Oracle® Fusion Middleware

Interoperability Solutions Guide for Oracle Web Services Manager

12*c* (12.2.1)

E57783-01

October 2015

Documentation for software developers that describes how to implement the most common OWSM interoperability scenarios.

Oracle Fusion Middleware Interoperability Solutions Guide for Oracle Web Services Manager, 12*c* (12.2.1)

E57783-01

Primary Author: Oracle Corporation

Contents

6 Interoperability with Microsoft WCF/.NET 4.5 Security Environments

7 Interoperability with Oracle Service Bus 10*g* Security Environments

8 Interoperability with Axis 1.4 and WSS4J 1.5.8 Security Environments

9 Interoperability with Oracle GlassFish Server Release 3.0.1

Preface

This preface describes the document accessibility features and conventions used in this guide—*Oracle Fusion Middleware Interoperability Solutions Guide for Oracle Web Services Manager*.

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 that have purchased support 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.

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.
italic	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. This document is the new edition of the formerly titled *Interoperability Guide for Oracle Web Services Manager*.

New and Changed Features for 12*c* (12.2.1)

Minor updates, such as fixes or corrections, were made to this document.

New and Changed Features for 12*c* (12.1.3)

Oracle JDeveloper 12*c* (12.1.3) includes the following new and changed features for this document:

- Chapter 5, "Interoperability with Microsoft WCF/.NET 3.5 Security Environments," now documents enabling secure conversation for the following interoperability scenario: "Username Token Over SSL" on page 5-14 and "Username Token With Message Protection (WS-Security 1.1)" on page 5-6. It also now documents support an additional policy in the following scenario: "WCF/.NET 3.5 Client with Microsoft Active Directory Federation Services 2.0 (ADFS 2.0) STS" on page 5-31.
- A new chapter has been added: Chapter 6, "Interoperability with Microsoft WCF/.NET 4.5 Security Environments," that documents the new support for Microsoft WCF/.NET 4.5.

New and Changed Features for 12*c* (12.1.2)

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

- The following Microsoft WCF/.NET 3.5 security environment interoperability scenarios have been added for this release:
  - "Kerberos with Message Protection Using Derived Keys" on page 5-25
  - "Kerberos with SPNEGO Negotiation" on page 5-29
  - "Kerberos with SPNEGO Negotiation and Credential Delegation" on page 5-30

Other Significant Changes in this Document for Release 12*c* (12.1.3)

For 12*c* (12.1.3), this guide has been reformatted to improve readability.

1 Overview of OWSM Interoperability

This guide describes interoperability of Oracle Web Services Manager (OWSM) with various security stacks.

Each chapter includes the following information:

- Overview of each security stack
- An explanation of the usage scenarios

For details regarding limitations and known problems, see "Web Services" in *Release Notes for Oracle Fusion Middleware Infrastructure*.

For definitions of unfamiliar terms found in this and other books, see the Glossary.

1.1 About OWSM Policies

You attach *OWSM policies* to web service endpoints. Each policy consists of one or more *assertions*, defined at the domain-level, that define the security requirements. A set of predefined policies and assertions are provided out-of-the-box.

For more details about the predefined policies, see "Predefined Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

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

1.2 OWSM Interoperability Scenarios

Table 1–1 describes the most common OWSM interoperability scenarios.

Table 1–1 Common OWSM Interoperability Scenarios

Security Stack	OWSM Policies	Interoperability Scenario
OWSM 10*g*	`oracle/wss10_message_protection_service_policy` `oracle/wss10_message_protection_client_policy`	"Anonymous Authentication with Message Protection (WS-Security 1.0)" on page 2-4
OWSM 10*g*	`oracle/wss10_username_token_with_message_protection_service_policy` `oracle/wss10_username_token_with_message_protection_client_policy`	"Username Token with Message Protection (WS-Security 1.0)" on page 2-6

Table 1–1 (Cont.) Common OWSM Interoperability Scenarios

Security Stack	OWSM Policies	Interoperability Scenario
OWSM 10*g*	oracle/wss10_saml_token_with_message_protection_service_policy oracle/wss10_saml_token_with_message_protection_client_policy	"SAML Token (Sender Vouches) with Message Protection (WS-Security 1.0)" on page 2-8
OWSM 10*g*	oracle/wss10_x509_token_with_message_protection_service_policy oracle/wss10_x509_token_with_message_protection_client_policy	"Mutual Authentication with Message Protection (WS-Security 1.0)" on page 2-11
OWSM 10*g*	oracle/wss_username_token_over_ssl_service_policy oracle/wss_username_token_over_ssl_client_policy	"Username Token Over SSL" on page 2-13
OWSM 10*g*	oracle/wss_saml_token_over_ssl_service_policy oracle/wss_saml_token_over_ssl_client_policy	"SAML Token (Sender Vouches) Over SSL (WS-Security 1.0)" on page 2-15
OC4J 10*g*	oracle/wss10_message_protection_service_policy oracle/wss10_message_protection_client_policy	"Anonymous Authentication with Message Protection (WS-Security 1.0)" on page 3-3
OC4J 10*g*	oracle/wss10_username_token_with_message_protection_service_policy oracle/wss10_username_token_with_message_protection_client_policy	"Username Token with Message Protection (WS-Security 1.0)" on page 3-6
OC4J 10*g*	oracle/wss10_saml_token_with_message_protection_service_policy oracle/wss10_saml_token_with_message_protection_client_policy	"SAML Token (Sender Vouches) with Message Protection (WS-Security 1.0)" on page 3-10
OC4J 10*g*	oracle/wss10_x509_token_with_message_protection_service_policy oracle/wss10_x509_token_with_message_protection_client_policy	"Mutual Authentication with Message Protection (WS-Security 1.0)" on page 3-13
OC4J 10*g*	oracle/wss_username_token_over_ssl_service_policy OR oracle/wss_saml_or_username_token_over_ssl_service_policy oracle/wss_username_token_over_ssl_client_policy	"Username Token Over SSL" on page 3-17
OC4J 10*g*	oracle/wss_saml_token_over_ssl_service_policy OR oracle/wss_saml_or_username_token_over_ssl_service_policy oracle/wss_saml_token_over_ssl_client_policy	"SAML Token (Sender Vouches) Over SSL (WS-Security 1.0)" on page 3-20

