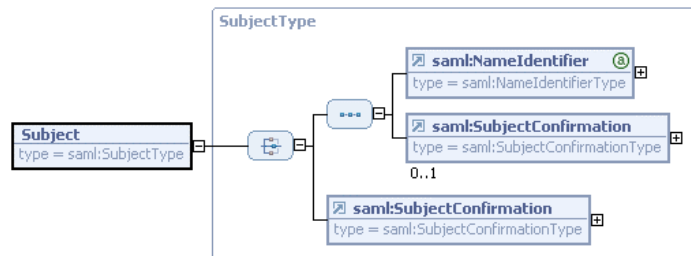


Figure 15: SAML Subject

The integrity of identity tokens handled by various parties can be assured by XML Digital signature [XMLSig], which is applied to an Assertion artifact containing one or more Statements. If the identity provider is known to the service provider then the integrity of the identity information contained in an identity token can be verified.

9.2 License Reference Tokens in OWS4

OWS-4 GeoDRM use and communicate license tokens in the form of a reference to the real "license" (therefore named "license reference token"). That way a license reference token as defined here is only a pointer, that is not valid without resolving the complete license token.

The following information is conveyed by the License Reference Token:

- URL for License Manager: A URL that points to where the license is stored (license manager)
- ID: A string that uniquely identifies the a license knowing the license manager

9.2.1 License Reference Tokens encoding

A License Reference Token is encoded as a SAML Assertion containing an Attribute Statement which in turn contains the following two attributes (both attributes are in the namespace <http://www.opengeospatial.org/schemas/ows4>):

- urn:opengeospatial:ows4:geodrm:licenseManagerURL
- urn:opengeospatial:ows4:geodrm:licenseID

```

<Assertion xmlns="urn:oasis:names:tc:SAML:1.0:assertion"
  xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
  xmlns:samlp="urn:oasis:names:tc:SAML:1.0:protocol"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  AssertionID="_6e6a4ef27ee135a97b4f0843a1317c7d"
  IssueInstant="2007-01-15T17:29:14.968Z"
  Issuer="http://iisdemo.informatik.unibw-
muenchen.de/ows4/DummyLicenseManager"
  MajorVersion="1" MinorVersion="1">
  <Conditions NotBefore="2007-01-15T17:29:14.968Z"
    NotOnOrAfter="2007-02-07T21:02:34.968Z">
  </Conditions>
  <AttributeStatement>
    <Subject>
      <SubjectConfirmation>
        <ConfirmationMethod>
          urn:oasis:names:tc:SAML:1.0:cm:sender-vouches
        </ConfirmationMethod>
      </SubjectConfirmation>
    </Subject>
    <Attribute
      AttributeName="urn:opengeospatial:ows4:geodrm:licenseManagerURL"
      AttributeNamespace="http://www.opengeospatial.org/schmas/ows4">
      <AttributeValue>
        http://iisdemo.informatik.unibw-muenchen.de/ows4/DummyLicenseManager
      </AttributeValue>
    </Attribute>
    <Attribute
      AttributeName="urn:opengeospatial:ows4:geodrm:licenseID"
      AttributeNamespace="http://www.opengeospatial.org/schmas/ows4">
      <AttributeValue>ID_LICENSE_3</AttributeValue>
    </Attribute>
  </AttributeStatement>
</Assertion>

```

As seen in the example, because it is a SAML assertion, the following information is also contained by the token:

- Issuing instance
- Validity period (see conditions)
 - Not Before
 - Not After (expiration)

Remark

All implementations in OWS4 use SAML 1.1 (and not 2.0). There are minor differences between 1.1 and 2.0. The group chose 1.1 because it is better supported in software libraries / packages. One of the used libraries is OpenSAML from Internet2:

<http://www.opensaml.org/>

9.2.2 Acquiring a License Reference Token

A License Reference Token can be acquired from the License Broker.

9.2.3 Attaching a License Reference Token to a SOAP Message

As mentioned, License Reference Tokens are implemented as SAML Assertions. As such, a License Reference Token is attached to a SOAP message as described in the SAML Token Profile of WS-Security ([SAMLWSS]).

The following shows a SOAP message containing a WFS Delete Transaction. The SOAP header contains a License Reference Token (in bold and italics) and an username identity token.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  <SOAP-ENV:Header>
    <wsse:Security
      xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-wssecurity-secext-1.0.xsd">
      <Assertion xmlns="urn:oasis:names:tc:SAML:1.0:assertion"
        xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
        xmlns:samlp="urn:oasis:names:tc:SAML:1.0:protocol"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        AssertionID="_8dbf0b675e095567bb6e2e8fc81144fc"
        IssueInstant="2006-10-31T10:52:53.703Z"
        Issuer="http://iisdemo.informatik.unibw-
muenchen.de/ows4/DummyLicenseManager"
        MajorVersion="1" MinorVersion="1">
        <Conditions NotBefore="2006-10-31T10:52:53.703Z"
          NotOnOrAfter="2006-12-18T00:39:33.703Z" />
        <AttributeStatement>
          <Subject>
            <SubjectConfirmation>
              <ConfirmationMethod
                urn:oasis:names:tc:SAML:1.0:cm:sender-vouches
              </ConfirmationMethod>
            </SubjectConfirmation>
          </Subject>
          <Attribute
            AttributeName="urn:opengeospatial:ows4:geodrm:licenseManagerURL"
            AttributeNamespace="http://www.opengeospatial.org/schemas/ows4">
            <AttributeValue>
              http://iisdemo.informatik.unibw-muenchen.de/ows4/DummyLicenseManager
            </AttributeValue>
          </Attribute>
        </Assertion>
      </wsse:Security>
    </SOAP-ENV:Header>
  </SOAP-ENV:Envelope>
```

