

Oracle® Fusion Middleware

Understanding Oracle Web Services Manager

12c (12.1.2)

E28242-01

June 2013

Documentation for developers and administrators that introduces features of the Oracle Web Services Manager (OWSM) policy framework and basic security concepts.

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate failsafe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

Contents

Preface	vii
Audience.....	vii
Documentation Accessibility	vii
Related Documents	vii
Conventions	viii
What's New in This Guide	ix
New and Changed Features for 12c (12.1.2).....	ix
Other Significant Changes in this Book	ix
1 Introducing Oracle Web Services Manager	
1.1 OWSM Overview	1-1
1.2 OWSM Features	1-2
1.3 OWSM Architecture	1-3
2 Understanding the OWSM Policy Framework	
2.1 Overview of OWSM Policy Framework	2-1
2.1.1 OWSM Policy Framework Components	2-2
2.1.2 OWSM Agent and Policy Manager Interaction	2-2
2.1.3 OWSM Agent and Policy Manager Characteristics	2-3
2.1.4 OWSM Agent and Policy Manager Request Flow	2-4
2.1.5 OWSM Configuration Artifacts	2-4
2.2 Understanding Policies	2-4
2.3 Building Policies Using Policy Assertions	2-7
2.3.1 Defining Multiple Policy Alternatives (OR Groups)	2-8
2.4 Understanding Policy Subjects	2-9
2.5 Attaching Policies to Policy Subjects	2-11
2.5.1 Direct Policy Attachment	2-11
2.5.2 Global Policy Attachments Using Policy Sets	2-11
2.5.2.1 Subject Types and Scope of Resources	2-12
2.5.2.2 Typical Uses for Global Policy Attachments	2-13
2.6 How Policies are Executed	2-13
2.7 OWSM Predefined Policies and Assertion Templates	2-15
2.8 Overriding Security Policy Configuration	2-15
2.9 Recommended Naming Conventions for Policies	2-15

3 Understanding Web Service Security Concepts

3.1	Overview of Web Service Security	3-2
3.1.1	Web Service Security Requirements	3-3
3.2	Understanding Transport-level and Application-level Security	3-3
3.3	Understanding Authentication	3-4
3.3.1	Digest Authentication	3-4
3.4	Understanding Authorization	3-5
3.5	Understanding Message Protection	3-5
3.5.1	Message Encryption	3-6
3.5.2	Message Signing (XML Signature)	3-7
3.6	Understanding Keys and Certificates	3-8
3.6.1	Overview of Private Keys and Certificates	3-8
3.6.2	How Different Security Policies Use Private Keys and Certificates	3-10
3.6.2.1	Message Protection Policy Types	3-10
3.6.2.1.1	SSL	3-10
3.6.2.1.2	wss11	3-10
3.6.2.1.3	wss10	3-11
3.6.2.2	Authentication Token Policy Types	3-12
3.6.2.2.1	Username Token	3-12
3.6.2.2.2	Kerberos Token	3-12
3.6.2.2.3	X.509 Certificate Token	3-12
3.6.2.2.4	SAML Sender Vouches Token	3-12
3.6.2.2.5	SAML Bearer and SAML HOK Tokens from an STS	3-13
3.6.3	How OWSM Locates Keystore And Key Passwords for the JKS Keystore	3-13
3.6.4	Setting Up Private Keys and Certificates for SSL Policies	3-14
3.6.5	Setting up Private Keys and Certificates for Message Protection Policies	3-15
3.7	How OWSM Uses the Credential Store	3-16
3.8	Understanding Security Policies	3-17
3.9	Understanding Security Tokens	3-18
3.9.1	Username Token	3-19
3.9.2	X.509 Certificate	3-19
3.9.3	Kerberos Token	3-19
3.9.4	SAML Token	3-19
3.10	Understanding Secure Attachments	3-21
3.11	Understanding Secure Conversation	3-21
3.11.1	Why Use WS-SecureConversation	3-22
3.11.1.1	Using WS-SecureConversation With WS-ReliableMessaging	3-23
3.11.2	WS-SecureConversation Architecture	3-23
3.11.3	When To Use WS-SecureConversation	3-25
3.11.4	When To Use Re-Authentication	3-26
3.11.5	Using Basic Mode Versus Advanced Mode	3-26
3.11.6	Persistence	3-27
3.11.6.1	Default Domain-Wide Persistence Implementation	3-27
3.11.6.2	Client- and Web Service-Specific Persistence Implementation	3-27
3.12	Understanding the Kerberos Protocol	3-28
3.12.1	Credential Delegation in Kerberos	3-29
3.12.2	Kerberos and SPNEGO	3-30

3.12.3	Kerberos and WS-SecureConversation Derived Keys	3-30
3.13	Understanding Web Services Addressing	3-31
3.14	Understanding Web Services Trust	3-31
3.15	Understanding Web Services ReliableMessaging	3-32

A Web Service Security Standards

Preface

This section describes the intended audience, how to use this guide, and provides information about documentation accessibility.

Audience

This guide is intended for:

- System and security administrators who administer Web services and manage security
- Application developers who are developing Web services and testing the security prior to deployment of the Web services
- Security architects who create security policies.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc>.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit

<http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info> or visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs> if you are hearing impaired.

Related Documents

For more information, see the following documents in the Oracle Fusion Middleware Web services documentation set:

- *Securing Web Services and Managing Policies with Oracle Web Services Manager*
- *Administering Web Services*
- *Understanding Web Services*
- *Developing Oracle Infrastructure Web Services*
- *Understanding WebLogic Web Services for Oracle WebLogic Server*

- *Developing Extensible Applications for Oracle Web Services Manager*
- *Interoperability Solutions Guide for Oracle Web Services Manager*
- *Developing JAX-WS Web Services for Oracle WebLogic Server*
- *Developing JAX-RPC Web Services for Oracle WebLogic Server*
- *Securing WebLogic Web Services for Oracle WebLogic Server*
- *WebLogic Web Services Reference for Oracle WebLogic Server*
- *Developing Fusion Web Applications with Oracle Application Development Framework*
- *Oracle Fusion Middleware Developer's Guide for Oracle WebCenter Portal*
- "Developing and Securing Web Services" in *Developing Applications with Oracle JDeveloper*

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
<i>italic</i>	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
monospace	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

What's New in This Guide

The following topics introduce the new and changed features of Oracle Web Services Manager (OWSM) and other significant changes that are described in this guide, and provides pointers to additional information.

New and Changed Features for 12c (12.1.2)

Oracle Fusion Middleware 12c (12.1.2) includes the following new and changed features for this document:

- Security enhancements, including:
 - Secure conversation, as described in "[Understanding Secure Conversation](#)" on page 3-21.
 - Kerberos security enhancements, including credential delegation and single sign-on using SPNEGO, and derived key support, as described in "[Understanding the Kerberos Protocol](#)" on page 3-28.
 - Digest authentication, as described in "[Digest Authentication](#)" on page 3-4.
 - Federated STS trust and token caching support, as described in "[Understanding Web Services Trust](#)" on page 3-31.
 - RESTful Web services security using Oracle Web Services Manager (OWSM) policies, as described in "[Understanding Policies](#)" on page 2-4.
- New policy categories, as described in "[Policy Categories](#)" on page 2-5.
- New and updated subject types, as described in "[Policy Subjects and Resource Scopes](#)" on page 2-9.

Other Significant Changes in this Book

For 12c (12.1.2), this guide has been updated in several ways. Following are the sections that have been added or changed.

- In this release, the *Security and Administrator's Guide for Web Services* delivered in Oracle Fusion Middleware 11g, has been split up into the following documents:
 - *Understanding Oracle Web Services Manager* (this document)
 - *Securing Web Services and Managing Policies with Oracle Web Services Manager*
 - *Administering Web Services*
- [Chapter 1, "Introducing Oracle Web Services Manager,"](#) has been updated to reflect OWSM 12c features and architecture.

- The policy interceptor pipelines illustrated in [Figure 2-5, "Policy Interceptors Acting on Messages Between a Client and Web Service \(SOAP\)"](#) have been updated to reflect the complete set of policy interceptors supported in 12c:
- Direct and global policy attachment descriptions have been enhanced, as described in ["Attaching Policies to Policy Subjects"](#) on page 2-11.

Introducing Oracle Web Services Manager

Oracle Web Services Manager (OWSM) provides a policy framework to manage and secure Web services consistently across your organization. It provides capabilities to build, enforce, run and monitor Web service policies, such as security, reliable messaging, MTOM, and addressing policies. OWSM can be used by both developers, at design time, and system administrators in production environments.

This chapter includes the following topics:

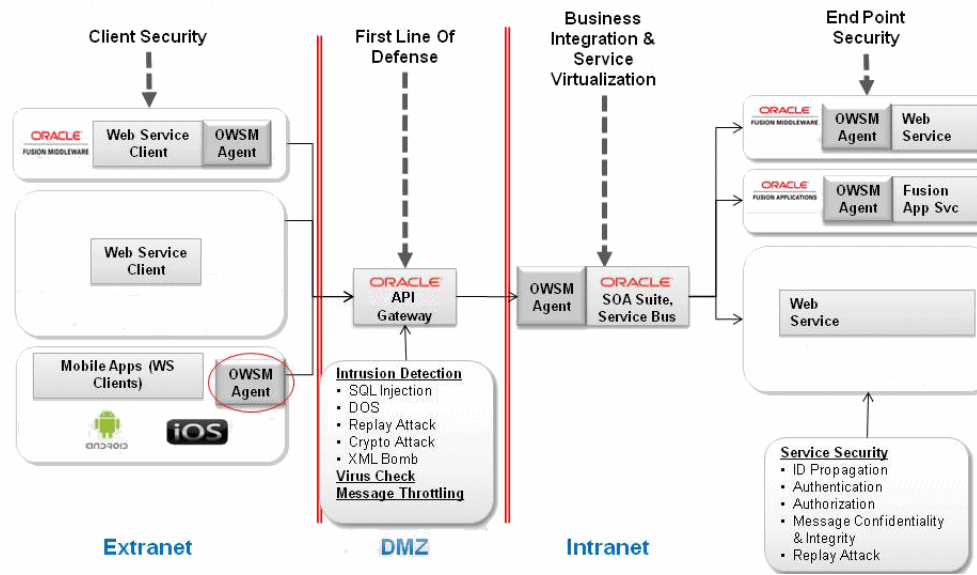
- [Section 1.1, "OWSM Overview"](#)
- [Section 1.2, "OWSM Features"](#)
- [Section 1.3, "OWSM Architecture"](#)

1.1 OWSM Overview

OWSM provides business agility to respond to security threats and security breaches by allowing policy changes to be enforced in real time without the need to interrupt the running business processes.

As shown in [Figure 1-1](#), OWSM provides the "first mile security" via client agents for securing Web service clients, and "last mile security" via server agents securing Web services. If your Web services are accessible only from inside the corporate intranet, they typically still require authentication and authorization. In addition, auditing is often required to address regulatory compliance.

Figure 1–1 Security Provided by OWSM Agents



OWSM allows for policy-driven centralized management of Web services with local enforcement. OWSM provides a policy framework to manage and secure Web services consistently across your organization.

The benefits of this policy driven approach include:

- Allows security to be declarative and externalized.
- Provides business agility to respond to security threats and security breaches by allowing policy changes to be enforced in real time without the need to interrupt the running business processes.
- Avoids the need for developers to understand security specifications and security implementation details.

OWSM allows you to:

- Centrally define and store declarative policies applied to the multiple Web services.
- Locally enforce policies through configurable agents.
- Monitor run time security events such as failed authentication or authorization.

You can use OWSM to secure the following categories of Oracle Web services:

- RESTful Web services
- Oracle Infrastructure Web services—ADF services
- WebLogic (Java EE) Web services

1.2 OWSM Features

OWSM includes an extensive array of policy and management features, including the following:

- Policy Management:
 - Global and direct policy attachment.

- Policy attachment at design time and post-deployment.
- Ability to attach/detach multiple policies to a Web service or client.
- Auto-select of client policies.
- Identity propagation across multiple Web services.
- Policy advertisement in WSDL.
- Monitoring/Management:
 - Centralized management, auditing and reporting.
 - Policy versioning and rollback.
 - Performance management, including metrics for service, port, and operation, policy dependencies per port, number of security violations, number of invocations, and more.
 - Policy export and import.
 - Policy impact analysis.
- Security standards supported:
 - A broad range of security standards is supported, as described in [Table A-1](#).
 - Pre-defined, reusable policies, including security, reliability, addressing, management and MTOM policies.
 - Custom policy extensions.

OWSM supports policy attachment at both design time and post-deployment, which provides capabilities for both developers and system administrators:

- Developers can attach OWSM policies from the Oracle JDeveloper context menu and property inspector. For more information, see "Developing and Securing Web Services" in *Developing Applications with Oracle JDeveloper*.
- System administrators can leverage OWSM through the Oracle Enterprise Manager Fusion Middleware Control and WLST. They can centrally define policies using the OWSM Policy Manager and enforce OWSM policies locally at run time.

Examples of specific tasks that you can perform using OWSM include the following:

- Handle WS-Security (for example, encryption, decryption, signing, signature validation, and so on).
- Define authentication and authorization policies against an LDAP directory.
- Generate standard security tokens (such as SAML tokens) to propagate identities across multiple Web services used in a single transaction.
- Segment policies into different namespaces by creating policies within different folders.
- Examine log files.

1.3 OWSM Architecture

[Figure 1-2](#) shows the main components of OWSM architecture.

Note: A subset of OWSM policies are supported for RESTful Web services, as described in Which OWSM Policies Are Supported for RESTful Web Services? in *Securing Web Services and Managing Policies with Oracle Web Services Manager*. The subset does not include all of the policy interceptor types shown in Figure 1-2.

Figure 1-2 Components of OWSM Architecture

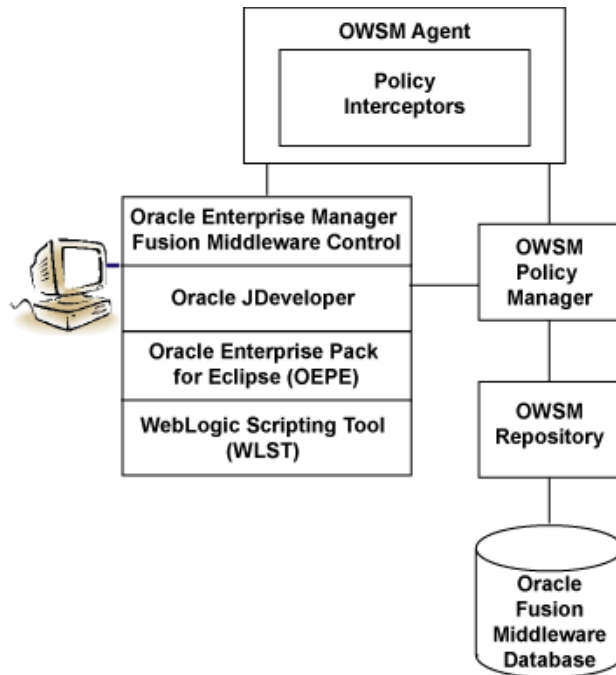


Table 1-1 describes the components of OWSM shown in Figure 1-2, and highlights their use in the figure.

Table 1-1 Components of OWSM Architecture

OWSM Component	Description
Oracle Enterprise Manager Fusion Middleware Control	Enables administrators to access OWSM's functionality to manage, secure, and monitor Web services.
Oracle JDeveloper	Provides a full-featured Java IDE that can be used for end-to-end development of Web services. Using visual and declarative tools, developers can build ADF and WebLogic Java EE Web services, automatically deploy them to an instance of Oracle WebLogic Server, and immediately test the running Web service. Alternatively, JDeveloper can be used to drive the creation of Web services from WSDL descriptions. JDeveloper is Ant-aware. You can use this tool to build and run Ant scripts for assembling the client and for assembling and deploying the service. For more information, see the Oracle JDeveloper online help. For information about installing JDeveloper, see <i>Installing Oracle JDeveloper</i> .