Table 1–1 (Cont.) Common OWSM Interoperability Scenarios

Security Stack	OWSM Policies	Interoperability Scenario
Oracle WebLogic Server 12*c*	oracle/wss11_username_token_with_message_protection_service_policy oracle/wss11_username_token_with_message_protection_client_policy	"Username Token With Message Protection (WS-Security 1.1)" on page 4-6
Oracle WebLogic Server 12*c*	oracle/wss11_username_token_with_message_protection_service_policy oracle/wss11_username_token_with_message_protection_client_policy	"Username Token With Message Protection (WS-Security 1.1) and MTOM" on page 4-9
Oracle WebLogic Server 12*c*	oracle/wss10_username_token_with_message_protection_service_policy oracle/wss10_username_token_with_message_protection_client_policy	"Username Token With Message Protection (WS-Security 1.0)" on page 4-10
Oracle WebLogic Server 12*c*	oracle/wss_username_token_over_ssl_service_policy	"Username Token Over SSL" on page 4-12
Oracle WebLogic Server 12*c*	oracle/wss_username_token_over_ssl_service_policy	"Username Token Over SSL with MTOM" on page 4-13
Oracle WebLogic Server 12*c*	oracle/wss_saml_token_over_ssl_service_policy	"SAML Token (Sender Vouches) Over SSL" on page 4-13
Oracle WebLogic Server 12*c*	oracle/wss11_saml20_token_with_message_protection_service_policy oracle/wss11_saml20_token_with_message_protection_client_policy	"SAML Token (Sender Vouches) Over SSL with MTOM" on page 4-15
Oracle WebLogic Server 12*c*	oracle/wss11_saml20_token_with_message_protection_service_policy oracle/wss11_saml20_token_with_message_protection_client_policy	"SAML Token 2.0 (Sender Vouches) With Message Protection (WS-Security 1.1)" on page 4-16
Oracle WebLogic Server 12*c*	oracle/wss11_saml_token_with_message_protection_service_policy oracle/wss11_saml_token_with_message_protection_client_policy	"SAML Token (Sender Vouches) with Message Protection (WS-Security 1.1)" on page 4-20
Oracle WebLogic Server 12*c*	oracle/wss11_saml_token_with_message_protection_service_policy oracle/wss11_saml_token_with_message_protection_client_policy	"SAML Token (Sender Vouches) with Message Protection (WS-Security 1.1) and MTOM" on page 4-24
Oracle WebLogic Server 12*c*	oracle/wss10_saml_token_with_message_protection_service_policy oracle/wss10_saml_token_with_message_protection_client_policy	"SAML Token (Sender Vouches) with Message Protection (WS-Security 1.0)" on page 4-25
Oracle WebLogic Server 12*c*	oracle/wss10_x509_token_with_message_protection_service_policy oracle/wss10_x509_token_with_message_protection_client_policy	"Mutual Authentication with Message Protection (WS-Security 1.0)" on page 4-29
Oracle WebLogic Server 12*c*	oracle/wss11_x509_token_with_message_protection_service_policy oracle/wss11_x509_token_with_message_protection_client_policy	"Mutual Authentication with Message Protection (WS-Security 1.1)" on page 4-32

Table 1–1 (Cont.) Common OWSM Interoperability Scenarios

Security Stack	OWSM Policies	Interoperability Scenario
Microsoft WCF/.NET 3.5	oracle/wsmtom_policy	"Message Transmission Optimization Mechanism (MTOM)" on page 5-3
Microsoft WCF/.NET 3.5	oracle/wss11_username_token_with_message_protection_service_policy OR oracle/wss11_saml_or_username_token_with_message_protection_service_policy oracle/wss11_username_token_with_message_protection_client_policy	"Username Token With Message Protection (WS-Security 1.1)" on page 5-6
Microsoft WCF/.NET 3.5	oracle/wss_saml_or_username_token_over_ssl_service_policy OR oracle/wss_username_token_over_ssl_service_policy	"Username Token Over SSL" on page 5-14
Microsoft WCF/.NET 3.5	oracle/wss11_x509_token_with_message_protection_service_policy oracle/wss11_x509_token_with_message_protection_client_policy	"Mutual Authentication with Message Protection (WS-Security 1.1)" on page 5-17
Microsoft WCF/.NET 3.5	oracle/wss11_kerberos_with_message_protection_service_policy	"Kerberos with Message Protection" on page 5-22
Microsoft WCF/.NET 3.5	wss11_kerberos_token_with_message_protection_basic128_service_policy	"Kerberos with Message Protection Using Derived Keys" on page 5-25
Microsoft WCF/.NET 3.5	Policy created with http_spnego_token_service_template	"Kerberos with SPNEGO Negotiation" on page 5-29
Microsoft WCF/.NET 3.5	Policy created with http_spnego_token_service_template	"Kerberos with SPNEGO Negotiation and Credential Delegation" on page 5-30
Oracle Service Bus 10*g*	wss10_username_token_with_message_protection_client_policy wss10_username_token_with_message_protection_service_policy	"Implementing a Username Token with WS-Security 1.0 Message Protection" on page 7-2
Oracle Service Bus 10*g*	oracle/wss10_saml_token_with_message_protection_service_policy oracle/wss10_saml_token_with_message_protection_client_policy	"Implementing a SAML Sender Vouches Token with WS-Security 1.0 Message Protection" on page 7-6
Oracle Service Bus 10*g*	oracle/wss_saml_or_username_token_over_ssl_service_policy	"Implementing a SAML or Username Token Over SSL" on page 7-11
Oracle Service Bus 10*g*	oracle/wss10_x509_token_with_message_protection_service_policy oracle/wss10_x509_token_with_message_protection_client_policy	"Implementing Mutual Authentication with WS-Security 1.0 Message Protection" on page 7-14
Axis 1.4 and WSS4J 1.5.8	oracle/wss10_username_token_with_message_protection_service_policy oracle/wss10_username_token_with_message_protection_client_policy	"Username Token with Message Protection (WS-Security 1.0)" on page 8-3

Table 1–1 (Cont.) Common OWSM Interoperability Scenarios

Security Stack	OWSM Policies	Interoperability Scenario
Axis 1.4 and WSS4J 1.5.8	`oracle/wss10_saml_token_with_message_protection_service_policy` `oracle/wss10_saml_token_with_message_protection_client_policy`	"SAML Token with Message Protection (WS-Security 1.0)" on page 8-6
Axis 1.4 and WSS4J 1.5.8	`oracle/wss_username_token_over_ssl_service_policy` `oracle/wss_username_token_over_ssl_client_policy`	"Username Token Over SSL" on page 8-9
Axis 1.4 and WSS4J 1.5.8	`oracle/wss_saml_token_over_ssl_service_policy` `oracle/wss_saml_token_over_ssl_client_policy`	"SAML Token (Sender Vouches) Over SSL" on page 8-11
GlassFish Enterprise Server	`oracle/wss11_saml_token_with_message_protection_service_policy` `oracle/wss11_saml_token_with_message_protection_client_policy`	"SAML Token (Sender Vouches) with Message Protection (WS-Security 1.1)" on page 9-4

2

Interoperability with OWSM 10g Security Environments

This chapter describes interoperability of Oracle Web Services Manager (OWSM) with OWSM 10*g* security environments.