```

        <Attribute
          AttributeName="urn:opengeospatial:ows4:geodrm:licenseID"
          AttributeNamespace="http://www.opengeospatial.org/schmas/ows4">
          <AttributeValue>ID_LICENSE_1</AttributeValue>
        </Attribute>
      </AttributeStatement>
    </Assertion>
    <wsse:UsernameToken
      xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-wssecurity-utility-1.0.xsd"
      wsu:Id="UsernameToken-3823725"
      xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-wssecurity-secext-1.0.xsd">
      <wsse:Username
        xmlns:wsse="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
        NGA-Officer
      </wsse:Username>
      <wsse:Password
        Type="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-username-token-profile-1.0#PasswordDigest"
        xmlns:wsse="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
        FMGWWN92e3lFA/3i22A3YzcCYnQ=
      </wsse:Password>
      <wsse:Nonce
        xmlns:wsse="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
        NCBh47YmJSzVuWm0MaRJDg==
      </wsse:Nonce>
      <wsu:Created
        xmlns:wsu="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
        2007-01-15T17:44:54.875Z
      </wsu:Created>
    </wsse:UsernameToken>
  </wsse:Security>

</SOAP-ENV:Header>
<SOAP-ENV:Body>
  <wfs:Transaction xmlns:wfs="http://www.opengis.net/wfs"
    xmlns="http://www.opengis.net/wfs"
    xmlns:gml="http://www.opengis.net/gml"
    xmlns:ogc="http://www.opengis.net/ogc"
    xmlns:tiger="http://www.census.gov"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    service="WFS"
    version="1.0.0"
    xsi:schemaLocation="http://www.census.gov
http://blade06.informatik.unibw-
muenchen.de:8080/geoserver/wfs/DescribeFeatureType?typeName=tiger:poi
http://www.opengis.net/wfs http://schemas.opengis.net/wfs/1.0.0/WFS-
transaction.xsd">
    <wfs:Delete typeName="tiger:poi">
      <ogc:Filter>
        <ogc:PropertyIsEqualTo>
          <ogc:PropertyName>tiger:NAME</ogc:PropertyName>
          <ogc:Literal>AM</ogc:Literal>
        </ogc:PropertyIsEqualTo>
      </ogc:Filter>
    </wfs:Delete>
  </SOAP-ENV:Body>

```

```

        </ogc:PropertyIsEqualTo>
    </ogc:Filter>
</wfs:Delete>
</wfs:Transaction>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
    