Table 1–1 (Cont.) Components of OWSM Architecture

OWSM Component	Description
WebLogic Scripting Tool (WSLT)	Enables administrators to view and configure Web services, and manage Web service policies from the command line. For more information, see <i>WLS Command Reference for WebLogic Server</i> .
OWSM Policy Manager	Reads/writes the policies, including predefined and custom policies from the OWSM Repository.
OWSM Agent	Manages the enforcement of policies via the Policy Interceptor Pipeline.
Policy Interceptors	Enforces policies. For more information, see " How Policies are Executed " on page 2-13.
OWSM Repository	Stores OWSM metadata, such as policies, policy sets, assertions templates, and policy usage data. The OWSM Repository is available as a database (for production use) or as files in the file system (for development use in JDeveloper).
Oracle Fusion Middleware Database	Provides database support for the OWSM Repository.

Subsequent chapters of this document describe conceptual information about the OWSM policy framework and security concepts. This document also includes a section on the security standards for Oracle Infrastructure Web Services.

The companion documents *Securing Web Services and Managing Policies with Oracle Web Services Manager* and *Administering Web Services* describe how to secure and administer Web services using OWSM, respectively.

Understanding the OWSM Policy Framework

This chapter describes the OWSM policy framework which manages and secures Web services consistently across your organization.

This chapter contains the following sections:

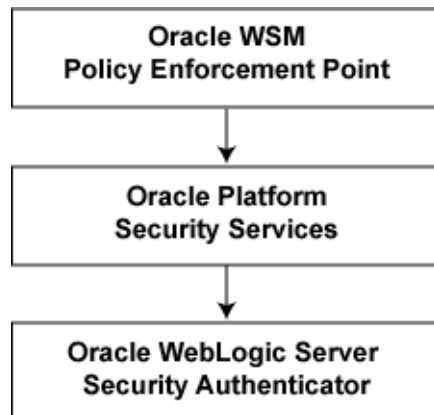
- [Section 2.1, "Overview of OWSM Policy Framework"](#)
- [Section 2.2, "Understanding Policies"](#)
- [Section 2.3, "Building Policies Using Policy Assertions"](#)
- [Section 2.4, "Understanding Policy Subjects"](#)
- [Section 2.5, "Attaching Policies to Policy Subjects"](#)
- [Section 2.6, "How Policies are Executed"](#)
- [Section 2.7, "OWSM Predefined Policies and Assertion Templates"](#)
- [Section 2.8, "Overriding Security Policy Configuration"](#)
- [Section 2.9, "Recommended Naming Conventions for Policies"](#)

2.1 Overview of OWSM Policy Framework

Oracle Web Services Manager (OWSM) provides a policy framework to manage and secure Web services consistently across your organization. It provides capabilities to build, enforce, run and monitor Web service policies, such as security, reliable messaging, MTOM, and addressing policies. OWSM can be used by both developers, at design time, and system administrators in production environments.

The policy framework is built using the WS-Policy standard. The OWSM Policy Enforcement Point (PEP) leverages Oracle Platform Security Service (OPSS) and the Oracle WebLogic Server authenticator for authentication and permission-based authorization, as shown in [Figure 2-1](#).

Figure 2–1 OWSM Policy Framework Leverages OPSS and Oracle WebLogic Server Security



2.1.1 OWSM Policy Framework Components

The OWSM Policy Framework consists of the following components:

- **Policy Manager** reads and writes policies including predefined and custom policies from the OWSM Repository. You can deploy the Policy Manager on separate Managed Servers.
- **Agent** is responsible for policy enforcement, execution and gathering of runtime statistics. The OWSM Agent is available on all Oracle Fusion Middleware Managed servers. It is configured on the same server as the application it protects.

The OWSM Agent is made up of a set of jar files, which are a part of underlying Web service stack. It does not have any session state. The Agent maintains an in-memory policy cache, which is populated at the Agent startup time. It does not use any JTA or JMS.

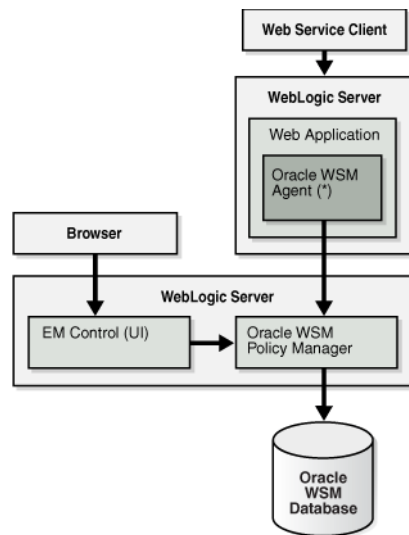
The OWSM Agent is made up of the following two pieces:

- **Policy Access Point (PAP)** communicates with Policy Manager. The Agent communicates with the Policy Manager through EJB invocations.
- **Policy Interceptor** is generated when a Web service is deployed and activated, or when a policy is attached to a Web service using Enterprise Manager. If new Web services are protected using OWSM, an additional instance of the interceptor is generated for each new Web service. Interceptor is responsible for policy enforcement.
- **OWSM Repository** Policies are stored in the OWSM Repository. It is typically backed by an Oracle database. For high availability purposes, Oracle recommends using an Oracle RAC database as the back end for OWSM Repository.
- **Enterprise Manager** is used to configure OWSM. It also displays different Web services metrics gathered by OWSM.

2.1.2 OWSM Agent and Policy Manager Interaction

Consider the high-level view of how the OWSM Agent and the OWSM Policy Manager interact, as shown in [Figure 2–2](#).

The OWSM Agent expects the OWSM Policy Manager to be deployed on at least one node of the domain.

Figure 2–2 OWSM Agent and Policy Manager Interaction

The Policy Manager is a stateless application which does not perform any caching. There is no special application level startup sequence performed when the Managed Server where the Policy Manager is deployed starts up. The Policy Manager communicates with the OWSM Repository to retrieve policies. The OWSM Repository can be stored in a database to provide MDS high availability.

The OWSM agent has an auto-discovery feature to locate and connect to an OWSM Policy Manager, and can also be configured explicitly to connect to a remote policy manager. See "Configuring OWSM Policy Access" in *Securing Web Services and Managing Policies with Oracle Web Services Manager* for additional information.

When the Agent connects to the Policy Manager, it downloads and caches the latest revision of policies. Once the Agent is up and running, it periodically attempts a cache refresh at a configurable interval. The default time is every 10 minutes.

For high availability scenarios, if an OWSM application is targeted to multiple nodes, it should be targeted to a cluster rather than to individual Managed Servers.

If a Managed Server has Web services deployed that are protected by OWSM, and the OWSM Agent is not able to communicate with any of the Policy Managers at startup time, Web service invocation fails.

2.1.3 OWSM Agent and Policy Manager Characteristics

The OWSM Agent is a set of .jar files available on every Oracle Fusion Middleware Managed server in a Web services stack.

The Policy Manager is contained in the `wsm-pm.ear` file. None of the services provided by OWSM are singletons, therefore, it can run in full active-active mode. OWSM services can be validated by `http://host:port/wsm-pm/validator`. This validator displays OWSM policies.

The OWSM Agent and Oracle Enterprise Manager interact with the Policy Manager using the EJB interfaces. The EJBs used in OWSM are stateless and can be deployed in a clustered environment. Therefore, there is no requirement to enable state replication in the cluster.

The OWSM Agent and Policy Manager need not be co-located. However, the Agent expects the Policy Manager to be deployed on at least one node of the domain. The

OWSM Agent has capabilities to auto-discover Policy Managers deployed in the domain.

External Dependencies

The OWSM Policy Manager depends on the following components:

- OWSM Repository for storing the policies
- OWSM Agent depends only on OWSM Policy Manager.

Both components must be available for OWSM to start and run properly.

2.1.4 OWSM Agent and Policy Manager Request Flow

When a protected Web service is accessed by a client application, the OWSM Agent queries the policy cache and enforces the applicable policies. Based on the policies, the request is authenticated, encrypted, decrypted, authorized or logged. It does not connect to the Policy Manager for any of these operations.

Runtime availability of the Policy Manager does not affect the functioning of the OWSM Agent, unless there is a configuration change, such as new Web services, which are protected by OWSM, being deployed, or new policies attached to existing Web services. If there is such a configuration change, then the OWSM Agent must connect to the Policy Manager to get the applicable policies. If it cannot connect after initial startup, it continues to operate based on the cached policies.

2.1.5 OWSM Configuration Artifacts

As described in "Managing OWSM Domain Configuration" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*, among other settings you can specify:

- Policy Manager URL (if configured)
- Cache Refresh Interval
- Clock skew, to allow for differences in system clock of the client and servers

These options are available from Oracle Enterprise Manager Fusion Middleware Control and are specific to each OWSM Agent installation.

Other configuration options at the container level, such as data sources for OWSM Repository location, and application targeting, are maintained as part of Oracle WebLogic Server Domain configuration, and are synchronized across a cluster of Oracle WebLogic Servers by Oracle WebLogic Server core infrastructure.

2.2 Understanding Policies

A Web service provider may define conditions (or policies) under which a service is to be provided. The WS-Policy framework enables you to specify policy information that can be processed by Web service applications, such as OWSM.

A policy is expressed as one or more policy assertions representing a Web service's capabilities or requirements. For example, a policy assertion may stipulate that a request to a Web service be encrypted. Likewise, a policy assertion can define the maximum message size that a Web service can accept.

WS-Policy expressions are associated with various Web services components using the WS-PolicyAttachment specification. WS-Policy information can be embedded in a

WSDL file, thus making it easy to expose Web service policies through a UDDI registry.

Policies can be attached directly to endpoints or globally to a range of endpoints of the same type, regardless of the deployment state using policy sets.

Note: Policy sets are supported for RESTful and Oracle Infrastructure Web services and clients. They are not supported for Java EE Web services and clients.

Oracle Fusion Middleware 12c (12.1.2) supports the categories of policies defined in [Table 2-1](#). The policies are part of the OWSM enterprise policy framework which allows policies to be centrally created and managed.

Table 2-1 Policy Categories

Policy Category	Description	Applies to SOAP, REST, or Both
Addressing	WS-Addressing policies that verify that SOAP messages include WS-Addressing headers in conformance with the WS-Addressing specification. Transport-level data is included in the XML message rather than relying on the network-level transport to convey this information. For more information on the WS-Addressing, see " Understanding Web Services Addressing " on page 3-31.	SOAP
Atomic Transactions	<p>WebLogic Web services enable interoperability with other external transaction processing systems, such as WebSphere, Microsoft .NET, and so on, through the support of the following specifications:</p> <ul style="list-style-type: none"> ■ WS-AtomicTransaction Version (WS-AT) 1.0, 1.1, and 1.2: http://docs.oasis-open.org/ws-tx/wstx-wsat-1.2-spec-cs-01/wstx-wsat-1.2-spec-cs-01.html ■ WS-Coordination Version 1.0, 1.1, and 1.2: http://docs.oasis-open.org/ws-tx/wstx-wscoor-1.2-spec-cs-01/wstx-wscoor-1.2-spec-cs-01.html <p>For more information about atomic transactions, see "Using Web Services Atomic Transactions" in <i>Developing Oracle Infrastructure Web Services</i>.</p>	SOAP
Configuration	Configuration policies that enable you to configure Web service features, such as Fast Infoset, schema validation, persistence, and so on.	SOAP
Management	Management policies that log request, response, and fault messages to a message log. Management policies may include custom policies.	SOAP

Table 2–1 (Cont.) Policy Categories

Policy Category	Description	Applies to SOAP, REST, or Both
Message Transmission Optimization Mechanism (MTOM) Attachments	<p>Binary content, such as an image in JPEG format, can be passed between the client and the Web service. In order to be passed, the binary content is typically inserted into an XML document as an <code>xsd:base64Binary</code> string. Transmitting the binary content in this format greatly increase the size of the message sent over the wire and is expensive in terms of the required processing space and time.</p> <p>Using MTOM, binary content can be sent as a MIME attachment, which reduces the transmission size on the wire. The binary content is semantically part of the XML document. Attaching an MTOM policy ensures that the message is converted to a MIME attachment before it is sent to the Web service or client.</p>	SOAP
Reliable Messaging	<p>Reliable messaging policies that implement the WS-ReliableMessaging standard describes a wire-level protocol that allows guaranteed delivery of SOAP messages, and can maintain the order of sequence in which a set of messages are delivered.</p> <p>The technology can be used to ensure that messages are delivered in the correct order. If a message is delivered out of order, the receiving system can be configured to guarantee that the messages will be processed in the correct order. The system can also be configured to deliver messages at least once, not more than once, or exactly once. If a message is lost, the sending system re-transmits the message until the receiving system acknowledges it receipt. For more information on WS-ReliableMessaging, see "Understanding Web Services ReliableMessaging" on page 3-32.</p>	SOAP
Security	<p>Security policies that implement the WS-Security 1.0 and 1.1 standards. They enforce message protection (message integrity and message confidentiality), and authentication and authorization of Web service requesters and providers. The following token profiles are supported: username token, X.509 certificate, Kerberos ticket, and Security Assertion Markup Language (SAML) assertion. For more information about Web service security tokens, see "Understanding Security Policies" on page 3-17 and "Understanding Security Tokens" on page 3-18.</p>	<p>Both</p> <p>A subset of security policies are supported for RESTful Web services, as described in "Which OWSM Policies Are Supported for RESTful Web Services?" in <i>Securing Web Services and Managing Policies with Oracle Web Services Manager</i>.</p>
SOAP Over JMS Transport	<p>Using SOAP over JMS transport, Web services and clients communicate using JMS destinations instead of HTTP connections, offering the following benefits:</p> <ul style="list-style-type: none"> ■ Reliability ■ Scalability ■ Quality of service <p>For more information about using SOAP over JMS transport, see "Using SOAP Over JMS Transport" in <i>Developing JAX-WS Web Services for Oracle WebLogic Server</i>.</p>	SOAP

2.3 Building Policies Using Policy Assertions

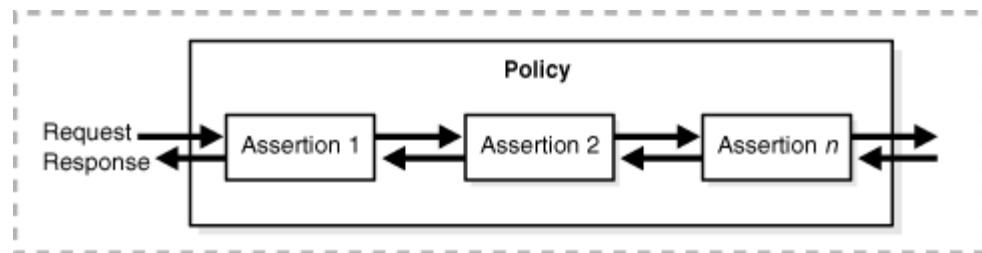
A policy is comprised of one or more policy **assertions**. A policy assertion is the smallest unit of a policy that performs a specific action for the request and response operations. Assertions, like policies, belong to one of the following categories: Reliable Messaging, Management, WS-Addressing, Security, and MTOM.

Policy assertions are chained together in a pipeline. The assertions in a policy are executed on the request message and the response message, and the same set of assertions are executed on both types of messages. The assertions are executed in the order in which they appear in the pipeline.

Note: See [Section 2.3.1, "Defining Multiple Policy Alternatives \(OR Groups\)"](#) to define multiple alternatives for policy enforcement with an **OR group**.

[Figure 2-3](#) illustrates a typical execution flow. For the request message, Assertion 1 is executed first, followed by Assertion 2, and Assertion n . Although the same assertions may be executed on the response message (if a response is returned at all), the actions performed on the response message differ from the request message, and the assertions are executed on the response message in reverse order. For the response message in [Figure 2-3](#), Assertion n is executed first, followed by Assertion 2, then Assertion 1.

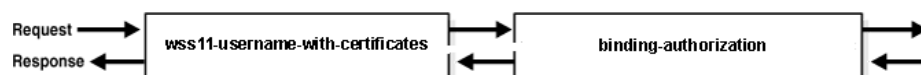
Figure 2-3 Policy Containing Assertions



For example, in [Figure 2-4](#), the policy contains two assertions:

1. `wss11-username-with-certificates`—Built using the `wss11_username_token_with_message_protection_service_template`, authenticates the user based on credentials in the WS-Security UsernameToken SOAP header.
2. `binding-authorization`—Built using the `binding_authorization_template`, provides simple role-based authorization for the request based on the authenticated subject at the SOAP binding level.