This chapter includes the following sections:

2.1 Overview of Interoperability with OWSM 10*g* Security Environments

With OWSM 10*g*, you specify *policy steps* at each policy enforcement point. The policy enforcement points in OWSM 10*g* include Gateways and Agents. Each policy step is a fine-grained operational task that addresses a specific security operation, such as authentication and authorization; encryption and decryption; security signature, token, or credential verification; and transformation. Each operational task is performed on either the web service request or response. For more details about the OWSM 10*g* policy steps, see "Oracle Web Services Manager Policy Steps" in *Oracle Web Services Manager Administrator's Guide 10g (10.1.3.4)* at `http://download.oracle.com/docs/cd/E12524_01/web.1013/e12575/policy_steps.htm#BABIAHEG`.

With OWSM 12*c*, you attach *policies* to web service endpoints. Each policy consists of one or more *assertions*, defined at the domain-level, that define the security requirements. A set of predefined policies and assertions are provided out-of-the-box.

Table 2–1 and Table 2–2 summarize the most common OWSM 10*g* interoperability scenarios based on the following security requirements: authentication, message protection, and transport.

For more information about:

- OWSM predefined policies, see "Predefined Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.
- Configuring and attaching OWSM 12*c* policies, see "Securing Web Services" and "Attaching Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.
- OWSM 10*g* policy steps, see "Oracle Web Services Manager Policy Steps" in *Oracle Web Services Manager Administrator's Guide 10g (10.1.3.4)* at `http://download.oracle.com/docs/cd/E12524_01/web.1013/e12575/policy_steps.htm#BABIAHEG`

> **Note:** In the following scenarios, ensure that you are using a keystore with v3 certificates. By default, the JDK 1.5 keytool generates keystores with v1 certificates.
>
> Review "A Note About OWSM 10g Gateways" on page 2-3 and "A Note About Third-party Software" on page 2-3 for important information about your usage of OWSM 10g Gateways and third-party software.

Table 2–1 OWSM 10g Service Policy and OWSM 12c Client Policy Interoperability

Identity Token	WS-Security Version	Message Protection	Transport Security	Service Policy	Client Policy
Anonymous	1.0	Yes	No	Request pipeline: Decrypt and Verify Signature Response pipeline: Sign Message and Encrypt	`oracle/wss10_message_protection_client_policy`
Username	1.0	Yes	No	Request pipeline: ▪ Decrypt and Verify Signature ▪ Extract Credentials (configured as WS-BASIC) ▪ File Authenticate Response pipeline: Sign Message and Encrypt	`oracle/wss10_username_token_with_message_protection_client_policy`
SAML	1.0	Yes	No	Request pipeline: ▪ XML Decrypt ▪ SAML—Verify WSS 1.0 Token Response pipeline: Sign Message and Encrypt	`oracle/wss10_saml_token_with_message_protection_client_policy`
Mutual Authentication	1.0	Yes	No	Request pipeline: Decrypt and Verify Response pipeline: Sign Message and Encrypt	`oracle/wss10_x509_token_with_message_protection_client_policy`
Username over SSL	1.0 and 1.1	No	Yes	Request pipeline: ▪ Extract Credentials ▪ File Authenticate	`wss_username_token_over_ssl_client_policy`
SAML over SSL	1.0 and 1.1	No	Yes	Request pipeline: ▪ Extract Credentials ▪ File Authenticate	`oracle/wss_saml_token_over_ssl_client_policy`

Table 2–2 OWSM 12c Service Policy and OWSM 10g Client Policy Interoperability

Identity Token	WS-Security Version	Message Protection	Transport Security	Service Policy	Client Policy
Anonymous	1.0	Yes	No	`oracle/wss10_message_protection_service_policy`	Request pipeline: Sign Message and Encrypt Response pipeline: Decrypt and Verify Signature
Username	1.0	Yes	No	`oracle/wss10_username_token_with_message_protection_service_policy`	Request pipeline: Sign Message and Encrypt Response pipeline: Decrypt and Verify Signature
SAML	1.0	Yes	No	`oracle/wss10_saml_token_with_message_protection_service_policy`	Request pipeline: ■ Extract Credentials (configured as WS-BASIC ■ SAML—Insert WSS 1.0 Sender-Vouches Token ■ Sign and Encrypt Response pipeline: Decrypt and Verify Signature
Mutual Authentication	1.0	Yes	No	`oracle/wss10_x509_token_with_message_protection_service_policy`	Request pipeline: Sign Message and Encrypt Response pipeline: Decrypt and Verify Signature
Username over SSL	1.0 and 1.1	No	Yes	`wss_username_token_over_ssl_service_policy`	N/A
SAML over SSL	1.0 and 1.1	No	Yes	`oracle/wss_saml_token_over_ssl_service_policy`	Request pipeline: ■ Extract Credentials ■ SAML—Insert WSS 1.0 Sender-Vouches Token

The following sections provide additional interoperability information about using OWSM 10*g* Gateways and third-party software with OWSM 12*c*.

2.2 A Note About OWSM 10*g* Gateways

Oracle Fusion Middleware 12*c* does not include a Gateway component. You can continue to use the OWSM 10g Gateway components with OWSM 10*g* policies in your applications.

2.3 A Note About Third-party Software

OWSM 10*g* supports policy enforcement for third-party application servers, such as IBM WebSphere and Red Hat JBoss. Oracle Fusion Middleware 12*c* only supports Oracle WebLogic Server. You can continue to use the third-party application servers with OWSM 10*g* policies.