```

9.3 Licenses

During OWS4 licenses were encoded as XACML policies. For license tokens the group chose the “pointer-like” license tokens because of simplicity of implementation.

Note that in large information found here is the same with the one found in [OWS4EngVP].

9.3.1 License Model [CO3]

The following diagram provides an overview of the logical model of a license as used in the OWS-4 GeoDRM initiative:

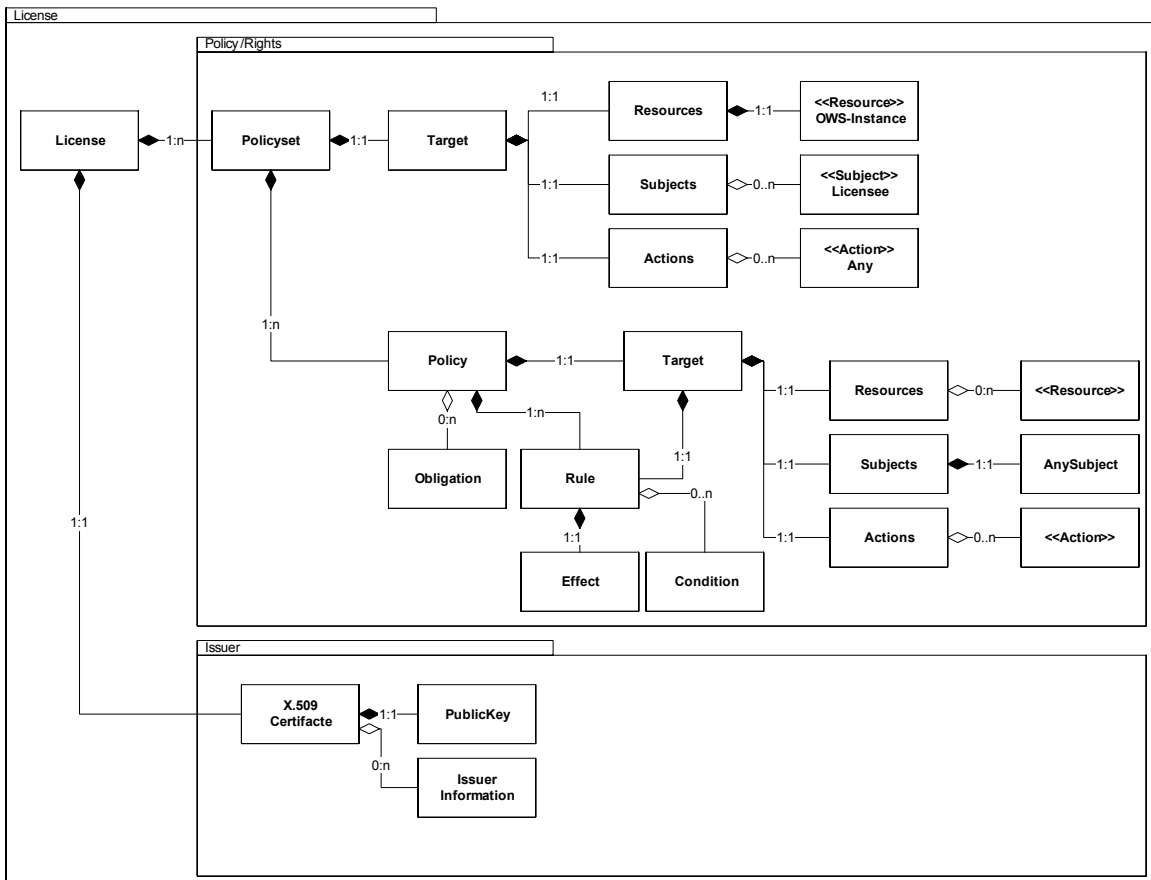


Figure 16: License as used in OWS-4

Licenses consist of one or more rights assigned to a user (licensee) and a signature that represents the license issuer.

A license contains at least one PolicySet. This PolicySet is a container of policies that represent the rights the license include and therefore the licensee owns. The policies/rights used in OWS-4 are always positive (~grants). The authorization side uses a “positive, closed” decision making pattern which is expected to be the most secure one. Only positive policies are used to avoid conflicting policies which could lead to security leaks and a granting permission is only given if the positive decision is non-ambiguous. All “not applicable” decisions are interpreted as “deny”.

Each policy may apply to a combination of resources and actions while the subject is usually defined on the root of the PolicySet as the subject is expected to the licensee. The licensee could be expressed in various ways including single human entities, groups, roles or indirect entities like sessions or IP-addresses.

Resource and actions *can* refer to the requests and resources that a particular OWS service offers. E.g. a WMS may offer a resource “layer” and a WFS a resource “featuretype”. Actions may accordingly be “GetMap” or “GetFeature”. We made good experience with including the whole service instance and type as a default resource for licenses applying the OWS services. Beside those resources and actions that could be derived from the service interface specification, there are truly other actions and resources. Problem with those definitions is that they are difficult to interpret when sharing licenses among multiple decisions points. On the other hand, defining a fixed list of actions/resources that could be used for grants within policies may be too restrictive and inflexible.

In any case we propose to define and use strong typed subject, actions and resources that should include a non-ambiguous designator for the type and the instance (maybe even the data type). For example:

Type	Instance	
OGC:WMS	http://SERVER/wms?	Resource
OGC:WMS:Request	GetMap	Action
Subject:Identifier	Persons_name	Subject

Types could be defined as Uniform Resource Names (URN).

Each policy may have conditions that have to be met in order to “get the grant”. Various types of conditions could be realized by modifying the policies that are part of the license. For example, the IP matching was formulated as equals-condition within the policy. Other conditions that could be checked during the authorization process could be time, date or other information that could be derived from the interaction context (scale, extent, etc.)

Other tests included an interesting feature of the used XACML encoding that enabled it to formulate obligations that are bound to a grant and that have to be performed by the enforcement point (Gatekeeper). Such obligations could be used to express pre- or post-processing instructions outside the gatekeeper. Like reduction of the quality of map images for guest accounts, clipping of certain areas, filtering of sensitive attributes etc.

According to use case #4 (Feature Updater), it is important to enforce restrictions based on the location, resp. geometry of features. In order to accommodate this, the UniBW provides a GeoXACML encoding for the PolicySet, according to [GeoXACML]. The main difference of such a policy encoding is that a XACML Condition element can be constructed by using Spatial Attributes and Spatial Functions for testing topological relations between geometries. Because a GeoXACML Policy encoding is compliant with the XACML schemas, it can use the same identical XML encoding, as described in the following section. An example of a GeoXACML Policy and a spatial Condition can be found in Appendix A.

9.3.1.1 License encoding

The licenses used in OWS-4 GeoDRM contain the following information:

- **id**: unique character sequence that identifies a license instance
- **policy set**: a set of policies describing permissions/denials attached to the license
- **signature**: a digital signature that ensures the integrity of the license and contains license issuer information (at least the name of issuer)

Licenses were encoded using the schema denoted in Figure 17. The XML schema of can be found in [OWS4EngVP].

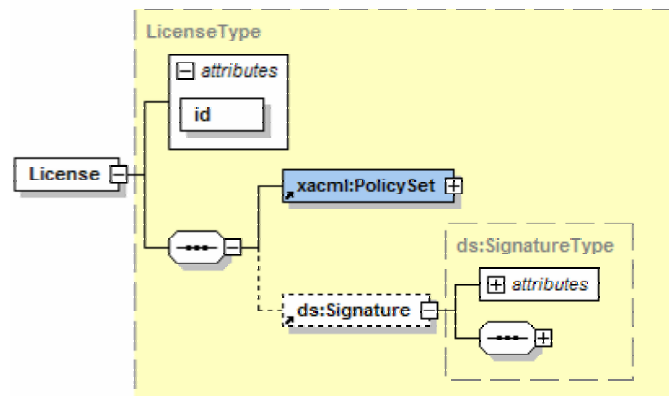


Figure 17: License encoding schema

The License id attribute may be of any type but has to be unique in the application system. The policy set shall be encoded using the eXtensible Access Control Markup Language (XACML), version 1.1 [XACML]. The <License> element shall be signed using “SOAP Security Extensions: Digital Signature” [XMLSig].

In OWS-4 GeoDRM rights/grants are encoded as XACML, but other REL languages could be applied as well.

Problem with a GeoDRM architecture dealing with different REL encodings is the encoding dependency between:

- License Broker and Manager for license creation and modification
- License Manager and Authorization Service for license resolving
- Gatekeeper and Authorization Service for decision making

In OWS-4 we tried to face those problems by using a common container structure for the license that contains the policies as a block and a reference token to be communicated. That way the License Broker is enabled to create a license in the REL it supports, pack it into a common license object which is transmitted and save in the manager using the manager's interface (which is not depended on any REL context definition language). Same counts for resolving licenses.

The latter problem remains; the dependency is at hand: a decision making component like the authorization service need to support the REL implementation(s). And the gatekeeper needs to know what input is required by the decision making component (this is usually not only depended on REL but to make it worse on the decision making capabilities).

The Licence Issuer is represented as X.509 Certificate. This certificate includes sufficient information about the issuer. In addition it will be used to sign the rights parts of the licence to ensure integrity and enables trust between license-provider and consumer.

Future requirements may need to use another encoding for the issuer as its information is limited (e.g. if issuer lineage information is needed).

9.4 The License Broker / Manager in OWS4

During OWS-4 the License Broker and the License Manager were seen as two different services. The License Broker is concerned with the issuing of licenses while the License Manager is concerned with storing the licenses and updating them. The License Manager also supports the retrieval of XACML Policies for License Reference Tokens.