Figure 2-4 Example Policy With Two Assertions



When the request message is sent to the Web service, the assertions are executed in the order shown. When the response message is returned to the client, the same assertions are executed, but this time in reverse order. The behavior of the assertion for the request message differs from the behavior for the response message. And, in some instances, it is possible that nothing happens on the response. For example, in the example above, the authorization assertion is only executed as part of the request.

2.3.1 Defining Multiple Policy Alternatives (OR Groups)

To define multiple alternatives for policy enforcement, you can define a set of assertions, called an **OR group**, within a service policy. At run time, based on the assertions defined in the OR group on the service side, a client has the flexibility to choose which *one* of the assertions to enforce.

For example, if a service-side policy defines an OR group that consists of the following assertions:

- wss11-saml-with-certificates
- wss11-username-with-certificates

At run-time, the client can choose to enforce either the wss11-saml-with-certificates assertion OR wss11-username-with-certificates assertion.

There is no limit to the number of assertions that can be included in an OR group. Each assertion must be valid for the policy and should support the policy requirements. For example, you should not include a log assertion in an OR group that otherwise contains security assertions and that is designed to enforce security. In this case, the log assertion would pass in the event the security assertions failed, resulting in no security.

When defining the OR group, carefully consider the order in which the assertions are added and the settings that are configured. For example, consider the following scenario:

- On the client side, you have attached the wss11_username_token_with_message_protection_client_policy policy with Include Timestamp enabled.
- On the service side, you have attached a custom OR group policy with two wss11_username_token_with_message_protection_service_template assertions defined, the first with Include Timestamp disabled and the second with Include Timestamp enabled.

In this scenario, the first assertion will get executed and the response will be sent with no timestamp. As a result, processing on the client side will fail because it is expecting a timestamp. This type of situation can occur whenever a client policy assertion expects a greater number of security requirements than the executed service policy assertion.

The following predefined service policies contain OR groups:

- oracle/wss_saml_or_username_token_over_ssl_service_policy—For more information, see "oracle/wss_saml_or_username_token_over_ssl_service_policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.
- oracle/wss_saml_or_username_token_service_policy—For more information, see "oracle/wss_saml_or_username_token_service_policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.
- oracle/wss11_saml_or_username_token_with_message_protection_service_policy—For more information, see "oracle/wss11_saml_or_username_token_with_message_protection_service_policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.
- oracle/multi_token_rest_service_policy—For more information, see "oracle/multi_token_rest_service_policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

- oracle/multi_token_over_ssl_rest_service_policy—For more information, see "oracle/multi_token_over_ssl_rest_service_policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

There are three predefined service policies that contain OR groups:

- oracle/wss_saml_or_username_token_over_ssl_service_policy—For more information, see "oracle/wss_saml_or_username_token_over_ssl_service_policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.
- oracle/wss_saml_or_username_token_service_policy—For more information, see "oracle/wss_saml_or_username_token_service_policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.
- oracle/wss11_saml_or_username_token_with_message_protection_service_policy—For more information, see "oracle/wss11_saml_or_username_token_with_message_protection_service_policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

2.4 Understanding Policy Subjects

A *policy subject* is the target resource to which policies are attached. As defined in the Web Services Policy 1.5 Framework specification, at <http://www.w3.org/TR/ws-policy/>, a policy subject is an entity (for example, an endpoint, message, resource, or operation) with which a policy can be associated. There are different policies for different types of resources (for example, a Web service).

Table 2–2 lists the policy subjects to which you can attach OWSM policies. In addition, the table lists equivalent name that is used to identify the policy subject type using WLST, and the valid resource scope (applicable when you are creating policy sets, as described in "Global Policy Attachments Using Policy Sets" on page 2-11).

Table 2–2 Policy Subjects and Resource Scopes

Policy Subject	WLST Name	Valid Resource Scope (Policy Sets)
ADF SOAP Web Service Connection	ws-connection	<ul style="list-style-type: none"> ■ Domain ■ Application ■ Application Module or Connection ■ Reference or Web Service Client ■ Port
ESS SOAP JOB Callback	Reserved for future use.	Reserved for future use.
ESS SOAP JOB Invoker	Reserved for future use.	Reserved for future use.
OSB JCA Business Service	Reserved for future use.	Reserved for future use.
OSB JCA Proxy Service	Reserved for future use.	Reserved for future use.
OSB RESTful Business Service	Reserved for future use.	Reserved for future use.
OSB RESTful Proxy Service	Reserved for future use.	Reserved for future use.
OSB SOAP Business Service	Reserved for future use.	Reserved for future use.

Table 2–2 (Cont.) Policy Subjects and Resource Scopes

Policy Subject	WLST Name	Valid Resource Scope (Policy Sets)
OSB SOAP Proxy Service	Reserved for future use.	Reserved for future use.
RESTful Client	rest-client	<ul style="list-style-type: none"> ■ Domain ■ Application ■ Application Module or Connection ■ Resource Path
RESTful Resource	rest-resource	<ul style="list-style-type: none"> ■ Domain ■ Application ■ Application Module or Connection ■ RESTful Application, Service, or Web Service Endpoint ■ Resource Path ■ HTTP Method ■ RESTful Resource Sub-path
SOA Component	Reserved for future use.	Reserved for future use.
SOA JCA Service	Reserved for future use.	Reserved for future use.
SOA JCA Reference	Reserved for future use.	Reserved for future use.
SOA RESTful Reference	Reserved for future use.	Reserved for future use.
SOA RESTful Service	Reserved for future use.	Reserved for future use.
SOA SOAP Reference	Reserved for future use.	Reserved for future use.
SOA SOAP Service	Reserved for future use.	Reserved for future use.
SOAP Asynchronous Callback Client	ws-callback	<ul style="list-style-type: none"> ■ Domain ■ Application ■ Application Module or Connection ■ Callback Interface
SOAP Web Service	ws-service	<ul style="list-style-type: none"> ■ Domain ■ Application ■ Application Module or Connection ■ RESTful Application, Service, or Web Service Endpoint ■ Port
SOAP Web Service Client	ws-client	<ul style="list-style-type: none"> ■ Domain ■ Application ■ Application Module or Connection ■ Reference or Web Service Client ■ Port

2.5 Attaching Policies to Policy Subjects

There are two points in the lifecycle of an application in which you can attach policies: at design time and post deployment.

- At design time, you can attach OWSM policies to applications programmatically. You typically do this using your favorite IDE, such as Oracle JDeveloper. Oracle JDeveloper automates ADF client policy attachment. For more information, see "Developing and Securing Web Services" in *Developing Applications with Oracle JDeveloper*
- Post-deployment, you OWSM policies to Oracle Infrastructure Web Services, RESTful Web services, and Java EE Web services using Oracle Enterprise Manager Fusion Middleware Control or WLST. This provides the most power and flexibility because it moves Web service security to the control of the security administrator. Policies can be attached directly to an endpoint, or globally to a range of endpoints using policy sets.

Note: Policy sets are supported for RESTful and Oracle Infrastructure Web services and clients. They are not supported for Java EE Web services and clients.

OWSM places a limit on the number of policies that may be attached to a subject based on the categories of the assertions that they contain. To support the attachment of policies both directly and externally (globally), OWSM determines the effective set of policies for a subject by taking into account the category of assertions within each policy, the priority of policy attachments, run-time constraints, and the status (enabled/disabled) of any policy attachments. For more information about effective policy calculation for an endpoint, see "How the Effective Set of Policies is Calculated" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

Regardless of whether you attach a policy at design time or post-deployment, the client-side policy must be the equivalent of the one associated with the Web service. If the two policy files are different, and there is a conflict in the assertions contained in the files, then the invocation of the Web service operation returns an error.

For more information about attaching policies, see "Attaching Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

2.5.1 Direct Policy Attachment

You can attach OWSM policies to directly to policy subjects, such as SOAP Web service and client endpoints, ADF SOAP Web service connections, and RESTful resources and clients after the application is deployed. For details about how to attach policies directly, see "Attaching Policies Directly Using Fusion Middleware Control" and "Attaching Policies to Web Services and Clients Using WLST".

2.5.2 Global Policy Attachments Using Policy Sets

Note: Policy sets are supported for RESTful and Oracle Infrastructure Web services and clients. They are not supported for Java EE Web services and clients.

A policy set, which can contain multiple policy references, is an abstract representation that provides a means to attach policies globally to a range of endpoints of the same type, regardless of the deployment state. You can create and manage policy sets using both Fusion Middleware Control and the WebLogic Scripting Tool, WLST.

Attaching policies globally using policy sets allows an administrator to ensure that all subjects are secured in situations where the developer, assembler, or deployer did not explicitly specify the policies to be attached. For example, if the developer did not specify policies in annotations or include policy references in deployment descriptors, then the deployer must attach them or chance a potential security risk. By attaching policies globally to a set of subjects by type, the administrator can ensure that all subjects are secured by default independent of, and even prior to, deployment. The administrator can, for example, define a policy set that attaches a security policy to all Web service endpoints in a domain. In this case, any new services added to the domain automatically inherit the security configuration defined in the policy set. For more information, see "Determining the Secure Status of an Endpoint" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

Policies attached globally using policy sets also provide the following:

- The ability to specify configuration overrides on a referenced policy that apply to all endpoints to which the policy set is scoped. For information about configuring overrides, see "Overriding Configuration Properties for Globally Attached Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.
- The ability to specify a run-time constraint that determines the context in which the policy set is relevant. For example, you can specify that a service use message protection when communicating with external clients only since the message may be transmitted over insecure public networks. However, when communicating with internal clients on a trusted network, message protection may not be required. For more information, see "Specifying Run-time Constraints in Policy Sets" *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

You can disable a globally attached policy for a specific endpoint or range of endpoints using predefined policies that do not enforce any behavior that are included with your Fusion Middleware installation. When you attach one of these policies to a specific endpoint, or at a lower scope, you disable the behavior of the policy that was attached globally at the higher scope. For more information, see "Disabling a Globally Attached Policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

Policy set definitions are stored as separate XML documents in the OWSM Repository under the `/policysets/global` directory.

2.5.2.1 Subject Types and Scope of Resources

[Table 2–2, "Policy Subjects and Resource Scopes"](#) lists the policy subjects to which you can attach OWSM policies and the valid resource scopes. For more information, see "Defining the Type and Scope of Resources for Globally Attached Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

Note: When creating policy sets, the SOAP Web Service and SOAP Web Service Client subject types refer to Oracle Infrastructure Web services and clients only, respectively. You cannot create policy sets for Java EE Web services and clients.

2.5.2.2 Typical Uses for Global Policy Attachments

Typical scenarios in which attaching policies globally can be useful include:

- All subjects of a given type need to be protected with the same set of policies, each using their default configuration. For example, all services in a domain need to be protected with authentication (using SAML or Username token) and WSS11 message protection. You can create a policy set to attach the appropriate policy to all services in the domain.
- A subset of subjects need to be protected with the same set of policies, but these policies are different from the domain-wide default. For example, all services need to be protected with authentication (using SAML or Username token), but the General Ledger application also needs stronger WSS11 message protection. You create one policy set that attaches an authentication policy to all services, and a second policy set that attaches the stronger message protection policy to the General Ledger application.
- A single subject needs to be protected by a policy in a category that is not already covered by the current set of global policy attachments and both policies need to be applied. For example, a highly-sensitive financials-based service endpoint requires permission for a client to access it in addition to the authentication and message protection required. In this case, directly attach the authorization policy to the financials-based service endpoint. The direct attachment is combined with the policies attached globally and both policies will be enforced.
- An application has been deployed with design-time policy attachments and needs to convert to using global policy attachments. The `migrateAttachments` WLST command can be used to migrate the attachments. For more information, see "Migrating Direct Policy Attachments to Global Policy Attachments" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

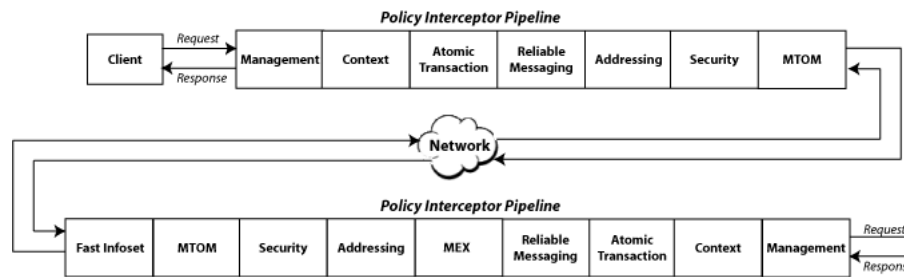
2.6 How Policies are Executed

When a request is made from a service consumer (also known as a client) to a service provider (also known as a Web service), the request is intercepted by one or more policy interceptors. These interceptors execute policies that are attached to the client and to the Web service. There are several types of interceptors that together form a policy interceptor chain. Each interceptor executes policies of the same type. The security interceptor intercepts and executes security policies, the MTOM interceptor intercepts and executes MTOM policies, and so on.

Policies attached to a client or Web service are executed in a specific order via the Policy Interceptor Pipeline, as shown in [Figure 2-5](#).

Note: A subset of OWSM policies are supported for RESTful Web services, as described in [Which OWSM Policies Are Supported for RESTful Web Services?](#) in *Securing Web Services and Managing Policies with Oracle Web Services Manager*. REST uses only the security policy interceptor type shown in [Figure 2-5](#).

Figure 2-5 Policy Interceptors Acting on Messages Between a Client and Web Service (SOAP)



As shown in the previous figure, when a client or a Web service *initiates* a message over SOAP, whether it be a request message in the case of a client, or a response message in the case of a Web service, the policies are intercepted in the following order: Management, Context (for SOAP request and response message handling), Atomic Transaction, Reliable Messaging, Addressing, Security, and MTOM. When a client or a Web service *receives* a message over SOAP, that is, a request message in the case of the Web service or a response message in the case of a client, the policies are executed in the reverse order and include additional interceptors: Fast Infoset, MTOM, Security, Addressing, MEX, Reliable Messaging, Atomic Transactions, Context, and Management.

A message may have one or more policies attached. Not every message will contain each type of policy. A message may contain a security policy and an MTOM policy. In this instance, the security interceptor executes the security policy, and the MTOM interceptor executes the MTOM policy. In this example, the other interceptors are not involved in processing the message.

The following describes how the policy interceptors act on messages between the client and the Web service over SOAP. (Refer to [Figure 2-5](#).)

1. The client sends a request message to a Web service.
2. The policy interceptors intercept and execute the policies attached to the client. After the client policies are successfully executed, the request message is sent to the Web service.
3. The request message is intercepted by policy interceptors which then execute any service policies that are attached to the Web service.
4. After the service policies are successfully executed, the request message is passed to the Web service. The Web service executes the request message and returns a response message.
5. The response message is intercepted by the policy interceptors which execute the service policies attached to the Web service. After the service policies are successfully executed, the response message is sent to the client.
6. The response message is intercepted by the policy interceptors which execute any client policies attached to the client.
7. After the client policies are successfully executed, the response message is passed to the client.

2.7 OWSM Predefined Policies and Assertion Templates

Note: The predefined policies and assertion templates installed in this release area read only. If you have modified existing predefined policies and assertion templates, they will not be overridden in the current release, but they will be overridden in the next release.

There is a set of predefined policies and assertion templates that are automatically available when you install Oracle Fusion Middleware. The predefined policies are based on common best practice policy patterns used in customer deployments.

You can immediately begin attaching these predefined policies to your Web services or clients. You can configure the predefined policies or create a new policy by making a copy of one of the predefined policies.

Predefined policies are constructed using assertions based on predefined assertion templates. You can create new assertion templates, as required.

For more information about the predefined policies and assertion templates, see:

- "Predefined Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*
- "Predefined Assertion Templates" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*

Note: WS-SecurityPolicy defines *scenarios* that describe examples of how to set up WS-SecurityPolicy policies for several security token types described in the WS-Security specification (supporting both WS-Security 1.0 and 1.1). The OWSM predefined policies support a subset of the WS-SecurityPolicy scenarios that represents the most common customer use cases.

2.8 Overriding Security Policy Configuration

Multiple Web services or clients may use the same policy. Each may have different policy configuration requirements such as username and password.