2.4 Anonymous Authentication with Message Protection (WS-Security 1.0)

This section describes how to implement anonymous authentication with message protection that conforms to the WS-Security 1.0 standard, in the following interoperability scenarios:

- "Configuring an OWSM 12c Web Service and an OWSM 10g Client" on page 2-4
- "Configuring an OWSM 10g Web Service and an OWSM 12c Client" on page 2-5

2.4.1 Configuring an OWSM 12*c* Web Service and an OWSM 10*g* Client

The following instructions tell how to configure an OWSM 12*c* web service and an OWSM 10*g* client to implement anonymous authentication with message protection that conforms to the WS-Security 1.0 standard:

To configure an OWSM 12c web service:

1. Clone the following policy: `oracle/wss10_message_protection_service_policy`

   For more information, see "Cloning a Web Service Policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*

2. Edit the policy settings, as follows:

   a. Disable the Include Timestamp configuration setting.

   b. Leave the default configuration set for all other configuration settings.

3. Attach the policy to a web service.

   For more information, see

   "Attaching Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*

To configure OWSM 10g client:

1. Register the web service (above) with the OWSM 10*g* Gateway.

   For more information, see"Registering Web Services to an OWSM Gateway" in the *OWSM Administrator's Guide 10g* at:
   `http://download.oracle.com/docs/cd/E12524_01/web.1013/e12575/gateways.htm`

2. Attach the following policy step to the request pipeline: Sign Message and Encrypt Gateway.

3. Configure the Sign Message and Encrypt policy step in the request pipeline, as follows:

   a. Set Encryption Algorithm to AES-128.

   b. Set Key Transport Algorithm to RSA-OAEP-MGF1P.

   c. Configure the keystore properties for message signing and encryption. The configuration should be in accordance with the keystore used on the server side

4. Attach the following policy step to the response pipeline: Decrypt and Verify Signature.

5. Configure the Decrypt and Verify Signature policy step in the response pipeline, by configuring the keystore properties for decryption and signature verification.

The configuration should be in accordance with the keystore used on the server side.

6. Navigate to the OWSM Test page and enter the virtualized URL of the web service.
7. Invoke the web service.

2.4.2 Configuring an OWSM 10*g* Web Service and an OWSM 12*c* Client

The following instructions tell how to configure an OWSM 10*g* web service and an OWSM 12*c* client to implement anonymous authentication with message protection that conforms to the WS-Security 1.0 standard:

To configure the OWSM 10g web service:

1. Register the web service with the OWSM 10*g* Gateway.

   For more information, see "Registering Web Services to an OWSM Gateway" in the *OWSM Administrator's Guide 10g* at: `http://download.oracle.com/docs/cd/E12524_01/web.1013/e12575/gateways.htm`

2. Attach the following policy step in the request pipeline: Decrypt and Verify Signature.
3. Configure the Decrypt and Verify Signature policy step in the request pipeline, as follows. Configure the keystore properties for decryption and signature verification. The configuration should be in accordance with the keystore used on the server side.
4. Attach the following policy step in the response pipeline: Sign Message and Encrypt.
5. Configure the Sign Message and Encrypt policy response pipeline as follows:
   a. Set Encryption Algorithm to AES-128.
   b. Set Key Transport Algorithm to RSA-OAEP-MGF1P.
   c. Configure the keystore properties for message signing and encryption. The configuration should be in accordance with the keystore used on the server side.

To configure the OWSM 12c client:

1. Create a client proxy using the virtualized URL of the web service registered on the OWSM Gateway.
2. Clone the following policy: `oracle/wss10_message_protection_client_policy`.
3. Edit the policy settings, as follows:
   a. Disable the Include Timestamp configuration setting.
   b. Leave the default configuration set for all other configuration settings.
4. Attach the policy to the web service client.

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

5. Configure the policy.

   For more information, see "oracle/wss10_message_protection_client_policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*

6. Invoke the web service.

2.5 Username Token with Message Protection (WS-Security 1.0)

This section tells how to implement username token with message protection that conforms to the WS-Security 1.0 standard:

- "Configuring an OWSM 12c Web Service and an OWSM 10g Client" on page 2-6
- "Configuring an OWSM 10g Web Service and an OWSM 12c Client" on page 2-7

2.5.1 Configuring an OWSM 12*c* Web Service and an OWSM 10*g* Client

The following instructions tell how to configure an OWSM 12*c* web service and an OWSM 10*g* client to implement username token with message protection that conforms to the WS-Security 1.0 standard:

To configure the OWSM 12c Web Service:

1. Clone the following policy: `oracle/wss10_username_token_with_message_protection_service_policy`
2. Edit the policy settings, as follows:
   a. Disable the Include Timestamp configuration setting.
   b. Leave the default configuration set for all other configuration settings.

      For more information, see "Cloning a Web Service Policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*
3. Attach the policy to a web service.

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

To configure the OWSM 10g client:

1. Register the web service (above) with the OWSM 10*g* Gateway.

   Fore more information, see "Registering Web Services to an OWSM Gateway" in the *OWSM Administrator's Guide 10g* at: `http://download.oracle.com/docs/cd/E12524_01/web.1013/e12575/gateways.htm`
2. Attach the following policy step to the request pipeline: Sign Message and Encrypt
3. Configure the Sign Message and Encrypt policy step in the request pipeline, as follows:
   a. Set Encryption Algorithm to AES-128.
   b. Set Key Transport Algorithm to RSA-OAEP-MGF1P.
   c. Set Encrypted Content to ENVELOPE.
   d. Set Signed Content to ENVELOPE.
   e. Configure the keystore properties for message signing and encryption. The configuration should be in accordance with the keystore used on the server side.
4. Attach the following policy step to the response pipeline: Decrypt and Verify Signature.

5. Configure the Decrypt and Verify Signature policy step in the response pipeline, as follows:

   a. Configure the keystore properties for decryption and signature verification. The configuration should be in accordance with the keystore used on the server side.

6. Navigate to the OWSM Test page and enter the virtualized URL of the web service.

7. Select the **Include Header** checkbox against WS-Security and provide valid credentials.

8. Invoke the web service.

2.5.2 Configuring an OWSM 10*g* Web Service and an OWSM 12*c* Client

The following instructions tell how to configure an OWSM 10*g* web service and an OWSM 12*c* client to implement username token with message protection that conforms to the WS-Security 1.0 standard:

To configure the OWSM 10g web service:

1. Register the web service with the OWSM 10*g* Gateway.

   For more information, see "Registering Web Services to an OWSM Gateway" in the *OWSM Administrator's Guide 10g* at:
   `http://download.oracle.com/docs/cd/E12524_01/web.1013/e12575/gateways.htm`

2. Attach the following policy steps in the request pipeline:

   a. Decrypt and Verify Signature

   b. Extract Credentials (configured as WS-BASIC)

   c. File Authenticate

   Note: You can substitute File Authenticate with LDAP Authenticate, Oracle Access Manager Authenticate, Active Directory Authenticate, or SiteMinder Authenticate.