During OWS-4, little effort was invested in the License Broker. This component was not defined as a service. There were two implementations (con terra and UniBW) both of them developed for testing purposes.

The License Manager was implemented during OWS-4 by con terra.

Note that parts of this section can also be found in [OWS4EngVP].

9.4.1 License Broker Service

This component was not defined as a service. In the two implementations in OWS-4, the License Broker was implemented as an HTML website. The user would log in, fill in some information and get a License Reference Token in the end.

9.4.2 License Manager Service:

It is the task of the License Manager Service to manage licenses. This management includes discovery and transactional functionalities for licenses. It may be

- Permanent and persistent management (for permanent licenses) or
- Transient management (for such licenses that have a time-based validity; e.g. session).
- The License Manager Service provides licenses to other components of the GeoDRM architecture. Especially the Authorization Service uses the License Manager Service to retrieve applicable licenses for a request or a license reference token for decision making.


The License Manager Service therefore serves the role of a policy administration point (PAP). It works closely together with

- License Broker Service: Negotiated transient and persistent licenses are stored at the License Manager Service.
- Authorization Service: During access control, the Authorization Service will provide the Gatekeeper with a decision. Beside the service request/response provided by the Gatekeeper, the Authorization Service will ask the License Manager Service for the appropriate XACML Policy.

The interface of the License Manager is logically divided into a discovery and a transaction interface.

- Transaction: methods needed to create, update and delete policies.
- Discovery: methods to find license, that match a request or a license token.

The discovery interface includes at a minimum a method to find a license token that matches to a license-token-reference. Other methods may be introduced to provide more sophisticated discovery functionalities (e.g. give all licenses that are applicable to a specific resource/action/subject etc.).

 Interface ^[e4]
<ul style="list-style-type: none"> - GetLicense <ul style="list-style-type: none"> ▪ Description: retrieve an existing license ▪ Parameters: <ul style="list-style-type: none"> ▪ one of <ul style="list-style-type: none"> • license id • license reference (embedded into a SAML Assertion) ▪ Returns: GetLicenseResponse element, containing the complete requested license if it exists - CreateLicense <ul style="list-style-type: none"> ▪ Description: Store a license and make it accessible ▪ Parameter: a single license document ▪ Returns: CreateLicenseResponse element containing the operation result, i.e. success or failure (e.g. due to duplicate id, ...)

- ReplaceLicense
 - Description: replaces an existing license
 - Parameter: a single license document that has the id of an existing license
 - Returns: ReplaceLicenseResponse element containing the operation result, i.e. success or failure (e.g. due to not existing id, ...)
- DeleteLicense
 - Description: removes an existing license
 - Parameter: id of the license to be deleted
 - Returns: DeleteLicenseResponse element containing the operation result, i.e. success or failure (e.g. due to not existing id, ...)
- CreateLicenseReference
 - Description: creates a unique, non-permanent reference to a licence
 - Parameters: license id and expiration time
 - Returns: CreateLicenseReferenceResponse element containing the reference id (embedded into a SAML Assertion)

10 Future Work

The purpose of this chapter is to present a list of issues that should make the object of investigation of future OGC test beds. The list is also targeted to the relevant groups within OGC (GeoDRM WG, Security WG).

The following items may be subject to further refinement and investigation:

1. A further investigation of the WS-Trust specification and how this can be applied to OGC Web Services and GeoDRM;
2. The definition of a service interface based on WS-Trust for the License Manager / License Broker. This would make the License Manger / License Broker a Security Token Service (STS) as defined in the WS-Trust specification. This makes sense because the License Broker interface has not been investigated in OWS-4, while the License Manager interface was defined without taking into consideration WS-Trust;
3. Implementation of the above mentioned service based on off-the-shelf products that support WS-Trust;
4. WS-Trust also supports a negotiation framework. Using this framework clients and services can dynamically negotiate a license needed for a request. Furthermore, the same framework can be used when acquiring a license token from the License Broker. This mechanism should be investigated and demonstrated by means of a prototype implementation;
5. Further investigation of cascading scenarios involving GeoDRM licenses and authentication of entities (this is described in section 7.4.3);
6. Implementation of license tokens based on the SAML profile of XACML ([SAMLXACML]).

Bibliography

- [WS-Trust] Web Services Trust Language, February 2005, online at <http://specs.xmlsoap.org/ws/2005/02/trust/WS-Trust.pdf>
- [WS-Security] OASIS Web Services Security, Version 1.1, February 2006, online at <http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf>
- [SAMLWSS] SAML Token Profile, Version 1.1, February 2006, online at <http://www.oasis-open.org/committees/download.php/16768/wss-v1.1-spec-os-SAMLTokenProfile.pdf>
- [XACML] XACML Version 2.0, February 2005, online at: <http://docs.oasis-open.org/xacml/2.0/XACML-2.0-OS-NORMATIVE.zip>
- [SAMLXACML] SAML 2.0 Profile of XACML 2.0, February 2005, online at http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-saml-profile-spec-os.pdf
- [WS-Architecture] W3C Web Services Architecture, February 2004, online at <http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/>
- [Topic12] OpenGIS Abstract Specification Topic 12: OpenGIS Service Architecture, same as ISO 19119, online at http://portal.opengeospatial.org/files/?artifact_id=1221
- [OWS4EngVP] OWS-4 IPR: GeoDRM Engineering Viewpoint and Supporting Architecture in OWS4, OGC #06-184
- [GeoDRM RM] The GeoDRM Reference Model, February 2006, OGC #06-004r4
- [GeoXACML] OpenGIS Discussion Paper: GeoXACML, a spatial extension to XACML, June 2005, OGC #05-036, online at https://portal.opengeospatial.org/files/index.php?artifact_id=10471

Annex A

A.1 Spatial Condition Example of a GeoXACML PolicySet

The following Condition element defines a test condition for the topological relation WITHIN between a feature's geometry, as it can be obtained by the following XPath expression and the Condition geometry, as highlighted in the example.