OWSM policy configuration override enables you to update the configuration on a per service or client basis without creating new policies for each. In this way, you can create policies that define default configuration values and customize those values based on your run-time requirements.

For example, you might specify the username and password when configuring a client policy, as the information may vary from client to client.

For more information about overriding security policy configuration, see "Overriding Policy Configuration Properties" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

You can define whether a configuration property can be overridden when creating custom assertions, as described in "Creating Custom Assertions" in *Developing Extensible Applications for Oracle Web Services Manager*.

2.9 Recommended Naming Conventions for Policies

The valid characters for directory, policy, and assertion template names are:

- Uppercase and lowercase letters
- Numerals
- Currency symbol (\$)
- Underscore (_)
- Hyphen (-)
- Spaces

Note: The first character in the name cannot be a hyphen or space.

Oracle recommends that you encode as much information as possible into the name of the policy so that you can tell, at a glance, what the policy does. For example, one of the predefined security policies that is delivered with Oracle Fusion Middleware 12c (12.1.2) is named `oracle/wss10_username_token_with_message_protection_service_policy`. Figure 2–6 identifies the different parts of this predefined policy name.

Figure 2–6 Identifying the Different Parts of a Policy Name



The following convention is used to name the predefined policies. The parts of the policy name are separated with an underscore character (_).

- **Path Location** – All policies are identified by the directory in which the policy is located. All predefined policies that come with the product are in the `oracle` directory.
- **Web services Standard** – If the policy uses a WS-Security standard, it is identified with `wss10` (WS-Security 1.0) or `wss11` (WS-Security 1.1). Or it could just be set to `wss` to indicate that it is independent of WS-Security 1.0 or 1.1.
- **Authentication token** – If the policy authenticates users, then the type of token is specified. The predefined options include:
 - `http_token` – HTTP token
 - `kerberos_token` – Kerberos token
 - `saml_token` – SAML token
 - `username_token` – Username and password token
 - `x509_token` – X.509 certificate token

You can also define custom authentication tokens.

- **Transport security** – If the policy requires that the message be sent over a secure transport layer, then the token name is followed by `over_ssl`, for example, `wss_http_token_over_ssl_client_template`.

- Message protection – If the policy also provides message confidentiality and message integrity, then this is indicated using the phrase *with_message_protection* as in Figure 2-6.
- Policy Type – Indicates the type of policy or assertion template— *client* or *service*. Use the term *policy* to indicate that it is a policy, or *template* to indicate that it is an assertion template. For example, there are predefined policy and template assertions that are distinguished, as follows:

```
wss10_message_protection_service_policy
```

```
wss10_message_protection_service_template
```

Whatever conventions you adopt, Oracle recommends you take some time to consider how to name your policies. This will make it easier for you to keep track of your policies as your enterprise grows and you create new policies.

It is recommended that you keep any policies you create in a directory that is separate from the oracle directory where the predefined policies are located. You can organize your policies at the root level, in a directory other than oracle, or in subdirectories. For example, all of the following are valid:

- wss10_message_protection_service_policy
- oracle/hq/wss10_message_protection_service_policy
- hq/wss10_message_protection_service_policy

Note: Use of the prefix "oracle_" in the policy name (for example, oracle_wss_http_token_service_policy) is not recommended as a best practice.

Understanding Web Service Security Concepts

This chapter describes the concepts behind Web services security. Web services security encompasses a number of requirements, such as authentication, authorization, and message protection.

This chapter contains the following sections:

- Section 3.1, "Overview of Web Service Security"
- Section 3.2, "Understanding Transport-level and Application-level Security"
- Section 3.3, "Understanding Authentication"
- Section 3.4, "Understanding Authorization"
- Section 3.5, "Understanding Message Protection"
- Section 3.6, "Understanding Keys and Certificates"
- Section 3.7, "How OWSM Uses the Credential Store"
- Section 3.8, "Understanding Security Policies"
- Section 3.9, "Understanding Security Tokens"
- Section 3.10, "Understanding Secure Attachments"
- Section 3.11, "Understanding Secure Conversation"
- Section 3.12, "Understanding the Kerberos Protocol"
- Section 3.13, "Understanding Web Services Addressing"
- Section 3.14, "Understanding Web Services Trust"
- Section 3.15, "Understanding Web Services ReliableMessaging"

Note: A subset of OWSM authentication and authorization policies are supported for RESTful Web services, as described in Which OWSM Policies Are Supported for RESTful Web Services? in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

This section primarily describes Web services over SOAP.

3.1 Overview of Web Service Security

Because of its nature (loosely coupled connections) and its use of open access (mainly HTTP), SOA implemented by Web services adds a new set of requirements to the security landscape:

- **Authentication**—Verifying that the user is who she claims to be. A user's identity is verified based on the credentials presented by that user, such as:
 1. Something one has, for example, credentials issued by a trusted authority such as a passport (real world) or a smart card (IT world).
 2. Something one knows, for example, a shared secret such as a password.
 3. Something one is, for example, biometric information.

Using a combination of several types of credentials is referred to as "strong" authentication, for example using an ATM card (something one has) with a PIN or password (something one knows). See "[Understanding Authentication](#)" on page 3-4 for more information.

- **Authorization (or Access Control)**—Granting access to specific resources based on an authenticated user's entitlements. Entitlements are defined by one or several attributes. An attribute is the property or characteristic of a user, for example, if "Marc" is the user, "conference speaker" is the attribute. See "[Understanding Authorization](#)" on page 3-5 for more information.
- **Confidentiality, privacy**—Keeping information secret. Accesses a message, for example a Web service request or an email, as well as the identity of the sending and receiving parties in a confidential manner. Confidentiality and privacy can be achieved by encrypting the content of a message and obfuscating the sending and receiving parties' identities. See "[Understanding Message Protection](#)" on page 3-5 for more information.
- **Integrity, non repudiation**—Making sure that a message remains unaltered during transit by having the sender digitally sign the message. A digital signature is used to validate the signature and provides non-repudiation. The timestamp in the signature prevents anyone from replaying this message after the expiration. For more information, see "[Understanding Message Protection](#)" on page 3-5.

Web services security requirements also involve credential mediation (exchanging security tokens in a trusted environment), and service capabilities and constraints (defining what a Web service can do, under what circumstances).

In many cases, Web services security tools such as OWSM rely on Public Key Infrastructure (PKI) environments. A PKI uses cryptographic keys (mathematical functions used to encrypt or decrypt data). Keys can be private or public. In an asymmetric cipher model, the receiving party's public key is used to encrypt plaintext, and the receiving party's matching private key is used to decrypt the ciphertext. Also, a private key is used to create a digital signature by signing the message, and the public key is used for verifying the signature. Public-key certificates (or certificates, for short) are used to guarantee the integrity of public keys.

Web services security requirements are supported by industry standards both at the transport level (Secure Socket Layer) and at the application level relying on XML frameworks.

For more information about the specifications and standards supported by Web services, see [Appendix A, "Web Service Security Standards."](#)

Note: Oracle has been instrumental in contributing to emerging standards, in particular the specifications hosted by the OASIS Web Services Secure Exchange technical committee.

3.1.1 Web Service Security Requirements

The following summarize the Web service security requirements:

- Use transport security to protect the communication channel between the Web service consumer and Web service provider.
- Use message-level security to ensure confidentiality by digitally encrypting message parts; integrity using digital signatures; and authentication by requiring username, X.509, or SAML tokens.

Oracle Web Services Manager (WSM) is designed to define and implement Web services security in heterogeneous environments, including authentication, authorization, message encryption and decryption, signature generation and validation, and identity propagation across multiple Web services used to complete a single transaction.

3.2 Understanding Transport-level and Application-level Security

Security concepts can be divided into those that pertain to the transport level and to the application level. Transport-level security secures the communications channel between applications. An example of a transport-level security protocol is Secure Socket Layer (SSL), otherwise known as Transport Layer Security (TLS), the Internet Engineering Task Force (IETF) officially standardized version of SSL. This is the most widely used transport-level data-communication protocol providing:

- Authentication (the communication is established between two trusted parties).
- Confidentiality (the data exchanged is encrypted).
- Message integrity (the data is checked for possible corruption).
- Secure key exchange between client and server.

SSL provides a secure communication channel, however, when the data is not "in transit," the data is not protected. This makes the environment vulnerable to attacks in multi-step transactions. (SSL provides point-to-point security, as opposed to end-to-end security.)

SSL can be used in three modes:

- No authentication: Neither the client nor the server authenticates itself to the other. No certificates are sent or exchanged. In this case, only confidentiality (encryption/decryption) is used.
- One-way authentication (or server authentication): Only the server authenticates itself to the client. The server sends the client a certificate verifying that the server is authentic. This is typically the approach used for Internet transactions such as online banking.
- Two-way authentication (or bilateral authentication): Both client and server authenticate themselves to each other by sending certificates to each other. This approach is necessary to prevent attacks from occurring between a proxy and a Web service endpoint.

SSL uses a combination of secret-key and public-key cryptography to secure communications. SSL traffic uses secret keys for encryption and decryption, and the exchange of public keys is used for mutual authentication of the parties involved in the communication.

Application-level security complements transport-level security. Application-level security is based on XML frameworks defining message confidentiality, integrity, authenticity (also known as message protection); message structure; trust management and federation. These components of application-level security are described in greater detail in the following sections, "[Understanding Message Protection](#)", "[Understanding Authentication](#)", and "[Understanding Authorization](#)".

3.3 Understanding Authentication

Authentication is verifying that the user is who they claim to be. A user's identity is verified based on the credentials presented by that user, such as:

- Something one has, for example, credentials issued by a trusted authority such as a digital certificate, standard Security Assertion Markup Language (SAML) token, or Kerberos token.
- Something one knows, for example, a shared secret such as a password.
- Something one is, for example, biometric information.

Using a combination of several types of credentials is referred to as "strong" authentication, for example using an ATM card (something one has) with a PIN or password (something one knows).

SAML is one of the most interesting security tokens because it supports both authentication and authorization. SAML is an open framework for sharing security information on the Internet through XML documents. SAML includes three parts:

- SAML Assertion—How you define authentication and authorization information.
- SAML Protocol—How you ask (SAML Request) and get (SAML Response) the assertions you need.
- SAML Bindings and Profiles—How SAML assertions ride "on" (Bindings) and "in" (Profiles) industry-standard transport and messaging frameworks.

The full SAML specification is used in browser-based federation cases. However, Web services security systems such as OWSM only use SAML assertions. The protocol and bindings are taken care of by WS-Security and the transport protocol, for example HTTP.

SAML assertions and references to assertion identifiers are contained in the WS-Security Header element, which in turn is included in the SOAP Envelope Header element (described in the WS-Security SAML Token Profile). The SAML security token is particularly relevant in situations where identity propagation is essential.

3.3.1 Digest Authentication

OWSM supports digest based authentication in username-token authentication policies.

Digest authentication is an authentication mechanism in which a Web application authenticates itself to a Web service by sending the server a digest, which is a cryptographic hash of the password, nonce, and timestamp.

When using digest authentication:

1. The client makes an un-authenticated request to the Web service, and the server sends a response with a digest authentication challenge indicating that it supports digest authentication.
2. The client generates a nonce and sends it to the service along with a timestamp, digest, and username. The digest is a cryptographic hash of the password, nonce, and timestamp.
3. The server generates the hash itself from the password (retrieved from the service store), nonce and timestamp (from the message), and if the generated hash matches the hash in the request, the request is allowed.

The advantage of digest authentication is it is resistant to replay attacks. The implementation maintains a cache of used nonces/timestamps for a specified period of time. All requests with a timestamp older than the specified timestamp are rejected as well as any requests that use the same timestamp/nonce pair as the most recent timestamp/nonce pair still in the cache. WebLogic Server stores this cache in a database.

3.4 Understanding Authorization

Frequently, authentication is the first step of determining whether a user should be given access to a Web service. After the user is authenticated, the second step is to verify that the user is authorized to access the Web service.

Authorization (also known as access control) is granting access to specific resources based on an authenticated user's entitlements. Entitlements are defined by one or several attributes. An attribute is the property or characteristic of a user, for example, if "Marc" is the user, "conference speaker" is the attribute.

Authorization enables you to determine what operations authenticated clients can access. There are three basic approaches to authorization:

- Role-based—Role-based security is based on the notion that a set of identities, known as principals, can be grouped into roles, and then a policy can be applied to each of the roles.
- Identity based—Identity Model enables you to manage claims and policies in order to authorize clients. With this approach, you can verify claims contained within the authenticated users' credentials. These claims can be compared with the set of authorization policies for the WCF service. Depending on the claims provided by the client, the service can either grant or deny access to the operation or resources. Identity Model is useful for fine-grained authorization and is most beneficial when using issue token authentication.
- Resource based—Individual resources are secured by using Windows access control lists (ACLs).

3.5 Understanding Message Protection

Message protection encompasses two concepts, **message confidentiality** and **message integrity**.

Message confidentiality involves keeping the data secret, as well as the identities of the sending and receiving parties. Confidentiality is achieved by encrypting the content of messages and obfuscating the identities of the sending and receiving parties. The sender uses the recipient's public key to encrypt the message. Only the recipient's private key can successfully decrypt the message, ensuring that it cannot be read by third parties while in transit. The Web service's base64-encoded public certificate is

published in the WSDL for use by the Web service client, as described in "Using the Service Identity CertificateExtensions" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

Message integrity is achieved by having an authority digitally sign the message. Digital signatures are used to authenticate the sender of the SOAP message and to ensure the integrity of the SOAP message (that is, to ensure that the SOAP message is not altered while in transit).

When a digital signature is applied to a SOAP message, a unique hash is produced from the message, and this hash is then encrypted with the sender's private key. When the message is received, the recipient decrypts the hash using the sender's public key.

Note: Generally, the recipient does not need to have the sender's public key in its keystore to validate the certificate. It is sufficient to have the root certificate in the keystore to verify the certificate chain. However, if the sender's public key is not present in the message, as in the case of the Thumbprint and SerialIssuer mechanisms, the sender's public key must be in the recipient's keystore.

This serves to authenticate the sender, because only the sender could have encrypted the hash with the private key. It also serves to ensure that the SOAP message has not been tampered with while in transit, because the recipient can compare the hash sent with the message with a hash produced on the recipient's end.

The message-protection assertion templates and predefined policies can be used to protect request and response messages by doing the following:

- Signing messages
- Encrypting messages
- Signing *and* encrypting messages
- Decrypting messages
- Verifying signatures
- Decrypting messages *and* verifying signatures

3.5.1 Message Encryption

The XML encryption specification describes a process for encrypting data and representing the result in XML. Specifically, XML encryption defines:

- How digital content is encrypted and decrypted.
- How the encryption key information is passed to a recipient.
- How encrypted data is identified to facilitate encryption.

An XML document may be encrypted as a whole or in part.

[Example 3-1](#) illustrates credit card data represented in XML.

Example 3-1 XML Representation of Credit Card Data

```
<PaymentInfo xmlns="http://www.example.com/payment">
  <CreditCard>
    <Name>John Smith</Name>
    <CreditCardNumber>4019 2445 0277 5567</NCreditCardNumber>
    <Limit>5000</Limit>
  </CreditCard>
</PaymentInfo>
```



```

    <Issuer>Example Bank</Issuer>
    <Expiration>04/02</Expiration>
  </CreditCard>
</PaymentInfo>

```

Example 3–2 illustrates the same XML snippet with the credit card number encrypted and represented by a cipher value.

Example 3–2 XML Representation of Encrypted Credit Card Data

```

<PaymentInfo xmlns='http://www.example.com/payment">
  <CreditCard>
    <Name>John Smith</Name>
    <CreditcardNumber>
      <EncryptedData xmlns="http://www..." Type="http://www...">
        <CipherData>
          <CipherValue>A23B4...5C56</CipherValue>
        </CipherData>
      </EncryptedData>
    <Limit>5000</Limit>
    <Issuer>Example Bank</Issuer>
    <Expiration>04/02</Expiration>
  </CreditCard>
</PaymentInfo>

```

3.5.2 Message Signing (XML Signature)

The XML Signature specification describes signature processing rules and syntax. XML Signature binds the sender's identity (or "signing entity") to an XML document. The document is signed using the sender's private key; the signature is verified using the sender's public key.