3. Configure the Decrypt and Verify Signature policy step in the request pipeline, as follows:

   a. Configure the keystore properties for extracting credentials. The configuration should be in accordance with the keystore used on the server side.

4. Configure the Extract Credentials policy step in the request pipeline, as follows:

   a. Set the Credentials location to WS-BASIC.

5. Configure the File Authenticate policy step in the request pipeline to use valid credentials.

6. Attach the following policy step in the response pipeline: Sign Message and Encrypt.

   a. Set Encryption Algorithm to AES-128.

   b. Set Key Transport Algorithm to RSA-OAEP-MGF1P.

c. Configure the keystore properties for message signing and encryption. The configuration should be in accordance with the keystore used on the server side.

To Configure the OWSM 12c Client:

1. Create a client proxy using the virtualized URL of the web service registered on the OWSM Gateway.
2. Clone the following policy: `oracle/wss10_username_token_with_message_protection_client_policy`

   For more information, see "Cloning a Web Service Policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.
3. Edit the policy settings, as follows:
   a. Disable the Include Timestamp configuration setting.
   b. Leave the default configuration set for all other configuration settings.

   For more information, see "Cloning a Web Service Policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.
4. Attach the policy to the web service client.

   For more information, see "Attaching Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*
5. Configure the policy.

   For more information, see "oracle/wss10_username_token_with_message_protection_client_policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*
6. Invoke the web service.

2.6 SAML Token (Sender Vouches) with Message Protection (WS-Security 1.0)

This section tells how to implement SAML token (sender vouches) with message protection that conforms to the WS-Security 1.0 standard, in the following interoperability scenarios:

- "Configuring an OWSM 12c Web Service and an OWSM 10g Client" on page 2-8
- "Configuring an OWSM 10g Web Service and an OWSM 12c Client" on page 2-10

2.6.1 Configuring an OWSM 12*c* Web Service and an OWSM 10*g* Client

The following instructions tell how to configure an OWSM 12*c* web service and an OWSM 10*g* client to implement SAML token (sender vouches) with message protection that conforms to the WS-Security 1.0 standard:

To Configure the OWSM 12c Web Service:

1. Clone the following policy: oracle/wss10_saml_token_with_message_protection_service_policy.

> **Note:** Oracle recommends that you do not change the predefined policies so that you will always have a known set of valid policies to work with.

2. Edit the policy settings, as follows:
   a. Disable the Include Timestamp configuration setting.
   b. Leave the default configuration set for all other configuration settings.
3. Attach the policy to the web service.

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

To Configure the OWSM 10g Client:

1. Register the web service with the OWSM 10*g* Gateway.

   For more information, see "Registering Web Services to an OWSM Gateway" in the OWSM Administrator's Guide 10g at: http://download.oracle.com/docs/cd/E12524_01/web.1013/e12575/gateways.htm
2. Attach the following policy steps in the request pipeline:
   a. Extract Credentials (configured as WS-BASIC)
   b. SAML—Insert WSS 1.0 Sender-Vouches Token
   c. Sign Message and Encrypt
3. Configure the Extract Credentials policy step in the request pipeline, as follows:
   a. Set the Credentials location to WS-BASIC.
4. Configure the SAML—Insert WSS 1.0 Sender-Vouches Token policy step in the request pipeline, as follows:
   a. Set Subject Name Qualifier to www.oracle.com
   b. Set Assertion Issuer as www.oracle.com
   c. Set Subject Format as UNSPECIFIED.
   d. Set other signing properties, as required.
5. Configure the Sign Message and Encrypt policy step in the request pipeline, as follows:
   a. Set the Encryption Algorithm to AES-128.
   b. Set Key Transport Algorithm to RSA-OAEP-MGF1P.
   c. Configure the keystore properties for decryption and signature verification. The configuration should be in accordance with the keystore used on the server side.
6. Attach the following policy step in the response pipeline: Decrypt and Verify Signature.
7. Configure the Decrypt and Verify Signature policy step in the response pipeline, as follows:
   a. Configure the keystore properties for decryption and signature verification. The configuration should be in accordance with the keystore used on the server side.
8. Navigate to the OWSM Test page and enter the virtualized URL of the web service.
9. Select Include Header checkbox against WS-Security and provide valid credentials.

10. Invoke the web service.

2.6.2 Configuring an OWSM 10*g* Web Service and an OWSM 12*c* Client

The following instructions tell how to configure an OWSM 10*g* web service and an OWSM 12*c* client to implement SAML token (sender vouches) with message protection that conforms to the WS-Security 1.0 standard:

To Configure the OWSM 10g Web Service:

1. Register the web service with the OWSM 10*g* Gateway.

   For more information, see "Registering Web Services to an OWSM Gateway" in the *OWSM Administrator's Guide 10g* at: `http://download.oracle.com/docs/cd/E12524_01/web.1013/e12575/gateways.htm`

2. Attach the following policy steps in the request pipeline:

   a. XML Decrypt

   b. SAML—Verify WSS 1.0 Token

3. Configure the XML Decrypt policy step in the request pipeline, as follows:

   a. Configure the keystore properties for XML decryption. The configuration should be in accordance with the keystore used on the server side.

4. Configure the SAML—Verify WSS 1.0 Token policy step in the request pipeline, as follows:

   a. Set the Trusted Issuer Name as www.oracle.com

5. Attach the following policy step in the response pipeline: Sign Message and Encrypt.

6. Configure the Sign Message and Encrypt policy step in the response pipeline, follows:

   a. Set Encryption Algorithm to AES-128.

   b. Set Key Transport Algorithm to RSA-OAEP-MGF1P.

   c. Configure the keystore properties for message signing and encryption. The configuration should be in accordance with the keystore used on the server side.