```
<Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:all-of">
  <Function
    FunctionId="http://www.geoxacml.org/1.0/function#within"/>
  <AttributeValue DataType="http://www.opengis.net/gml#polygon">
    <gml:Polygon xmlns:gml="http://www.opengis.net/gml" gid="P2"
      srsName="EPSG:4326">
      <gml:outerBoundaryIs>
        <gml:LinearRing>
          <gml:coordinates cs="," ts=" ">-
            74.28798767828596,40.72400955310945 -
            74.12552621736093,40.722605998371435 -
            74.12552621736093,40.614883172228936 -
            74.28939123302396,40.61558494959794 -
            74.28798767828596,40.72400955310945 -
            74.28798767828596,40.72400955310945 -
            74.28798767828596,40.72400955310945</gml:coordinates>
          </gml:LinearRing>
        </gml:outerBoundaryIs>
      </gml:Polygon>
    </AttributeValue>
    <AttributeSelector DataType="http://www.opengis.net/gml#box"
      MustBePresent="false" RequestContextPath="//ogc:BBOX/gml:Box"/>
  </Condition>
```

A.2 Example of a GeoXACML PolicySet

The following GeoXACML Policy describes the licensed rights of the “Field-Engineer” as used in the OWS-4 demo scenario for use case #4:

- Field-Engineer can request features of type ows4:Aerodrome_A, ows4:Taxiway_A, ows4:Aircraft_Hangar_A, ows4:Runway_A, ows4:Apron_A
- Field-Engineer can request features of type ows4:HeliPad_P2 in the area around the airport (the corresponding spatial condition is shown above)
- Field-Engineer can Update features of type ows4:HeliPad_P2, ows4:Taxiway_A
- Field-Engineer can Insert features of type ows4:HeliPad_P2 WITHIN area around the airport