Signing and signature verification can be done using asymmetric or symmetric keys. XML Signature also ensures non-repudiation of the signing entity, that is, it provides proof that messages have not been altered since they were signed.

A signature can apply to a whole document or just part of a document, as shown in the following example.

Example 3–3 XML Representation of Signed Data

```

<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
<!-- The signedInfo element allows us to sign any portion of a
document -->
  <SignedInfo>
    <CanonicalizationMethod Algorithm="http://www..."/>
    <SignatureMethod Algorithm="http://www..."/>
    <Reference URI="#Body">
      <DigestMethod Algorithm="http://www..."/>
      <DigestValue>o+jtqliertF6DrUb...X809M/CmySg</DigestValue>
    </Reference>
  </SignedInfo>
<!-- Following is the result of running the algorithm over the
document. If changes are made to the document, the SignatureValue is
changed. The security application verifies the SignatureValue,
extracts the X.509 cert and uses it to authenticate the user -->
  <SignatureValue>oa+ttbsvSFi...EtrD2oNC5</SignatureValue>
  <KeyInfo>
    <KeyValue>
      <!-- Following is the public key that matches the private key

```

```
that signs the document -->
<RSAKeyValue>
  <Modulus>5TT/oolzTiP++Ls6GLQUM8xoFFrAlZQ...</Modulus>
  <Exponent>EQ==</Exponent>
</RSAKeyValue>
</KeyValue>
<!-- Following is the certificate -->
<X509Data>
  <X509Certificate>wDCCAXqgAwIBAgI...</X509Certificate>
</X509Data>
</KeyInfo>
</Signature>
```

3.6 Understanding Keys and Certificates

Before you can use any message protection security policies or message protection and authentication with SSL security policies, you need to set up your keystores and truststores. (Authentication-only security policies do not require keys.)

The keystore contains the entities private keys and certificates associated with those private keys. A truststore contains certificates from a Certificate Authority (CA), or other entities that this entity trusts. The keystore and the truststore can be maintained together in a common store, such as with Oracle Web Services Manager (OWSM).

Before configuring your Web services, you need to determine the type of private keys and certificates required, the names for the keys and keystores, and then set up your environment accordingly.

3.6.1 Overview of Private Keys and Certificates

Private keys, digital certificates, and trusted certificate authorities establish and verify server identity and trust.

SSL uses public key encryption technology for authentication. With public key encryption, a public key and a private key are generated for a server. Data encrypted with the public key can only be decrypted using the corresponding private key and data verified with a public key can only have been signed with the corresponding private key. The private key is carefully protected so that only the owner can decrypt messages that were encrypted using the public key.

The public key is embedded in a digital certificate with additional information describing the owner of the public key, such as name, street address, and e-mail address. A private key and digital certificate provide identity for the server.

The data embedded in a digital certificate is verified by a certificate authority and digitally signed with the certificate authority's digital certificate. Well-known certificate authorities include Verisign and Entrust.net. The trusted certificate authority (CA) certificate establishes trust for a certificate.

An application participating in an SSL connection is authenticated when the other party evaluates and accepts the application's digital certificate. Web browsers, servers, and other SSL-enabled applications generally accept as genuine any digital certificate that is signed by a trusted certificate authority and is otherwise valid. For example, a digital certificate can be invalidated because it has expired or the digital certificate of the certificate authority used to sign it expired. A server certificate can be invalidated if the host name in the digital certificate of the server does not match the URL specified by the client.

The different types of trusted certificates that you can use in your environment, along with the benefits and disadvantages of each, are as follows:

- **Self-signed certificates** — A self-signed certificate is a certificate that is signed by the entity creating it.

Benefits:

- Easy to generate because you can do it yourself, for example, using the `keytool` command for the JKS keystore as described in "Generating Private Keys and Creating the Java Keystore" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.
- Can be used in production as long as you use only a new certificate that you have generated.

Disadvantages:

- Self-signed certificates can quickly become unmanageable if you have many clients and services that need to communicate with each other. For example, if you have three clients communicating with two services, you need to generate a private key and self-signed certificate for both services, and then import the two certificates into the truststore of all three clients.

- **Demonstration Certificate Authority (CA) signed certificates**— WebLogic Server includes a set of demonstration private keys, digital certificates, and trusted certificate authorities that are for development only.

Benefits:

- Easy to use because they are available and configured for use in the default WebLogic Server installation in a development environment.

Disadvantages:

- Should never be used in a production environment. The private key of the demo certificate CA is available to all installations of WebLogic Server, therefore each installation can generate a demo CA signed certificate using the same key. As a result, you cannot trust these certificates.

- **Internal CA signed certificates** — An internal CA signed certificate is a certificate that you issue yourself using an internal CA that you can setup for your intranet. This type of certificate can be used if your services are mostly internal only.

Benefits:

- You have complete control over the certificate issuance process because you create the certificates yourself. You can control to whom the certificates are issued, how long the certificates remain valid, and so on. For example, if you are issuing certificates to your partners, you can issue them only to partners in good standing.

Disadvantages:

- You need to ensure that all clients have the internal CA root certificate imported into their truststore.

- **External CA signed certificates** — An external CA signed certificate is a certificate that has been issued by a reputable CA such as Verisign and Entrust.net. This type of certificate should be used if your services are external facing.

Benefits:

- In most cases, clients are already set up to trust these external CAs. Therefore, those clients do not have to modify their truststore.

Disadvantages:

- You do not have any control over the certificate issuance process.

3.6.2 How Different Security Policies Use Private Keys and Certificates

OWSM security policies that require the use of private keys address two aspects: message protection and authentication:

- Message protection encompasses two concepts, **message confidentiality** and **message integrity**. Message confidentiality involves keeping the data secret and is achieved by encrypting the content of messages. Message integrity ensures that a message remains unaltered during transit by having the sender digitally sign the message.
- Authentication involves verifying that the user is who they claim to be. A user's identity is verified based on the credentials presented by that user.

The predefined OWSM policies that are included with your installation support various options for message protection and authentication. These options are described in the following sections.

Note: The naming convention used for OWSM policies identifies the type of options being used. For example, the policy `oracle/wss10_username_token_with_message_protection_service_policy` is a message protection service policy that uses the wss10 Web services standard and requires a `username_token` for authentication. For more information about policy naming conventions, see "[Recommended Naming Conventions for Policies](#)" on page 2-15.

3.6.2.1 Message Protection Policy Types

The types of message protection policies and how they work are described in the following sections.

3.6.2.1.1 SSL Policies that include the SSL option, such as `oracle/wss_saml_or_username_token_over_ssl_service_policy`, use one-way SSL for message protection.

When using policies of this type, you need to do the following:

- On the service side, set up private keys at the SSL termination point as described in "[Setting Up Private Keys and Certificates for SSL Policies](#)" on page 3-14.
- On the client side, set up the truststore to trust the service keys.

The private key is used to protect the messages for the SSL handshake, at which time the client and service agree on a shared session key. After the SSL handshake, the private key is not used, and all traffic between the client and the service are signed and encrypted using the shared session key.

For information on how to configure SSL, see "Configuring Transport-Level Security (SSL)" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

3.6.2.1.2 wss11 Policies of this type use WS-Security 1.1 for message protection.

When using wss11 policies, you need to do the following:

- On the service side, set up private keys and define as the Encryption Key Alias in the OWSM Keystore Configuration screen. For details see "Configuring the

OWSM Keystore Using Fusion Middleware Control" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

- On the client side, you need to configure the client-side trust by obtaining the server's certificate in one of the following ways:
 - Use the service's public certificate published in the WSDL using the Service Identity Certificate extension as described in "Using the Service Identity Certificate Extensions" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*. You also need to import either the server certificate itself, or the root certificate from the CA that issued the server certificate, into the client truststore. You can choose any alias name for the server certificate.
 - Import the server certificate into the client keystore using any alias you choose, and specify that alias using the `keystore.recipient.alias` property using a configuration override when you attach the policy. For this method you need to import the actual server certificate, you cannot import the CA root certificate.

For each request, the following occurs:

1. The client creates a symmetric key, encrypts this symmetric key with the service's public key as configured with Encryption Key Alias, and then encrypts and signs the whole message with the symmetric key.
2. When the service receives the message, it decrypts the encrypted key first, and then decrypts and verifies the whole message.
3. The Web service then uses the same symmetric key to encrypt and sign the response that it sends back to the client.

3.6.2.1.3 wss10 Policies of this type use WS-Security 1.0 for message protection.

When using wss10 policies, you need to do the following:

- Set up private keys on both the client and service side. On the client side, you need to set a signature key alias, and on the service side you need both an encryption key alias and signature key alias. Note that you can normally use the same key for both.
- On the client side, you need to configure the client-side trust by obtaining the server's certificate in one of the following ways:
 - Use the service's public certificate published in the WSDL using the Service Identity Certificate extension as described in "Using the Service Identity Certificate Extensions" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*. You also need to import either the server certificate itself, or the root certificate from the CA that issued the server certificate, into the client truststore. You can choose any alias name for the server certificate.
 - Import the server certificate into the client keystore using any alias you choose, and specify that alias using the `keystore.recipient.alias` property using a configuration override when you attach the policy. For this method you need to import the actual server certificate, you cannot import the CA root certificate.
- On the service side, you need to configure the service to trust the client, either by importing these certificates directly, or importing the CA that issued these certificates.

Similar to the wss11 option, the client creates a symmetric key, and then encrypts the symmetric key with the service's public key. The difference, however, is that it only uses this symmetric key for encrypting the message; it doesn't use it for signing the message. Instead, the client signs the request message with its own private signature key as defined by the Signature Key alias, and the service signs the response with its private signature key.

3.6.2.2 Authentication Token Policy Types

The tokens that are supported for authentication, and the private keys and certificates that are used with these policy types are described in the following sections. For information on how to configure authentication, see "Configuring Authentication" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

Note that in the following sections, "signature key alias" means different things in different contexts.

- In SAML sender vouches policies, it is the key used to sign the SAML assertion. This proves the authenticity of the SAML assertion, and SAML Login module will then assert the user specified in the SAML assertion.
- In wss10 policies, it is used to sign the request and response message to prevent them from being tampered over the wire.
- In X.509 authentication policies, it is used to authenticate a particular end user.

3.6.2.2.1 Username Token A username token carries basic authentication information such as a username and password. When a username token is used with an authentication-only policy, no private keys are used. When used in a policy that includes authentication and message protection, the keys required for message protection are required.

3.6.2.2.2 Kerberos Token A Kerberos token is comprised of a binary authentication and session token. When a kerberos token is used with an authentication-only policy, no private keys are used. When used in a policy that includes authentication and message protection, the keys required for message protection are required.

3.6.2.2.3 X.509 Certificate Token Request messages are signed with the end user's signature key. On the client side you need to configure a signature key alias with the end user's signature key.

3.6.2.2.4 SAML Sender Vouches Token In SAML sender vouches, the client signs the SAML token with its own private signature key.

Use the SAML sender vouches token with each of the message protection options as follows:

- With SSL: SAML sender vouches requires two-way SSL. Therefore, you need to set up an SSL client-side private key, and corresponding trust certificate on the service side. If your SSL terminates before WebLogic Server, such as in the Oracle HTTP Server or in the Load balancer, you must configure these layers to propagate the client certificate all the way to WebLogic Server.
- With wss11: Normally wss11 does not need a client-side signature key. However, when you use wss11 with SAML, you must set up a signature key on the client side, and configure it using the signature key alias. You must also add this client certificate or its issuer to the service's truststore.

- With wss10: There is no additional setup to use SAML. The regular client signature key that is used for signing the request is also used for signing the SAML token.

Note: Be very cautious when using the SAML signature key. It is a very powerful key as it enables the client side to impersonate any user. Consider configuring the server side to limit the number of SAML signers that is accepts, by setting up a Trusted DN list. For information about setting up a trusted DN, see "Configuring SAML Trusted Issuers and DN Lists Using Fusion Middleware Control" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

3.6.2.2.5 SAML Bearer and SAML HOK Tokens from an STS For these options, the client does not construct the SAML token. Instead it is STS that constructs and signs the SAML token.

When using tokens from an STS, you must add the STS's certificate or its issuer to the service's truststore. Optionally, you can configure the STS in the Trusted DN list.

3.6.3 How OWSM Locates Keystore And Key Passwords for the JKS Keystore

OWSM expects JKS keystore and key passwords to be in the Credential Store Framework (CSF). Here is how it works.

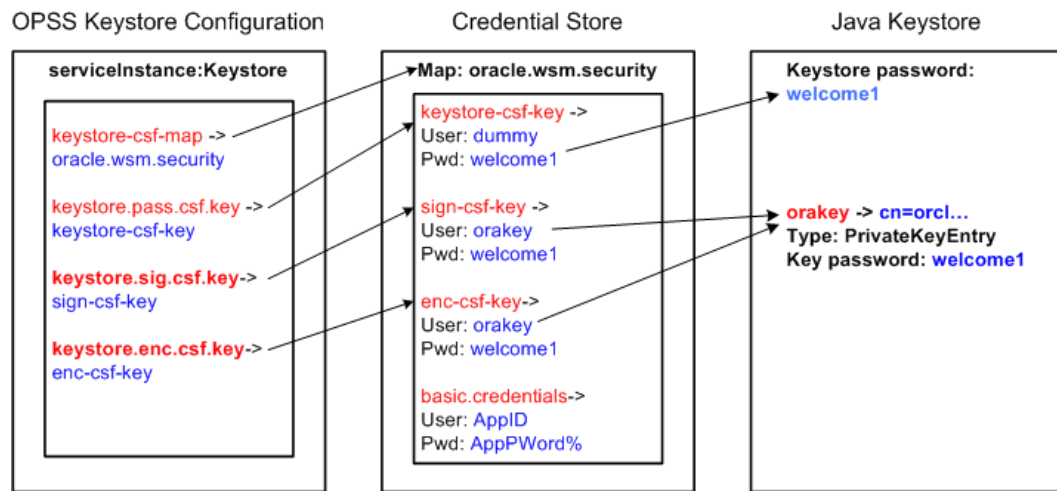
- A JKS keystore file is protected by a keystore password.
- A keystore file consists of zero or more private keys, and zero or more trusted certificates. Each private key has its own password, (although it is common to set the key passwords to be the same as the keystore password). OWSM needs to know both the keystore password and key password.
- The CSF consists of many *maps*, each with a distinct name. OWSM only uses the map `oracle.wsm.security`.
- Inside each map is a mapping from multiple `csf-key` entries to corresponding credentials. A `csf-key` is just a simple name, but there can be many different types of credentials. The most common type of credential is a password credential which is primarily comprised of a username and a password.

OWSM refers to the following `csf-keys` for the JKS keystore inside the `oracle.wsm.security` map:

- `keystore-csf-key` - This key should contain the keystore password. The username is ignored.
- `enc-csf-key` - This key should contain the encryption key alias as the username, and the corresponding key password.
- `sign-csf-key` - This key should contain the signature key alias as the username, and the corresponding key password.

In addition to these `csf-keys`, you should add a `csf-key` entry for every new private key that you want OWSM to use, for example when you want to specify signature and encryption keys in configuration overrides.

Figure 3–1 illustrates the relationship between the JKS keystore configuration in the OPSS, the `oracle.wsm.security` map in the credential store, and the OWSM Java keystore.

Figure 3–1 OWSM Keystore Configuration for Message Protection

As shown in the figure:

- The `keystore.csf.map` property points to the OWSM map in the credential store that contains the CSF aliases. In this case `keystore.csf.map` is defined as the recommended name `oracle.wsm.security`, but it can be any value.
- The `keystore.pass.csf.key` property points to the CSF alias `keystore-csf-key` that is mapped to the username and password of the JKS keystore. Only the password is used; username is redundant in the case of the keystore.
- The `keystore.sig.csf.key` property points to the CSF alias `sign-csf-key` that is mapped to the username and password of the private key that is used for signing in the JKS keystore.
- The `keystore.enc.csf.key` property points to the CSF alias `enc-csf-key` that is mapped to the username and password of the private key that is used for decryption in the JKS keystore.