To Configure the OWSM 12c Client:

1. Create a client proxy using the virtualized URL of the web service registered on the OWSM Gateway.

2. Clone the following policy: `oracle/wss10_saml_token_with_message_protection_client_policy`.

   For more information, see "Cloning a Web Service Policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*

3. Edit the policy settings, as follows:

   1. Disable the Include Timestamp configuration setting.

   2. Leave the default configuration set for all other configuration settings.

4. Attach the policy to the web service client.

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

5. Configure the policy.

   For more information, see "oracle/wss10_saml_token_with_message_protection_client_policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

6. Invoke the web service.

2.7 Mutual Authentication with Message Protection (WS-Security 1.0)

This section tells how to implement mutual authentication with message protection that conform to the WS-Security 1.0 standard, in the following interoperability scenarios:

- "Configuring an OWSM 12c Web Service and an OWSM 10g Client" on page 2-11
- "Configuring an OWSM 10g Web Service and an OWSM 12c Client" on page 2-12

2.7.1 Configuring an OWSM 12*c* Web Service and an OWSM 10*g* Client

The following instructions tell how to configure an OWSM 12*c* web service and an OWSM 10*g* client to implement mutual authentication with message protection that conform to the WS-Security 1.0 standard:

To Configure the OWSM 12c Web Service:

1. Clone the following policy: `oracle/wss10_x509_token_with_message_protection_service_policy`.

   For more information, see "Cloning a Web Service Policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*

2. Edit the policy settings, as follows:

   a. Disable the Include Timestamp configuration setting.

   b. Leave the default configuration set for all other configuration settings.

3. Attach the policy to the web service.

To Configure the OWSM 10g Client:

1. Register the web service (above) with the OWSM 10*g* Gateway.

   For more information, see "Registering Web Services to an OWSM Gateway" in the *OWSM Administrator's Guide 10g* at: `http://download.oracle.com/docs/cd/E12524_01/web.1013/e12575/gateways.htm`

2. Attach the following policy step in the request pipeline: Sign Message and Encrypt.

3. Configure the Sign Message and Encrypt policy step in the request pipeline, as follows:

   a. Set Encryption Algorithm to AES-128.

   b. Set Key Transport Algorithm to RSA-OAEP-MGF1P.

   c. Configure the keystore properties for message signing and encryption. The configuration should be in accordance with the keystore used on the server side.

4. Attach the following policy step in the response pipeline: Decrypt and Verify Signature.
5. Configure the Decrypt and Verify Signature policy step in the response pipeline, as follows:
   a. Configure the keystore properties for decryption and signature verification. The configuration should be in accordance with the keystore used on the server side.
6. Update the following property in the `gateway-config-installer.properties` file located at *ORACLE_HOME*`/j2ee/`*oc4j_instance*`/applications/gateway/gateway/WEB-INF`:

   ```
   pep.securitysteps.signBinarySecurityToken=true
   ```
7. Restart OWSM 10*g* Gateway.
8. Navigate to the OWSM Test page and enter the virtualized URL of the web service.
9. Invoke the web service.

2.7.2 Configuring an OWSM 10*g* Web Service and an OWSM 12*c* Client

The following instructions tell how to configure an OWSM 10*g* web service and an OWSM 12*c* client to implement mutual authentication with message protection that conform to the WS-Security 1.0 standard:

To Configure the OWSM 10g Web Service:

1. Register the web service (above) with the OWSM 10*g* Gateway.

   "Registering Web Services to an OWSM Gateway" in the *OWSM Administrator's Guide 10g* at:
   `http://download.oracle.com/docs/cd/E12524_01/web.1013/e12575/gateways.htm`
2. Attach the following policy steps in the request pipeline: Decrypt and Verify.
3. Configure the Decrypt and Verify Signature policy step in the request pipeline, as follows:
   a. Configure the keystore properties for decryption and signature verification. The configuration should be in accordance with the keystore used on the server side.
4. Attach the following policy steps in the response pipeline: Sign Message and Encrypt.
5. Configure the Sign Message and Encrypt policy step in the response pipeline, as follows:
   a. Set Encryption Algorithm to AES-128.
   b. Set Key Transport Algorithm to RSA-OAEP-MGF1P.
   c. Configure the keystore properties for message signing and encryption. The configuration should be in accordance with the keystore used on the server side.

To Configure the OWSM 12c Client:

1. Create a client proxy using the virtualized URL of the web service registered on the OWSM Gateway.

2. Clone the following policy: `oracle/wss10_x509_token_with_message_protection_client_policy`.

   For more information, see "Cloning a Web Service Policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*

3. Edit the policy settings, as follows:

   a. Disable the Include Timestamp configuration setting.

   b. Leave the default configuration set for all other configuration settings.

4. Attach the policy to the web service client.

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

5. Configure the policy.

   "oracle/wss10_x509_token_with_message_protection_client_policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*

6. Invoke the web service.

2.8 Username Token Over SSL

This section tells how to implement username token over SSL, in the following interoperability scenarios:

- "Configuring an OWSM 12c Web Service and an OWSM 10g Client" on page 2-13
- "Configuring an OWSM 10g Web Service and an OWSM 12c Client" on page 2-14

For more information about:

- Configuring SSL on WebLogic Server, see "Configuring Transport-Level Security (SSL)" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.
- Configuring SSL on OC4J, see `http://download.oracle.com/docs/cd/B14099_19/web.1012/b14013/configssl.htm`.

2.8.1 Configuring an OWSM 12*c* Web Service and an OWSM 10*g* Client

The following instructions tell how to configure an OWSM 12*c* web service and an OWSM 10*g* client to implement username token over SSL:

To Configure the OWSM 12c Web Service:

1. Configure the server for SSL.

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

2. Attach the following policy: `wss_username_token_over_ssl_service_policy`.

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

To Configure the OWSM 10g Client:

1. Configure the server for SSL

   For more information, see " Configuring OC4J and SSL" in *Oracle Application Server Containers for J2EE Security Guide* at http://download.oracle.com/docs/cd/B14099_19/web.1012/b14013/configssl.htm

2. Register the web service (above) with the OWSM 10*g* Gateway.

   "Registering Web Services to an OWSM Gateway" in the *OWSM Administrator's Guide 10g* at: http://download.oracle.com/docs/cd/E12524_01/web.1013/e12575/gateways.htm

3. Navigate to the OWSM Test page and enter the virtualized URL of the web service.

4. Select the **Include Header** checkbox against WS-Security and provide valid credentials.

5. Invoke the web service.

2.8.2 Configuring an OWSM 10*g* Web Service and an OWSM 12*c* Client

The following instructions tell how to configure an OWSM 10*g* web service and an OWSM 12*c* client to implement username token over SSL:

To Configure the OWSM 10g Web Service:

1. Configure the server for SSL

   For more information, see " Configuring OC4J and SSL" in *Oracle Application Server Containers for J2EE Security Guide* at http://download.oracle.com/docs/cd/B14099_19/web.1012/b14013/configssl.htm

2. Register the web service (above) with the OWSM 10*g* Gateway.

   "Registering Web Services to an OWSM Gateway" in the *OWSM Administrator's Guide 10g* at: http://download.oracle.com/docs/cd/E12524_01/web.1013/e12575/gateways.htm

3. Attach the following policy steps to the request pipeline:

   a. Extract Credentials

   b. File Authenticate

   > **Note:** You can substitute File Authenticate with LDAP Authenticate, Oracle Access Manager Authenticate, Active Directory Authenticate, or SiteMinder Authenticate.