```
<?xml version="1.0" encoding="UTF-8"?>
<PolicySet xmlns="urn:oasis:names:tc:xacml:1.0:policy"
  xmlns:xacml="urn:oasis:names:tc:xacml:1.0:policy" xmlns:xacml-
  context="urn:oasis:names:tc:xacml:1.0:context"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  PolicyCombiningAlgId="urn:oasis:names:tc:xacml:1.0:policy-
  combining-algorithm:first-applicable" PolicySetId="LICENSE_ID_2"
  xsi:schemaLocation="urn:oasis:names:tc:xacml:1.0:policy cs-
```

```

xacml-schema-policy-01.xsd">
  <Description>This PolicySet represents the granted rights to
  Field-Engineer through LICENSE_ID_2</Description>
  <Target>
    <Subjects>
      <AnySubject/>
    </Subjects>
    <Resources>
      <Resource>
        <ResourceMatch
  MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue
  DataType="http://www.w3.org/2001/XMLSchema#string">ID_LICENSE_2<
  /AttributeValue>
          <ResourceAttributeDesignator
  AttributeId="http://www.geoxacml.org/1.0/resource#license-id"
  DataType="http://www.w3.org/2001/XMLSchema#string"/>
          </ResourceMatch>
        </Resource>
      </Resources>
    <Actions>
      <AnyAction/>
    </Actions>
  </Target>
  <Policy PolicyId="Field-Engineer"
  RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-
  algorithm:first-applicable"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <PolicyDefaults>
      <XPathVersion>http://www.w3.org/TR/1999/Rec-xpath-
  19991116</XPathVersion>
    </PolicyDefaults>
    <Target>
      <Subjects>
        <Subject>
          <SubjectMatch
  MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
            <AttributeValue
  DataType="http://www.w3.org/2001/XMLSchema#string">Field-
  Engineer</AttributeValue>
            <SubjectAttributeDesignator
  AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
  DataType="http://www.w3.org/2001/XMLSchema#string"
  SubjectCategory="urn:oasis:names:tc:xacml:1.0:subject-
  category:access-subject"/>
            </SubjectMatch>
          </Subject>
        <Subject>
          <SubjectMatch
  MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
            <AttributeValue
  DataType="http://www.w3.org/2001/XMLSchema#string">CN=Bill Field
  Engineer, OU=INF3, O=UniBW, L=Munich, ST=Bavaria,
  C=DE</AttributeValue>
            <SubjectAttributeDesignator
  AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
  DataType="http://www.w3.org/2001/XMLSchema#string"
  SubjectCategory="urn:oasis:names:tc:xacml:1.0:subject-
  category:access-subject"/>
            </SubjectMatch>
          </Subject>
        </Subjects>
      </Resources>
    <Resource>
      <ResourceMatch
  MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
        <AttributeValue

```

```

    <AttributeValue>http://geoser
    ver.itc.nl:8080/geoserver/wfs</AttributeValue>
    <ResourceAttributeDesignator
    AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
    DataType="http://www.w3.org/2001/XMLSchema#anyURI"/>
    </ResourceMatch>
  </Resource>
</Resources>
<Actions>
  <AnyAction/>
</Actions>
</Target>
<Rule Effect="Permit" RuleId="rule-2.1">
  <Description>Field-Engineer can request features of type
  ows4:Aerodrome_A, ows4:Taxiway_A, ows4:Aircraft_Hangar_A,
  ows4:Runway_A, ows4:Apron_A</Description>
  <Target>
    <Subjects>
      <AnySubject/>
    </Subjects>
    <Resources>
      <Resource>
        <ResourceMatch
        MatchId="urn:oasis:names:tc:xacml:1.0:function:integer-less-
        than">
          <AttributeValue
          DataType="http://www.w3.org/2001/XMLSchema#integer">0</Attribute
          Value>
            <AttributeSelector
            DataType="http://www.w3.org/2001/XMLSchema#integer"
            RequestContextPath="count(//wfs:Query[@typeName='ows4:Aerodrome_
            A'])"/>
          </ResourceMatch>
        </Resource>
        <Resource>
          <ResourceMatch
          MatchId="urn:oasis:names:tc:xacml:1.0:function:integer-less-
          than">
            <AttributeValue
            DataType="http://www.w3.org/2001/XMLSchema#integer">0</Attribute
            Value>
              <AttributeSelector
              DataType="http://www.w3.org/2001/XMLSchema#integer"
              RequestContextPath="count(//wfs:Query[@typeName='ows4:Taxiway_A'
              ])/>
            </ResourceMatch>
          </Resource>
          <Resource>
            <ResourceMatch
            MatchId="urn:oasis:names:tc:xacml:1.0:function:integer-less-
            than">
              <AttributeValue
              DataType="http://www.w3.org/2001/XMLSchema#integer">0</Attribute
              Value>
                <AttributeSelector
                DataType="http://www.w3.org/2001/XMLSchema#integer"
                RequestContextPath="count(//wfs:Query[@typeName='ows4:Aircraft_H
                angar_A'])"/>
              </ResourceMatch>
            </Resource>
            <Resource>
              <ResourceMatch
              MatchId="urn:oasis:names:tc:xacml:1.0:function:integer-less-
              than">
                <AttributeValue
                DataType="http://www.w3.org/2001/XMLSchema#integer">0</Attribute
                Value>

```