3.6.4 Setting Up Private Keys and Certificates for SSL Policies

The following list summarizes the keys and trust you must configure on the client and service side to use SSL policies:

- **Service-side configuration:** For SSL security policies, you must setup the private keys at the SSL termination point. These termination points typically consist of one of the following:
 - Java EE container, such as WebLogic Server. For configuration details, see "Configuring Keystores for SSL" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.
 - Oracle HTTP Server, if you have configured it as a Web proxy between the client and WebLogic Server. For configuration details, see "Configuring SSL on Oracle HTTP Server" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.
 - Load balancer, if you have a load balancer in front of WebLogic Server or Oracle HTTP Server.

Note: With SSL you can only have one private key per server, so if there are multiple Web services running on the same server, they all use the same private key. This SSL private key needs to be generated with the same DN as the host name, although for testing purposes, you can turn off the host name verifier on the client side.

Sample basic configuration: Use the demonstration digital certificates, private keys, and trusted CA certificates that are included with WebLogic Server. These keys and certificates are provided for development use only and should not be used in a production environment.

Advanced configuration: In a production environment, use an internal or external CA.

- **Client-side configuration:** On the client side, you need to import the server certificates into the client truststore. If the server side is using self-signed certificates, you need to include them directly. If the server side is using certificates that are signed using a CA, import the CA root certificate into the client truststore. Note that each type of Web service client has a different client truststore:
 - For Java EE (WebLogic) Web services, you need to import the keys into the WebLogic Server trust store. The demonstration CA certificate is already present in the WebLogic Server truststore.
 - For Oracle Infrastructure Web services you need to specify the truststore using `javax.net.ssl*` system properties, or specify it in the connection. For details, see "Configuring SSL for a Web Service Client" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.
 - For asynchronous Web services, you need to configure the truststore as described in "Configuring SSL for Asynchronous Web Services" in *Developing Oracle Infrastructure Web Services*.

3.6.5 Setting up Private Keys and Certificates for Message Protection Policies

For OWSM message protection security policies, you need to setup your private keys in the OWSM keystore.

There is a single OWSM keystore per domain, and it is shared by all Web services and clients running in the domain. This keystore contains both private keys and trust certificates. The JDK cacerts file is not used by OWSM.

The following sections describe a basic OWSM keystore configuration and an advanced configuration.

Sample Basic Configuration

The easiest way to set up the OWSM keystore is to create a single self-signed private key and use it for the entire domain. When you create the private key and keystore, you specify a name and a password for the keystore, for example `default-keystore.jks` as the keystore name if you are using a JKS keystore, and `welcome1` as the password for the keystore. You also specify an alias name and password to use when referring to the private key, for example `orakey` as the alias name and `welcome1` as the key password. You can use the same key and alias for both the signature key alias and the encryption key alias, and the same password for both the keystore and the alias. You do not need to add any trusted certificates, as certificates associated with private keys are automatically considered as trusted.

Once you have created the keys and keystore, you need to provide the keystore password, and alias names and passwords to OWSM. You can do so using either Fusion Middleware Control or WLST.

The procedures in "Generating Private Keys and Creating the Java Keystore" and "Configuring the OWSM Keystore Using Fusion Middleware Control" in *Securing Web Services and Managing Policies with Oracle Web Services Manager* describe how to setup this basic configuration for a JKS keystore using the names and passwords specified in this example. In your own environment, you should use names and passwords that are appropriate for your configuration.

As long as your client and server are on the same domain, this set up is sufficient to work with most of the policies. That is, you can use any wss10 or wss11 policies with or without SAML.

If you have multiple related domains that share a common JPS root, you can copy this keystore file to all the domains. By doing so, all the related domains will share this single key for all encryption and signing.

Advanced Setup Considerations

As described in "[Sample Basic Configuration](#)" on page 3-15, the simplest way to set up message protection security is to have a single private key for all Web services in the domain.

For more sensitive Web services, you need to configure each Web service to use its own distinct private encryption key. These private keys need to exist in the OWSM keystore. Ensure that each one uses a different alias name, for example `ServiceA`, and `ServiceB`, and that you add the aliases to the credential store. When you attach a policy to the service, you need to use a configuration override to indicate the specific alias name that the Web service requires, otherwise it will use the default alias that you configured for the domain, for example `oraclekey`.

The procedure in "Adding Keys and User Credentials to the Credential Store" in *Securing Web Services and Managing Policies with Oracle Web Services Manager* describes how to add these sample aliases to the credential store.

You should also use trusted certificates issued by an internal or external CA, instead of self-signed certificates, because it is much easier to manage the trusted CA certificates. Be sure, however, to set up the SAML signers Trusted DN list, as described in "Configuring SAML Trusted Issuers and DN Lists Using Fusion Middleware Control" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*. This is especially important if you import external CA certificates into the OWSM Keystore, otherwise any user with a certificate will be able to sign a SAML token and impersonate any user.

3.7 How OWSM Uses the Credential Store

The Credential Store Framework (CSF) provides a way to store, retrieve, and delete credentials for a Web Service and other applications. OWSM uses the CSF to manage the credentials in a secure form by retrieving the following information:

- Alias names and passwords for keys in the Java keystore
For details about how OWSM uses the credential store to look up alias names and passwords from the Java keystore, see "[How OWSM Locates Keystore And Key Passwords for the JKS Keystore](#)" on page 3-13.
- Usernames and passwords used for authentication

Suppose, for example, that you have a Web service that accepts a username token for authentication. If you create a Web service client to talk to this Web service, you need to configure the Web service client with a username and password that can be sent to the Web service. You store this username and password in the credential store (using either Fusion Middleware Control or WLST) and assign it a csf key.

For example, the `oracle/wss_username_token_client_policy` policy includes the `csf-key` property, with a default value of `basic.credentials`. To use the `wss_username_token_client_policy`, you should create a new password credential in the CSF using the credential name `basic.credentials`, and the username and password with which the client needs to connect. If you have two Web service clients that use this same client policy, these clients can either share the same password credential, which defaults to `basic.credentials`, or each one can have its own credential. In the latter case, you need to create two password credentials in the CSF, for example `App1.credentials` and `App2.credentials`, for Client1 and Client2 respectively. For Client1, you set the `csf-key` configuration override to `App1.credentials`, and for Client2, you set the `csf-key` property to `App2.credentials`. For more information, see "Overriding Policy Configuration Properties" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*. Note that in both cases, the usernames and passwords must represent valid users in the OPSS identity store.

A password credential can store a username and password. A generic credential can store any credential object.

The CSF configuration is maintained in the `jps-config.xml` file in the `domain-home/config/fmwconfig` directory.

When you configure the OWSM keystore using Fusion Middleware Control, as described in "Configuring the OWSM Keystore Using Fusion Middleware Control" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*, the aliases and passwords that you specify are securely stored in the credential store. If, however, you add other aliases to the keystore, or you need to add authentication credentials for a client, you need to ensure that they are configured and stored in the credential store.

3.8 Understanding Security Policies

WS-SecurityPolicy is part of the Web Services Secure Exchange (WS-SX) set of specifications hosted by OASIS (in addition to WS-SecurityPolicy, the WS-SX technical committee defines two other sets of specifications: WS-Trust and WS-SecureConversation, described later in this chapter).

WS-SecurityPolicy defines a set of security policy assertions used in the context of the WS-Policy framework. WS-SecurityPolicy assertions describe how messages are secured on a communication path. Oracle has contributed to the OASIS WS-SX technical committee several practical security scenarios (a subset of which is provided by OWSM 12c). Each security scenario describes WS-SecurityPolicy policy expressions.

WS-SecurityPolicy *scenarios* describe examples of how to set up WS-SecurityPolicy policies for several security token types described in the WS-Security specification (supporting both WS-Security 1.0 and 1.1). The subset of the WS-SecurityPolicy scenarios supported by OWSM 12c represents the most common customer use cases. Each scenario has been tested in multiple-vendor WS-Security environments.

To illustrate WS-SecurityPolicy, let's use a scenario supported by OWSM: UsernameToken with plain text password. As mentioned earlier, Username token is one of the security tokens specified by WS-Security. This specific scenario uses a policy

that says that a requester must send a password in a Username token to a recipient who has authority to validate that token. The password is a default requirement for the WS-Security Username Token Profile 1.1.

This scenario is only recommended when confidentiality of the password is not an issue, such as a pre-production test scenario with dummy passwords.

Example 3-4 Example of WS-SecurityPolicy

```
<wsp:Policy>
  <sp:SupportingTokens>
    <wsp:Policy>
      <sp:UsernameToken/>
    </wsp:Policy>
  </sp:SupportingTokens>
</wsp:Policy>
```

An example of a message that conforms to the above stated policy is shown below.

Example 3-5 Example of Message Conforming to WS-SecurityPolicy

```
<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope xmlns:soap="...">
  <soap:Header>
    <wsse:Security soap:mustUnderstand="1" xmlns:wsse="...">
      <wsse:UsernameToken>
        <wsse:Username>Marc</wsse:Username>
        <wsse:Password Type="http://docs.oasis open.org...">
          XYZ
        </wsse:Password>
        <wsse:Nonce EncodingType="...#Base64Binary">qB...</wsse:Nonce>
        <wsu:Created>2008-01-02T00:01:03Z</wsu:Created>
      </wsse:UsernameToken>
    </wsse:Security>
  </soap:Header>
  <soap:Body>
    <Oracle xmlns=http://xmlsoap.org/Oracle>
      <text>EchoString</text>
    </Oracle>
  </soap:Body>
</soap:Envelope>
```

The example above contains a <Nonce> element and a <Created> timestamp, which, while optional, are recommended to improve security of requests against replay and other attacks. A nonce is a randomly generated (unique) number. The timestamp can be used to define the amount of time the security token is valid.

3.9 Understanding Security Tokens

Web Services Security (WS-Security) specifies SOAP security extensions that provide confidentiality using XML Encryption and data integrity using XML Signature. WS-Security also includes profiles that specify how to insert different types of binary and XML security tokens in WS-Security headers for authentication and authorization purposes.

Web services security supports the following security tokens:

- Username—defines how a Web service consumer can supply a username as a credential for authentication). For more information, see "[Username Token](#)" on page 3-19

- X.509 certificate—a signed data structure designed to send a public key to a receiving party. For more information, see "[X.509 Certificate](#)" on page 3-19
- Kerberos ticket—a binary authentication and session token. For more information, see "[Kerberos Token](#)" on page 3-19
- Security Assertion Markup Language (SAML) assertion—shares security information over the Internet through XML documents. For more information, see "[SAML Token](#)" on page 3-19

3.9.1 Username Token

The username token carries basic authentication information. The `username-token` element propagates username and password information to authenticate the message.

3.9.2 X.509 Certificate

An X.509 digital certificate is a signed data structure designed to send a public key to a receiving party. A certificate includes standard fields such as certificate ID, issuer's Distinguished Name (DN), validity period, owner's DN, owner's public key, and so on.

Certificates are issued by certificate authorities (CA). A CA verifies an entity's identity and grants a certificate, signing it with the CA's private key. The CA publishes its own certificate which includes its public key.

Each network entity has a list of the certificates of the CAs it trusts. Before communicating with another entity, a given entity uses this list to verify that the signature of the other entity's certificate is from a trusted CA.

3.9.3 Kerberos Token

Kerberos token is a cross-platform authentication and single sign-on system. The Kerberos protocol provides mutual authentication between two entities relying on a shared secret (symmetric keys). Kerberos uses the following terminology:

- A Principal is an identity for a user (i.e., a user is assigned a principal), or an identity for an application offering Kerberos services.
- A Realm is a Kerberos server environment; a Kerberos realm can be a domain name such as EXAMPLE.COM (by convention expressed in uppercase).

Kerberos involves a client, a server, and a trusted party to mediate between them called the Key Distribution Center (KDC). Each Kerberos realm has at least one KDC. KDCs come in different packages based on the operating platform used (for example, on Microsoft Windows, the KDC is a domain service). The Kerberos Token profile of WS-Security allows business partners to use Kerberos tokens in service-oriented architectures.

3.9.4 SAML Token

The Security Assertion Markup Language (SAML) is an open framework for sharing security information over the Internet through XML documents. SAML was designed to address the following:

- Limitations of web browser cookies to a single domain: SAML provides a standard way to transfer cookies across multiple Internet domains.

- Proprietary web single sign-on (SSO): SAML provides a standard way to implement SSO within a single domain or across multiple domains. This functionality is provided by the Oracle Identity Federation product.
- Federation: SAML facilitates identity management (e.g., account linking when a single user is known to multiple web sites under different identities), also supported by Oracle Identity Federation.
- Web Services Security: SAML provides a standard security token (a SAML assertion) that can be used with standard web services security frameworks (e.g., WS-Security) – This is the use of SAML that is particularly relevant to web services security, fully supported by OWSM.
- Identity propagation: SAML provides a standard way to represent a security token that can be passed across the multiple steps of a business process or transaction, from browser to portal to networks of web services, also a feature supported by OWSM.

The SAML framework includes 4 parts:

- Assertions: How you define authentication and authorization information.
- Protocols: How you ask (SAML Request) and get (SAML Response) the assertions you need.
- Bindings: How SAML Protocols ride on industry-standard transport (e.g., HTTP) and messaging frameworks (e.g., SOAP).
- Profiles: How SAML Protocols and Bindings combine to support specific use cases.

In the context of WS-Security, only SAML assertions are used. The protocols and bindings are provided by the WS-Security framework. SAML is widely adopted by the industry, both for browser-based federation and federation enabled by web services flows.

SAML assertions are very popular security tokens within WS-Security because they are very expressive and can help prevent man-in-the-middle and replay attacks.

Typically, a SAML assertion makes statements about a principal (a user or an application). All SAML assertions include the following common information:

- Issuer ID and issuance timestamp
- Assertion ID
- Subject
- Name
- Optional subject confirmation (for example, a public key)
- Optional conditions (under which an assertion is valid)
- Optional advice (on how an assertion was made)

SAML assertions can include three types of statements:

- Authentication statement: issued by an authentication authority upon successful authentication of a subject. It asserts that Subject S was authenticated by Means M at Time T.
- Attribute statement: issued by an attribute authority, based on policies. It asserts that Subject S is associated with Attributes A, B, etc. with values a, b, and so on.

- Authorization decision statement (deprecated in SAML 2.0, now supported by XACML): issued by an authorization authority which decides whether to grant the request by Subject S, for Action A (e.g., read, write, etc.), to Resource R (e.g., a file, an application, a Web service), given Evidence E.

SAML assertions can be embedded (i.e., a SAML assertion can contain another SAML assertion). SAML assertions can be signed (using XML Signature) and/or encrypted (using XML Encryption).

3.10 Understanding Secure Attachments

OWSM policies support two mechanisms to secure attachments: Packaging SOAP messages with attachments (SwA), and Message Transmission Optimization Mechanism (MTOM).

Packaging SOAP messages with attachments (SwA) has become common for any data that cannot be placed inside SOAP Envelope. The primary SOAP message can reference additional entities as attachments or attachments with MIME headers. For more information, see "Securing SwA Attachments" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

Using MTOM, binary content can be sent as a MIME attachment, which reduces the transmission size on the wire. The binary content is semantically part of the XML document. Attaching an MTOM policy ensures that the message is converted to a MIME attachment before it is sent to the Web service or client. See "MTOM Attachment Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager* for more information.

3.11 Understanding Secure Conversation

OWSM implements the Web Services Trust (WS-Trust 1.3) and Web Services Secure Conversation (WS-SecureConversation 1.3) specifications, which together provide secure communication between Web services and their clients.

The Web Services Secure Conversation Language (WS-SecureConversation) specification

(<http://docs.oasis-open.org/ws-sx/ws-secureconversation/v1.4/os/ws-secureconversation-1.4-spec-os.doc>) defines extensions that build on Web Services Security (WS-Security) 1.1 and 1.0 and Web Services Trust Language (WS-Trust) to provide secure communication across one or more messages. Specifically, this specification defines mechanisms for establishing and sharing security contexts or any credentials, and deriving keys from established security contexts (or any shared secret).

OWSM includes policies for which WS-SecureConversation is enabled by default, as described in "Which Policies Support WS-SecureConversation?" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*. You may find that using the preconfigured WS-SecureConversation policies makes your security tasks easier to view at a glance and manage.