4. Configure the Extract Credentials policy step in the request pipeline, as follows:

   a. Configure the Credentials Location as WS-BASIC.

5. Configure the File Authentication policy step in the request pipeline with the appropriate credentials.

To Configure the OWSM 12c Client:

1. Create a client proxy using the virtualized URL of the web service registered on the OWSM Gateway. Ensure that when generating the client, HTTP is specified in the URL along with the HTTP port number.

2. Clone the following policy: `oracle/wss_username_token_over_ssl_client_policy`.

   For more information, see "Cloning a Web Service Policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*

3. Edit the policy settings, as follows:

   a. Disable the Include Timestamp configuration setting.

   b. Leave the default configuration set for all other configuration settings.

4. Attach the policy to the web service client.

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

5. Configure the policy.

   For more information, see "oracle/wss_username_token_over_ssl_client_policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*

6. Invoke the web service.

2.9 SAML Token (Sender Vouches) Over SSL (WS-Security 1.0)

This section tells how to implement SAML token (sender vouches) over SSL that conforms to the WS-Security 1.0 standard, in the following interoperability scenarios:

- "Configuring an OWSM 12c Web Service and an OWSM 10g Client" on page 2-15
- "Configuring an OWSM 10g Web Service and OWSM 12c Client" on page 2-16

For more information about:

- Configuring SSL on WebLogic Server, see "Configuring Transport-Level Security (SSL)" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.
- Configuring SSL on OC4J, see `http://download.oracle.com/docs/cd/B14099_19/web.1012/b14013/configssl.htm`.

2.9.1 Configuring an OWSM 12*c* Web Service and an OWSM 10*g* Client

The following instructions tell how to configure an OWSM 12*c* web service and an OWSM 10*g* client to implement SAML token (sender vouches) over SSL that conforms to the WS-Security 1.0 standard:

To Configure the OWSM 12c Web Service:

1. Configure the server for two-way SSL.

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

2. Clone the following policy: `oracle/wss_saml_token_over_ssl_service_policy`.

   For more information, see "Cloning a Web Service Policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*

3. Edit the policy settings, as follows:

   a. Disable the Include Timestamp configuration setting.

For more information, see "Cloning a Web Service Policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*

4. Attach the policy.

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

To Configure the OWSM 10g Client:

1. Configure the server for two-way SSL.

   For more information, see " Configuring OC4J and SSL" in *Oracle Application Server Containers for J2EE Security Guide* at `http://download.oracle.com/docs/cd/B14099_19/web.1012/b14013/configssl.htm`

2. Register the web service (above) with the OWSM 10*g* Gateway.

   For more information, see "Registering Web Services to an OWSM Gateway" in the *OWSM Administrator's Guide 10g* at: `http://download.oracle.com/docs/cd/E12524_01/web.1013/e12575/gateways.htm`

3. Attach the following policy steps to the request pipeline:

   a. Extract Credentials

   b. SAML—Insert WSS 1.0 Sender-Vouches Token

4. Configure the Extra Credentials policy step in the request pipeline, as follows:

   a. Configure the Credentials Location as WS-BASIC.

5. Configure the SAML—Insert WSS 1.0 Sender-Vouches Token policy step in the request pipeline, as follows:

   a. Configure the Subject Name Qualifier as www.oracle.com

   b. Configure the Assertion Issuer as www.oracle.com

   c. Configure the Subject Format as UNSPECIFIED.

   d. Configure the Sign the assertion as false.

6. Navigate to the OWSM Test page and enter the virtualized URL of the web service.

7. Select **Include Header** checkbox against WS-Security and provide valid credentials.

8. Invoke the web service.

2.9.2 Configuring an OWSM 10*g* Web Service and OWSM 12*c* Client

The following instructions tell how to configure an OWSM 10*g* web service and an OWSM 12*c* client to implement SAML token (sender vouches) over SSL that conforms to the WS-Security 1.0 standard:

To Configure the OWSM 10g Web Service:

1. Configure the server for two-way SSL.

   For more information, see " Configuring OC4J and SSL" in *Oracle Application Server Containers for J2EE Security Guide* at `http://download.oracle.com/docs/cd/B14099_19/web.1012/b14013/configssl.htm`

2. Register the web service (above) with the OWSM 10*g* Gateway.

   For more information, see "Registering Web Services to an OWSM Gateway" in the *OWSM Administrator's Guide 10g* at:
   `http://download.oracle.com/docs/cd/E12524_01/web.1013/e12575/gateways.htm`

3. Attach the policy step: SAML—Verify WSS 1.0 Token

4. Configure the SAML—Verify WSS 1.0 Token policy step in the request pipeline, as follows:

   a. Under Signature Verification Properties, set Allow signed assertions only to false.

   b. Set the Trusted Issuer Name to `www.oracle.com`.

To Configure the OWSM 12c Client:

1. Configure the server for two-way SSL.

   For more information, see "Configuring SSL on WebLogic Server (Two-Way)" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*

2. Create a client proxy using the virtualized URL of the web service registered on the OWSM gateway.

3. s: `oracle/wss_saml_token_over_ssl_client_policy`.

   For more information, see "Cloning a Web Service Policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*

4. Edit the policy settings, as follows:

   a. Disable the Include Timestamp configuration setting.

   b. Leave the default configuration set for all other configuration settings.

5. Attach the policy to the web service client.

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

6. Configure the policy.

   For more information, see "oracle/wss_username_token_over_ssl_client_policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*

7. Invoke the web service.

3

Interoperability with Oracle Containers for Java EE (OC4J) 10*g* Security Environments

This chapter describes the most common Oracle Containers for Java EE (OC4J) 10*g* interoperability scenarios based on the following security requirements: authentication, message protection, and transport.