```

        <AttributeSelector
DataType="http://www.w3.org/2001/XMLSchema#integer"
RequestContextPath="count(//wfs:Query[@typeName='ows4:Apron_A'])"
"/>
    </ResourceMatch>
</Resource>
<Resource>
    <ResourceMatch
MatchId="urn:oasis:names:tc:xacml:1.0:function:integer-less-than">
        <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#integer">0</AttributeValue>
        <AttributeSelector
DataType="http://www.w3.org/2001/XMLSchema#integer"
RequestContextPath="count(//wfs:Query[@typeName='ows4:Runway_A'])"
"/>
    </ResourceMatch>
</Resource>
<Resource>
    <ResourceMatch
MatchId="urn:oasis:names:tc:xacml:1.0:function:integer-less-than">
        <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#integer">0</AttributeValue>
        <AttributeSelector
DataType="http://www.w3.org/2001/XMLSchema#integer"
RequestContextPath="count(//wfs:Query[@typeName='ows4:Administrative_Boundary_L'])"
"/>
    </ResourceMatch>
</Resource>
</Resources>
<Actions>
    <Action>
        <ActionMatch
MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
            <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">GetFeature</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>
</Rule>
<Rule Effect="Permit" RuleId="rule-2.2">
    <Description>Field-Engineer can request features of type
ows4:HeliPad_P2 in the area around the airport</Description>
    <Target>
        <Subjects>
            <AnySubject/>
        </Subjects>
        <Resources>
            <Resource>
                <ResourceMatch
MatchId="urn:oasis:names:tc:xacml:1.0:function:integer-less-than">
                    <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#integer">0</AttributeValue>
                    <AttributeSelector
DataType="http://www.w3.org/2001/XMLSchema#integer"
RequestContextPath="count(//wfs:Query[@typeName='ows4:HeliPad_P2'])"
"/>

```

```

        </ResourceMatch>
      </Resource>
    </Resources>
  <Actions>
    <Action>
      <ActionMatch
MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">GetFeature</A
tttributeValue>
          <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
FunctionId="urn:oasis:names:tc:xacml:1.0:function:all-of">
    <Function
FunctionId="http://www.geoxacml.org/1.0/function#within"/>
    <AttributeValue
DataType="http://www.opengis.net/gml#polygon">
      <gml:Polygon
xmlns:gml="http://www.opengis.net/gml" gid="P2"
srsName="EPSG:4326">
        <gml:outerBoundaryIs>
          <gml:LinearRing>
            <gml:coordinates cs="," ts=" "-
74.28798767828596,40.72400955310945 -
74.12552621736093,40.722605998371435 -
74.12552621736093,40.614883172228936 -
74.28939123302396,40.61558494959794 -
74.28798767828596,40.72400955310945 -
74.28798767828596,40.72400955310945 -
74.28798767828596,40.72400955310945</gml:coordinates>
          </gml:LinearRing>
        </gml:outerBoundaryIs>
      </gml:Polygon>
    </AttributeValue>
    <AttributeSelector
DataType="http://www.opengis.net/gml#box" MustBePresent="false"
RequestContextPath="//ogc:BBOX/gml:Box"/>
  </Condition>
</Rule>
<Rule Effect="Permit" RuleId="rule-2.3">
  <Description>Field-Engineer can Update features of type
ows4:HeliPad_P2, ows4:Taxiway_A</Description>
  <Target>
    <Subjects>
      <AnySubject/>
    </Subjects>
    <Resources>
      <Resource>
        <ResourceMatch
MatchId="urn:oasis:names:tc:xacml:1.0:function:integer-less-
than">
          <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#integer">0</Attribute
Value>
            <AttributeSelector
DataType="http://www.w3.org/2001/XMLSchema#integer"
RequestContextPath="count(//wfs:Update[@typeName='ows4:HeliPad_P
2'])"/>
          </ResourceMatch>
        </Resource>
      </Resource>
    </Resources>
  </Target>
</Rule>

```