In addition, the OWSM security policies described in "Which Policies Support WS-SecureConversation?" include a configuration setting that allows you to enable and configure WS-SecureConversation for that policy.

This section describes the following topics:

- "Why Use WS-SecureConversation" on page 3-22
- "WS-SecureConversation Architecture" on page 3-23

- ["When To Use WS-SecureConversation"](#) on page 3-25
- ["When To Use Re-Authentication"](#) on page 3-26
- ["Using Basic Mode Versus Advanced Mode"](#) on page 3-26
- ["Persistence"](#) on page 3-27

3.11.1 Why Use WS-SecureConversation

There are two primary reasons you might want to use WS-SecureConversation: performance and security.

WS-Security, the standard employed by the OWSM security policies, provides the basic mechanism for securing messages.

However, without WS-SecureConversation, a client using an OWSM security policy (for example, `oracle/wss11_username_with_message_protection`) that exchanges multiple messages must repeatedly authenticate itself and perform expensive asymmetric operations such as key exchanges in each request.

To securely exchange multiple messages, a client and a Web service typically require a security context in which to exchange the messages. WS-SecureConversation provides just such a context. It adds a "handshake" process, which allows a Web service and its client to authenticate to each other and to establish a shared security context. The security context is shared by the client and Web service for the lifetime of a communication session. This context contains a shared secret key that can be used to secure subsequent messages between the client and service, and can improve performance by avoiding repeated key exchanges in multi-message exchange scenarios.

Enabling secure conversation means that there is no need to repeatedly exchange the keys and authenticate each time.

Consider the following sequence:

1. When the first request is made by the client, the handshake happens between the client and the Web service.
2. The client authenticates itself to the service as defined in the bootstrap policy using the WS-Trust protocol.
3. The Web service returns the secure context token (SCT) containing the binary secret that is used for subsequent requests to secure messages during the communication session.

Note: The authentication mechanism required by the Web service does not change, merely the frequency with which the authentication operation is performed.

WS-SecureConversation provides different benefits depending on whether you are using Wss11, Wss10, or SSL OWSM policies, as follows:

- Wss11 — Wss11 scenarios involve one or two asymmetric cryptographic operations in the request depending on the policy. When WS-SecureConversation is in use, the authentication and asymmetric cryptographic operations are done only once at bootstrap time and the subsequent application requests will use the SCT to secure the messages. The SCT uses only symmetric cryptographic operations, which are less expensive.

For example, when a "username with message protection" policy is enabled with WS-SecureConversation, the bootstrap policy uses the username token for authentication and Wss11 for message protection. However, subsequent messages do not involve any authentication and the messages are protected by the SCT.

- Wss10 — Wss10 scenarios involve four asymmetric cryptographic operations in the request and response messages. When WS-SecureConversation is in use, the authentication and asymmetric cryptographic operations are done only once at bootstrap time and the subsequent application requests use the SCT to secure the messages. The SCT uses only symmetric cryptographic operations, which are less expensive.

For example, when a "username with message protection" policy is enabled with WS-SecureConversation, the bootstrap policy uses the username token for authentication and Wss10 for message protection. However, subsequent messages do not involve any authentication and the messages are protected by the SCT.

- SSL — With SSL scenarios, communication happens via SSL throughout the session. At the bootstrap time, authentication happens. Subsequent requests use the SCT to sign the timestamp and no authentication token is sent. Signing the timestamp by SCT proves that the request is sent by the authenticated client.

In these scenarios, SSL is used for message protection; signing the timestamp with the SCT is used for authentication.

For example if WS-SecureConversation is enabled for a "username over SSL" policy, the bootstrap policy uses the username token for authentication and SSL for message protection. However, subsequent messages will also use SSL but will contain a timestamp signed by the SCT instead of the username token.

3.11.1.1 Using WS-SecureConversation With WS-ReliableMessaging

A particularly important use of WS-SecureConversation is to provide security for WS-ReliableMessaging (WS-RM) policies. WS-RM benefits from the use of secure conversation to prevent sequence attacks.

As explained in the WS-ReliableMessaging specification (<http://docs.oasis-open.org/ws-rx/wsrn/v1.2/wsrn.html>), because reliable messaging sequences are expected to exchange a number of messages, it is recommended that a security context be established by using the WS-Trust and WS-SecureConversation mechanisms for protecting sequences.

Therefore, you should attach a WS-SecureConversation-enabled security policy with your WS-RM policy.

3.11.2 WS-SecureConversation Architecture

The WS-SecureConversation specification (<http://docs.oasis-open.org/ws-sx/ws-secureconversation/v1.4/os/ws-secureconversation-1.4-spec-os.doc>) defines extensions that build on Web Services Security (WS-Security) and Web Services Trust Language (WS-Trust).

The specification is the best source of information about the WS-SecureConversation architecture, features, and functions.

At a minimum, you should be familiar with the following concepts:

- WS-Trust — As described in Web Services Trust Language (WS-Trust) in *Understanding WebLogic Web Services for Oracle WebLogic Server*, the Web Services Trust Language (WS-Trust) specification defines extensions that build on Web

Services Security (WS-Security) 1.1 and 1.0 to provide a framework for requesting and issuing security tokens, and to broker trust relationships.

- Security Context — A security context is an abstract concept that refers to an established authentication state and negotiated key(s) that may have additional security-related properties.
- Security Context Token — A security context token (SCT) is a representation of the security context abstract concept, which allows a context to be named by a URI and used with WS-Security. Once the context and secret have been established (authenticated), you can then compute derived keys for each key usage in the secure context.
- Derived Keys — As described in the WS-SecureConversation specification (<http://docs.oasis-open.org/ws-sx/ws-secureconversation/v1.4/os/ws-secureconversation-1.4-spec-os.doc>), "A security context token implies or contains a shared secret. This secret MAY be used for signing and/or encrypting messages, but it is recommended that derived keys be used for signing and encrypting messages associated only with the security context."

Again as described in the WS-SecureConversation specification, "Once the context and secret have been established (authenticated), the mechanisms described in Derived Keys can be used to compute derived keys for each key usage in the secure context."

Derived keys are useful for message protection. Instead of using the same SCT across multiple requests, a different key derived from the SCT is used in each request, which improves overall security.

When you enable WS-SecureConversation for a policy, OWSM uses derived keys by default for WSS10 and WSS11. (For SSL policies, message protection is done using SSL and a derived key is not necessary.)

- Session management — OWSM maintains the client and server secure conversation session information based on a computed Session ID.

On the Web server side, the Session ID is maintained based on the port used by the Web service.

Client sessions are expressed by the term "reference," which is similar in concept to a client port/binding that enables message communication.

In the WS-SecureConversation implementation, each client reference is a separate WS-SecureConversation session. From the perspective of a Web service client request, this leads to the following outcomes:

- Multiple requests can belong to the same reference.
- All the requests with the same Session ID belong to the same session.
- The state for which the Session ID is valid depends on the re-authentication setting.

OWSM computes the Session ID at runtime for each message, and associates one or more requests to a session. OWSM uses user credentials, service information, and policy and configuration data to compute the Session ID.

The Session ID is especially important when used with Oracle WS-RM policies, where for security and performance reasons multiple messages in an RM session are protected by the same secure conversation session.

- Inner and outer policies — In the OWSM implementation of WS-SecureConversation, a secure conversation policy has actually two policies:

inner and outer. The bootstrap (inner) policy is used to obtain the token and establish the handshake between the client and the Web service. The outer policy is used for application messages when making requests with the token.

The message security settings for the outer policy are obtained from the original OWSM WS-Security policy, such as `wss11_username_with_message_protection`. The message security settings for the inner policy are then derived from the outer policy.

In most cases, you do not need to be concerned with the details of the inner and outer policies, as OWSM handles this on your behalf. However, the OWSM WS-SecureConversation implementation provides an Advanced setting that provides additional control, as described in ["Using Basic Mode Versus Advanced Mode"](#) on page 3-26.

- Re-authentication — OWSM includes a `re-authenticate` control that indicates whether to create a separate session for each user or to allow users to share the same session. A user is authenticated only once whether `re-authenticate` is true or not.

There is one supported use case in which the user ID might be different for each application message and therefore needs to be authenticated in each message during the WS-SecureConversation session: ID propagation with SAML sender vouches.

Re-authentication allows multiple users to share a session. In this case, the authentication token is sent in each request because multiple users share the session. However, there is no need to exchange keys and asymmetric operations (sign, encrypt) are not performed in subsequent requests.

The state for which the Session ID is valid depends on the re-authentication setting:

- If `re-authenticate` is false, on the client side the Session ID is maintained for a single reference for a given user.

On the server side, the Session ID is maintained based on the port used by the Web service.

- If `re-authenticate` is true, on the client side the Session ID is maintained for a single reference, which may involve multiple users.

On the server side, the Session ID is maintained based on the port used by the Web service.

3.11.3 When To Use WS-SecureConversation

You should consider using WS-SecureConversation in the following scenarios:

- You are using any OWSM WS-RM policy.
- Your Web service client is protected with an OWSM security policy (for example, `oracle/wss11_username_with_message_protection`) and frequently exchanges multiple messages.

When a Web service client or service are secured by OWSM and expect to be involved in multiple message exchanges, it makes sense to enable WS-SecureConversation. Enabling WS-SecureConversation provides better performance because the SCT secures subsequent messages between the client and service and you do not incur the overhead of repeated authentication and public key crypto operations.

Note: For ID propagation use cases, WS-SecureConversation provides a performance benefit mainly for message protection because the authentication token is sent in each message during the session at the expense of performance.

Consider the following scenarios in which multiple message exchanges might happen and WS-SecureConversation might be useful:

- One to One — In this case, a client application invokes a particular Web service multiple times on behalf of a single user.
- One to One with re-authenticate=true (identity propagation) — In this case, a client application invokes a particular Web service multiple times. However, a different identity may need to be passed to the Web service in each subsequent request.

A single secure conversation session is created for all users.

3.11.4 When To Use Re-Authentication

You can enable the re-authenticate control only in the case of ID propagation with SAML sender vouches policies. You should use re-authentication when the user ID might be different for each application message. In this situation, the user is authenticated in each message.

The bootstrap is done using the client identity and the end user identity is passed in all application requests to the service. WS-SecureConversation provides a benefit mainly for message protection, because the authentication token is sent in each message during the session at the expense of performance.

By default, the re-authenticate control is not set with WS-SecureConversation, and you can enable it only when WS-SecureConversation is also enabled.

3.11.5 Using Basic Mode Versus Advanced Mode

In the OWSM implementation of WS-SecureConversation, a secure conversation policy has actually two policies: inner and outer.

The bootstrap (inner) policy is used to obtain the token and establish the handshake between the client and the Web service. The outer policy is used for application messages when making requests with the token.

The message security settings for the outer policy are obtained from the original OWSM WS-Security policy, such as `oracle/wss11_username_with_message_protection`. The message security settings for the inner policy are then derived from the outer policy.

Therefore in most cases, you do not need to be concerned with the details of the inner and outer policies, as OWSM handles this on your behalf. However, the OWSM WS-SecureConversation implementation provides the Advanced screen shown in [Figure 3-2](#) that provides additional control.

Figure 3–2 Secure Conversation Advanced Screen

The screenshot shows the 'Secure Conversation' configuration screen. It is divided into three main sections:

- Secure Conversation:** Contains checkboxes for 'Enabled' (checked), 'Re-Authenticate' (unchecked), and 'Derived Keys' (checked). Below this, 'Version' is set to '1.3', and 'Client Entropy' and 'Server Entropy' are both checked.
- Bootstrap Message Security:** Offers two radio button options: 'Inherit From Application Setting' (unchecked) and 'Use Independent Setting' (checked). Under 'Use Independent Setting', there are four checkboxes: 'Algorithm Suite' (set to 'Basic256' with an edit icon), 'Include Timestamp' (checked), 'Encrypt Signature' (unchecked), and 'Encrypt Signature' (checked).
- Message Security:** Similar to the Bootstrap section, it has 'Algorithm Suite' (set to 'Basic128' with an edit icon), 'Include Timestamp' (checked), and 'Encrypt Signature' (unchecked).

The following Advanced options are available:

- Client entropy
- Server entropy
- Derived Keys
- Bootstrap Message Security:
 - Inherit From Application Setting
 - Use Independent Setting:
 - * Algorithm Suite
 - * Encrypt Signature
 - * Signature Confirmation
 - * Include Timestamp

See "Configure Secure Conversation: Advanced Management" in *Developing Oracle Infrastructure Web Services* for additional information on setting these options.

3.11.6 Persistence

This section describes how to manage persistence for a session.

3.11.6.1 Default Domain-Wide Persistence Implementation

OWSM includes a default domain-wide persistence implementation that supports the Coherence Cluster and in-memory persistence providers.

The Coherence persistence provider is the default when running in WebLogic Server, for both the Web service client and Web service. Otherwise, the in-memory persistence provider is the default.

The level of persistence granularity for session recovery is the session object.

This persistence implementation is enabled by default and does not require any configuration.

3.11.6.2 Client- and Web Service-Specific Persistence Implementation

Each client and Web service can specify one or more (one per port) persistence providers, which can be either the Coherence provider or the in-memory provider.

To do this, use one of the mechanisms described in "Configure Persistence".

3.12 Understanding the Kerberos Protocol

Kerberos is an authentication protocol that enables computers (clients and servers) communicating over a non-secure network to prove their identity to one another in a secure manner, with the help of a trusted third party. In Kerberos, this trusted third party is the Key Distribution Center (KDC), which contains key information for clients and servers, called principals. The KDC consists of two components:

- The Authentication Service (AS), which authenticates a principal with the KDC
- The Ticket Granting Service (TGS), which provides authenticated principals with tickets they can use to request services from other principals in the KDC.

OWSM supports MIT Kerberos and Microsoft Active Directory as the KDC. For information about using MIT Kerberos, see "Using MIT Kerberos" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*. For information about using Microsoft Active Directory, see "Using Microsoft Active Directory with Key Distribution Center" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

Here are the high-level steps involved when Kerberos is used for message security between a client principal and a server principal:

1. AS-REQ (request to Authentication Service): The client begins the authentication process by sending the user ID to the AS.
2. AS-REP (reply from Authentication Service): The AS responds with:
 - A client/TGS session key, encrypted using a hash of the user's password from the KDC
 - A Ticket Granting Ticket (TGT), encrypted using the secret key of the TGS.
3. TGS-REQ (request to Ticket Granting Service): To begin communicating with services, the client first sends the following to the TGS:
 - The TGT it received from the AS
 - The ID of the requested service
 - An authenticator, encrypted using the client/TGS session key from the AS
4. TGS-REP (reply from Ticket Granting Service): The TGS decrypts the TGT using its secret key, extracts the client/TGS session key from the decrypted TGT, and then uses this session key to decrypt the authenticator. It then responds with:
 - A client/server session key, encrypted using the client/TGS session key
 - A Service Ticket (ST), encrypted using the service's secret key
5. AP-REQ (request to application): After receiving the reply from the TGS, the client initiates contact with the service by sending it:
 - The ST it received from the TGS
 - A new authenticator, encrypted using the client/server session key from the TGS
6. AP-REP (reply from application): The service decrypts the ST using its secret key, extracts the client/server session key, and then uses this session key to decrypt the authenticator. It then extracts the timestamp from the decrypted authenticator, adds one to it, and sends this value back to the client after encrypting it using the client/server session key.

7. The client decrypts the confirmation and checks whether the timestamp is correctly updated. If so, the client can trust the server and can begin issuing service requests.

For information on configuring OWSM to support the Kerberos protocol, see "Configuring the Kerberos Login Module" and "Configuring Kerberos Tokens" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

3.12.1 Credential Delegation in Kerberos

Sometimes, a service needs to access another service or server in order to complete a client request. In order to establish such a connection, Kerberos requires the first service to be authenticated to the second service or server using the client's user account and authority level. The mechanism Kerberos provides to meet this requirement is called credential delegation.