```

        <ResourceMatch
MatchId="urn:oasis:names:tc:xacml:1.0:function:integer-less-
than">
            <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#integer">0</Attribute
Value>
            <AttributeSelector
DataType="http://www.w3.org/2001/XMLSchema#integer"
RequestContextPath="count(//wfs:Update[@typeName='ows4:Taxiway_A
'])"/>
        </ResourceMatch>
    </Resource>
</Resources>
<Actions>
    <Action>
        <ActionMatch
MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
            <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">Update</Attri
buteValue>
            <ActionAttributeDesignator
AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </ActionMatch>
    </Action>
</Actions>
</Target>
</Rule>
<Rule Effect="Permit" RuleId="rule-2.4">
    <Description>Field-Engineer can Insert features of type
ows4:HeliPad_P2 WITHIN area around the airport</Description>
    <Target>
        <Subjects>
            <AnySubject/>
        </Subjects>
        <Resources>
            <Resource>
                <ResourceMatch
MatchId="urn:oasis:names:tc:xacml:1.0:function:integer-less-
than">
                    <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#integer">0</Attribute
Value>
                    <AttributeSelector
DataType="http://www.w3.org/2001/XMLSchema#integer"
RequestContextPath="count(//wfs:Insert/ows4:HeliPad_P2)"/>
                </ResourceMatch>
            </Resource>
        </Resources>
        <Actions>
            <Action>
                <ActionMatch
MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
                    <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">Insert</Attri
buteValue>
                    <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
FunctionId="urn:oasis:names:tc:xacml:1.0:function:all-of">
        <Function
FunctionId="http://www.geoxacml.org/1.0/function#within"/>

```

```

        <AttributeValue
DataType="http://www.opengis.net/gml#polygon">
        <gml:Polygon
xmlns:gml="http://www.opengis.net/gml" gid="P2"
srsName="EPSG:4326">
            <gml:outerBoundaryIs>
                <gml:LinearRing>
                    <gml:coordinates cs="," ts=" "-
74.28798767828596,40.72400955310945 -
74.12552621736093,40.722605998371435 -
74.12552621736093,40.614883172228936 -
74.28939123302396,40.61558494959794 -
74.28798767828596,40.72400955310945 -
74.28798767828596,40.72400955310945 -
74.28798767828596,40.72400955310945</gml:coordinates>
                </gml:LinearRing>
            </gml:outerBoundaryIs>
        </gml:Polygon>
    </AttributeValue>
    <AttributeSelector
DataType="http://www.opengis.net/gml#point" MustBePresent="true"
RequestContextPath="//wfs:Transaction/wfs:Insert/ows4:HeliPad_P2
/ows4:the_geom/gml:Point"/>
    </Condition>
</Rule>
<Rule Effect="Deny" RuleId="rule-2.5">
    <Description>Field-Engineer can NOT delete features of
type ows4:HeliPad_P2, ows4:Taxiway_A,
ows4:Runway_A</Description>
    <Target>
        <Subjects>
            <AnySubject/>
        </Subjects>
        <Resources>
            <Resource>
                <ResourceMatch
MatchId="urn:oasis:names:tc:xacml:1.0:function:integer-less-
than">
                    <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#integer">0</Attribute
Value>
                    <AttributeSelector
DataType="http://www.w3.org/2001/XMLSchema#integer"
RequestContextPath="count(//wfs:Delete[@typeName='ows4:HeliPad_P
2'])"/>
                </ResourceMatch>
            </Resource>
            <Resource>
                <ResourceMatch
MatchId="urn:oasis:names:tc:xacml:1.0:function:integer-less-
than">
                    <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#integer">0</Attribute
Value>
                    <AttributeSelector
DataType="http://www.w3.org/2001/XMLSchema#integer"
RequestContextPath="count(//wfs:Delete[@typeName='ows4:Taxiway_A
'])"/>
                </ResourceMatch>
            </Resource>
            <Resource>
                <ResourceMatch
MatchId="urn:oasis:names:tc:xacml:1.0:function:integer-less-
than">
                    <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#integer">0</Attribute
Value>

```

```

        <AttributeSelector
      DataType="http://www.w3.org/2001/XMLSchema#integer"
      RequestContextPath="count(/wfs:Delete[@typeName='ows4:Runway_A'
    ])" />
      </ResourceMatch>
    </Resource>
  </Resources>
<Actions>
  <Action>
    <ActionMatch
      MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <AttributeValue
          DataType="http://www.w3.org/2001/XMLSchema#string">Delete</Attri
        buteValue>
        <ActionAttributeDesignator
          AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
          DataType="http://www.w3.org/2001/XMLSchema#string" />
        </ActionMatch>
      </Action>
    </Actions>
  </Target>
</Rule>
  <Rule Effect="Deny" RuleId="DenyAll" />
</Policy>
</PolicySet>

```