A common way to provide credential delegation in Kerberos is through the use of the FORWARDABLE and FORWARDED flags in Kerberos tickets, a technique called forwarded TGT. Here are a high-level steps involved in using forwarded TGT:

1. The user requests the KDC for a TGT with forwardable flag set (Forwardable TGT) by setting the KDC option named FORWARDABLE in the initial AS-REQ.
2. The client requests a FORWARDED ticket by presenting this forwardable TGT to the TGS. The client also sets the KDC Option named FORWARDED in the request (TGS_REQ) in addition to providing a set of service addresses for the new ticket.
3. More such tickets (with the FORWARDED flag set) can be obtained from the KDC by providing the FORWARDED ticket obtained in Step 2.

Here are more detailed steps that specifies the message sequence:

1. The user authenticates to the KDC by sending a KRB_AS_REQ message and requests a forwardable TGT.
2. The KDC returns a forwardable TGT in the KRB_AS_REP message.
3. The user requests a forwarded TGT based on the forwardable TGT from Step 2. This is done by the KRB_TGS_REQ message.
4. The KDC returns a forwarded TGT for the user in the KRB_TGS_REP message.
5. The user makes a request for a service ticket to Service 1 using the TGT returned in Step 2. This is done by the KRB_TGS_REQ message.
6. The ticket granting service (TGS) returns the service ticket in a KRB_TGS_REP message.
7. The user makes a request to Service 1 by sending a KRB_AP_REQ message, presenting the service ticket, the forwarded TGT, and the session key for the forwarded TGT.
8. To fulfill the user's request, Service 1 needs to invoke Service 2 to perform some action on behalf of the user. Service 1 uses the forwarded TGT of the user and sends that in the KRB_TGS_REQ message to the KDC, asking for a ticket to Service 2 in the name of the user.
9. The KDC returns a ticket for Service 2 to Service 1, in a KRB_TGS_REP message, along with a session key that Service 1 can use. The ticket identifies the client as the user, and not Service 1.
10. Service 1 makes a request to Service 2 using a KRB_AP_REQ, acting as the user.
11. Service 2 performs the task and responds.

12. Service 1 responds to the user's request with the response it gets from Service 2.

For information on configuring OWSM to use credential delegation, see "Configuring Credential Delegation" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

3.12.2 Kerberos and SPNEGO

SPNEGO (Simple and Protected GSS-API Negotiation Mechanism) is a standard that enables a client and a service to negotiate a method to use for authentication. Because SPNEGO uses HTTP headers to perform the negotiation, it is especially useful in a cross-platform context such as the web, where SOAP and REST endpoints that use HTTP are common.

When Kerberos is used in SPNEGO negotiation, the Kerberos token is wrapped in the HTTP header under the auth-scheme Negotiate. The WWW-Authenticate and Authorization headers are used to communicate the SPNEGO token between the client and the service, as follows:

1. The client requests access to a protected service on the server without any Authorization header.
2. Since there is no Authorization header in the request, server responds with the status code 401 (Unauthorized) and the WWW-Authenticate header set to Negotiate.
3. The client uses the user credentials to obtain the Kerberos token and then sends it to the server in the Authorization header of the new request. For example, Authorization: Negotiate a8742100000492aa874209....
4. The server decodes the token in the Authorization header. If the context is not complete (as in the case of Mutual Authentication), the server responds with a 401 status code and a WWW-Authenticate header containing the decoded data. For example, WWW-Authenticate: Negotiate 74900a2a....
5. The client decodes this data and sends new data back to the server. This cycle continues until the security context is established.

For information on configuring OWSM to use Kerberos with SPNEGO, see "Configuring Kerberos With SPNEGO Negotiation" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

3.12.3 Kerberos and WS-SecureConversation Derived Keys

The Web Services Secure Conversation (WS-SecureConversation) specification includes a feature called **derived keys**, which enables parties that have already authenticated to each other to use a common secret to derive additional keys for various uses, such as signing and encrypting messages. Moreover, the WS-SecureConversation specification defines two types of derived keys:

- Explicit derived keys, which use the `wsc:DerivedKeyToken` element to contain the token information. The `ds:KeyInfo` element then contains a reference to this information.
- Implicit derived keys, which include the token information directly in the `ds:KeyInfo` element.

When using Kerberos in a WS-SecureConversation context, you can configure OWSM to use derived keys by enabling the Use Derived Keys option in the OWSM assertions for Kerberos.

3.13 Understanding Web Services Addressing

The Web Services Addressing (WS-Addressing) specification (<http://www.w3.org/TR/ws-addr-core/>) provides transport-neutral mechanisms to address Web services and messages. In particular, the specification defines a number of XML elements used to identify Web service endpoints and to secure end-to-end endpoint identification in messages.

SOAP does not provide a standard way to specify where a message is going or how responses or faults are returned. WS-Addressing provides an XML framework for identifying web services endpoints and for securing end-to-end endpoint identification in messages.

A Web service endpoint is a resource (such as an application or a processor) to which web services messages are sent.

The following is an example using WS-Addressing (`wsa` is the namespace for WSAddressing):

Example 3–6 Example of WS-Addressing

```
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
  <S:Header>
    <wsa:MessageID>http://example.com/xyz-abcd-123</wsa:MessageID>
    <wsa:ReplyTo>
      <wsa:Address>http://example.myClient1</wsa:Address>
    </wsa:ReplyTo>
```

WS-Addressing is transport-independent; that is, the request may be over JMS and the response over HTTP. WS-Addressing is used with other WS-* specifications, such as WS-Policy.

3.14 Understanding Web Services Trust

The WS-Trust 1.3 specification (<http://docs.oasis-open.org/ws-sx/ws-trust/v1.3/ws-trust.html>) defines extensions to WS-Security that provide a framework for requesting and issuing security tokens, and to broker trust relationships. WS-Trust extensions provide methods for issuing, renewing, and validating security tokens. To secure communication between a Web service client and a Web service, the two parties must exchange security credentials. As defined in the WS-Trust specification, these credentials can be obtained from a SecurityTokenService (STS), which acts as trust broker.

There are multiple scenarios in which you might consider using an STS, including:

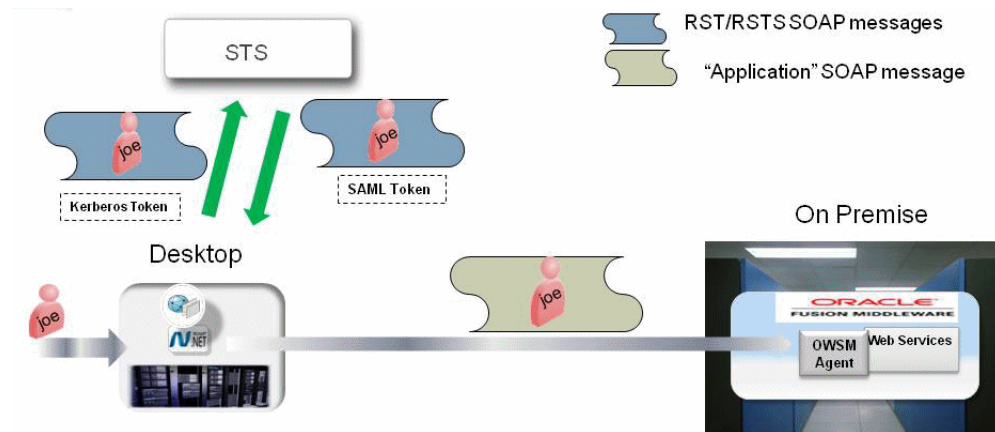
- **Token Exchange/Conversion** — Assume that you need to exchange one kind of token for another type of token. For example, if the client has a Kerberos token but the Web service requires a SAML token. You can use the STS to exchange the Kerberos token for a SAML token.
- **Federation** — Identity federation allows a user to consolidate the many local identities he has configured among multiple service providers. With a federated identity, the individual can log in at one service provider site and move to an affiliated service provider site without having to re-authenticate or re-establish his identity.

For example, you might use the STS to map a client user name to the user name expected by the Web service.

- **Centralized Trust** — The STS is trusted by both the Web service client and the Web service. You use this trust to provide interoperable security tokens.

Consider the token exchange scenario shown in [Figure 3–3](#). In this scenario, a customer has a desktop application (for example, a .NET Web service) that is talking to a backend Web service that can accept a SAML token.

Figure 3–3 STS Token Exchange



In [Figure 3–3](#) user "joe" logs into his desktop and a Kerberos ticket is created. When the user opens the desktop application and performs an operation, this results in a backend Web service call and we want to propagate the identity of "joe" to the backend application. However the token we have is a Kerberos token on the client side and the backend Web Service only accepts a SAML token. You can use an STS to do a token conversion or token exchange.

For information on configuring OWSM to support the Kerberos protocol, see "Configuring WS-Trust" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

3.15 Understanding Web Services ReliableMessaging

WS-ReliableMessaging makes message exchanges reliable. It ensures that messages are delivered reliably between distributed applications regardless of software component, system, or network failures. Ordered delivery is assured and automatic retransmission of failed messages does not have to be coded by each client application.

Consider using reliable messaging if your Web service is experiencing the following problems:

- network failures or dropped connections
- messages are lost in transit
- messages are arriving at their destination out of order

WS-ReliableMessaging considers the source and destination of a message to be independent of the client/server model. That is, the client and the server can each act simultaneously as both a message source and destination on the communications path.

For information on WS-ReliableMessaging (WS-RM), see "Using Web Services Reliable Messaging" in *Developing Oracle Infrastructure Web Services*.

Web Service Security Standards

This appendix summarizes the security standards for Oracle Infrastructure Web Services. For a complete list of standards supported for Oracle Infrastructure Web services, see "Supported Standards" in *Developing Oracle Infrastructure Web Services*.

Security standards are implemented in non-XML frameworks at the transport level, and in XML frameworks at the application level.

Table A-1 lists the standards that are key to providing secure and manageable environments at both the transport and application levels.

For a complete list and descriptions of standards for WebLogic Web services, see "Features and Standards Supported by WebLogic Web Services" in *Understanding WebLogic Web Services for Oracle WebLogic Server*.

Table A-1 Web Services Standards and Specification URLs

Standard	Description and Specification URL
Web Services Interoperability Organization—Basic Security Profile	<p>Oracle considers interoperability of Web services platforms to be more important than providing support for all possible edge cases of the Web services specifications. Oracle complies with the following specification from the Web Services Interoperability Organization and considers it to be the baseline for Web services interoperability.</p> <p>For more information, see:</p> <ul style="list-style-type: none"> Basic Security Profile 1.0 Specification: http://www.ws-i.org/Profiles/BasicSecurityProfile-1.0.html
Transport Layer Security—SSL	<p>Secure Sockets Layer (SSL), also known as Transport Layer Security (TLS), is the most widely used transport-layer data-communication protocol.</p> <p>For more information, see:</p> <ul style="list-style-type: none"> "Understanding Transport-level and Application-level Security" on page 3-3 SSL 3.0: http://tools.ietf.org/html/rfc6101
XML Encryption (Confidentiality)	<p>The XML encryption specification describes a process for encrypting data and representing the result in XML.</p> <p>For more information, see:</p> <ul style="list-style-type: none"> "Message Encryption" on page 3-6 XML Encryption Syntax and Processing Specification: http://www.w3.org/TR/xmlenc-core

Table A-1 (Cont.) Web Services Standards and Specification URLs

Standard	Description and Specification URL
XML Signature (Integrity, Authenticity)	<p>The XML Signature specification describes signature processing rules and syntax. XML Signature binds the sender's identity (or "signing entity") to an XML document. The document is signed using the sender's private key; the signature is verified using the sender's public key.</p> <p>For more information, see:</p> <ul style="list-style-type: none">■ "Message Signing (XML Signature)" on page 3-7■ XML Signature WG Specification: http://www.w3.org/Signature/
WS-Security	<p>Web Services Security (WS-Security) specifies SOAP security extensions that provide confidentiality using XML Encryption and data integrity using XML Signature. WS-Security also includes profiles that specify how to insert different types of binary and XML security tokens in WS-Security headers for authentication and authorization purposes.</p> <p>For more information, see:</p> <ul style="list-style-type: none">■ "Understanding Security Policies" on page 3-17■ OASIS Web Services Security (WSS) TC Specification: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss
Username Token	<p>The username token carries basic authentication information. The username-token element propagates username and password information to authenticate the message.</p> <p>For more information, see:</p> <ul style="list-style-type: none">■ "Username Token" on page 3-19■ Web Services Security UsernameToken Profile 1.0 Specification: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0.pdf
X.509 Certificate	<p>An X.509 digital certificate is a signed data structure designed to send a public key to a receiving party. A certificate includes standard fields such as certificate ID, issuer's Distinguished Name (DN), validity period, owner's DN, owner's public key, and so on.</p> <p>For more information, see:</p> <ul style="list-style-type: none">■ "X.509 Certificate" on page 3-19■ Web Services Security X.509 Certificate Token Profile Specification: http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0.pdf
Kerberos Token	<p>Kerberos token is a cross-platform authentication and single sign-on system. The Kerberos protocol provides mutual authentication between two entities relying on a shared secret (symmetric keys).</p> <p>For more information, see:</p> <ul style="list-style-type: none">■ "Kerberos Token" on page 3-19■ Web Services Security Kerberos Token Profile 1.1 Specification: http://www.oasis-open.org/committees/download.php/16788/wss-v1.1-spec-os-KerberosTokenProfile.pdf
SAML Token	<p>The Security Assertion Markup Language (SAML) is an open framework for sharing security information over the Internet through XML documents.</p> <p>For more information, see:</p> <ul style="list-style-type: none">■ "SAML Token" on page 3-19■ Web Services Security SAML Token Profile Specification: http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.0.pdf

Table A-1 (Cont.) Web Services Standards and Specification URLs

Standard	Description and Specification URL
WS-Policy	<p>A Web service provider may define conditions (or policies) under which a service is to be provided. The WS-Policy framework enables one to specify policy information that can be processed by web services applications, such as Oracle WSM.</p> <p>For more information, see:</p> <ul style="list-style-type: none">■ "Understanding Policies" on page 2-4■ Web Services Policy 1.2 - Framework (WS-Policy) Specification: http://www.w3.org/Submission/WS-Policy/
WS-SecurityPolicy	<p>WS-SecurityPolicy defines a set of security policy assertions used in the context of the WS-Policy framework. WS-SecurityPolicy assertions describe how messages are secured on a communication path.</p> <p>For more information, see:</p> <ul style="list-style-type: none">■ "Understanding Security Policies" on page 3-17■ WS-SecurityPolicy 1.2 Specification: http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/ws-securitypolicy-1.2-spec-os.html
Web Services Addressing (WS-Addressing)	<p>SOAP does not provide a standard way to specify where a message is going or how responses or faults are returned. WS-Addressing provides an XML framework for identifying web services endpoints and for securing end-to-end endpoint identification in messages.</p> <p>For more information, see:</p> <ul style="list-style-type: none">■ "Understanding Web Services Addressing" on page 3-31■ Web Services Addressing 1.0 - Core Specification: http://www.w3.org/TR/ws-addr-core/
WS-Trust	<p>Defines extensions to WS-Security that provide a framework for requesting and issuing security tokens, and to broker trust relationships. WS-Trust extensions provide methods for issuing, renewing, and validating security tokens.</p> <p>For more information, see:</p> <ul style="list-style-type: none">■ "Understanding Web Services Trust" on page 3-31■ WS-Trust 1.3 Specification: http://docs.oasis-open.org/ws-sx/ws-trust/v1.3/ws-trust.html
WS-ReliableMessaging	<p>WS-ReliableMessaging (WS-RM) defines a framework for identifying and managing the reliable delivery of messages between Web services endpoints.</p> <p>For more information, see:</p> <ul style="list-style-type: none">■ "Using Web Services Reliable Messaging" in <i>Oracle Fusion Middleware Developer's Guide for Oracle Infrastructure Web Services</i>■ Web Services Reliable Messaging Specification: http://docs.oasis-open.org/ws-rx/wsrp/200702/wsrp-1.1-spec-os-01-e1.html
WS-SecureConversation	<p>The Web Services Secure Conversation Language (WS-SecureConversation) is built on top of the WS-Security and WS-Policy models to provide secure communication between services. This specification defines mechanisms for establishing and sharing security contexts, and deriving keys from security contexts, to enable a secure conversation</p> <p>For more information, see:</p> <ul style="list-style-type: none">■ WS-SecureConversation 1.4 Specification: http://docs.oasis-open.org/ws-sx/ws-secureconversation/v1.4/ws-secureconversation.html