This chapter includes the following sections:

3.1 Overview of Interoperability with OC4J 10*g* Security Environments

In OC4J 10*g*, you configure your security environment, as described in the following documents.

- For information about using Application Server Control to configure the web service, see *Oracle Application Server Advanced Web Services Developer's Guide* at `http://download.oracle.com/docs/cd/B31017_01/web.1013/b28975/toc.htm`.
- For information about using JDeveloper to develop and configure your client-side application, see *Developing Applications with Oracle JDeveloper*.
- For information about how to modify the XML-based deployment descriptor files, see *Oracle Application Server Web Services Security Guide 10g (10.1.3.1.0)* at: `http://download.oracle.com/docs/cd/B31017_01/web.1013/b28976/toc.htm`

With OWSM 12*c*, you attach *policies* to web service endpoints. Each policy consists of one or more *assertions*, defined at the domain-level, that define the security requirements. A set of predefined policies and assertions are provided out-of-the-box.

For more information about:

- OWSM predefined policies, see "Predefined Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

- Configuring and attaching OWSM 12*c* policies, see "Securing Web Services" and "Attaching Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

Table 3–2 and Table 3–2 summarize the most common OC4J 10*g* interoperability scenarios based on the following security requirements: authentication, message protection, and transport.

> **Note:** In the following scenarios, ensure that you are using a keystore with v3 certificates. By default, the JDK 1.5 keytool generates keystores with v1 certificates.

Table 3–1 OWSM 12c Service Policy and Oracle OC4J 10g Client Policy Interoperability

Identity Token	WS-Security Version	Message Protection	Transport Security	Service Policy	Client Policy
Anonymous	1.0	Yes	No	`oracle/wss10_message_protection_service_policy`	See Table 3–4 , " Configuring the OC4J 10g Client" on page 3-4
Username	1.0	Yes	No	`oracle/wss10_username_token_with_message_protection_service_policy`	See Table 3–10, " Configuring the OC4J 10g Client" on page 3-8
SAML	1.0	Yes	No	`oracle/wss10_saml_token_with_message_protection_service_policy`	See Table 3–4, " Configuring the OC4J 10g Client" on page 3-11
Mutual Authentication	1.0	Yes	No	`oracle/wss10_x509_token_with_message_protection_service_policy`	See Table 3–10, " Configuring the OC4J 10g Client" on page 3-15

Username over SSL	1.0 and 1.1	No	Yes	`oracle/wss_username_token_over_ssl_service_policy`
				OR
				`oracle/wss_saml_or_username_token_over_ssl_service_policy`
					See Table 3–16, " Configuring the OC4J 10g Client" on page 3-19

SAML over SSL	1.0 and 1.1	No	Yes	`oracle/wss_saml_token_over_ssl_service_policy`
				OR
				`oracle/wss_saml_or_username_token_over_ssl_service_policy`
					See Table 3–22, " Configuring the OC4J 10g Client" on page 3-25

Table 3–2 Oracle OC4J 10g Service Policy and OWSM 12c Client Policy Interoperability

Identity Token	WS-Security Version	Message Protection	Transport Security	Service Policy	Client Policy
Anonymous	1.0	Yes	No	See Table 3–6, " Configuring the OC4J 10g Web Service" on page 3-5	`oracle/wss10_message_protection_client_policy`
Username	1.0	Yes	No	See Table 3–12, " Configuring the OC4J 10g Web Service" on page 3-10	`oracle/wss10_username_token_with_message_protection_client_policy`
SAML	1.0	Yes	No	See Table 3–6, " Configuring the OC4J 10g Web Service" on page 3-12	`oracle/wss10_saml_token_with_message_protection_client_policy`
Mutual Authentication	1.0	Yes	No	See Table 3–12, " Configuring the OC4J 10g Web Service" on page 3-16	`oracle/wss10_x509_token_with_message_protection_client_policy`
Username over SSL	1.0 and 1.1	No	Yes	See Table 3–18, " Configuring the OC4J 10g Web Service" on page 3-20	`oracle/wss_username_token_over_ssl_client_policy`
SAML over SSL	1.0 and 1.1	No	Yes	See Table 3–24, " Configuring the OC4J 10g Web Service" on page 3-24	`oracle/wss_saml_token_over_ssl_client_policy`

3.2 Anonymous Authentication with Message Protection (WS-Security 1.0)

This section tells how to implement anonymous authentication with message protection that conforms to the WS-Security 1.0 standard, in the following interoperability scenarios:

- "Configuring an OWSM 12c Web Service and an OC4J 10g Client"
- "Configuring an OC4J 10g Web Service and an OWSM 12c Client"

3.2.1 Configuring an OWSM 12*c* Web Service and an OC4J 10*g* Client

The following instructions tell how to configure an OWSM 12*c* web service and an OWSM 10*g* client to implement anonymous authentication with message protection that conforms to the WS-Security 1.0 standard:

To Configure the OWSM 12c Web Service:

1. Create a web service application.
2. Attach the following policy to the entry point of the web service: `oracle/wss10_message_protection_service_policy`.

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

To Configure the OC4J 10g Client:

1. Create a client proxy for the web service using Oracle JDeveloper.

   For more information, see "Developing and Securing Web Services" in *Developing Applications with Oracle JDeveloper*.

2. Use the Oracle JDeveloper wizard to secure the proxy by right-clicking on the proxy project and selecting **Secure Proxy**.
3. Click **Authentication** in the Proxy Editor navigation bar and set the following options:
   a. Select No Authentication.
4. Click **Inbound Integrity** in the Proxy Editor navigation bar and set the following options:
   a. Select **Verify Inbound Signed Request Body**.
   b. Select **Verify Timestamp** and **Creation Time Required in Timestamp**.
   c. Enter the **Expiration Time** (in seconds).
   d. Select all options under **Acceptable Signature Algorithms**.
5. Click **Outbound Integrity** in the Proxy Editor navigation bar and set the following options:
   a. Select **Sign Outbound Messages**.
   b. Select **Add Timestamp to Outbound Messages** and **Creation Time Required in Timestamp**.
   c. Enter the **Expiration Time** (in seconds).
6. Click **Inbound Confidentiality** in the Proxy Editor navigation bar and set the following options:
   a. Select **Decrypt Inbound Message Content**.
   b. Select all options under **Acceptable Signature Algorithms**.
7. Click **Outbound Confidentiality** in the Proxy Editor navigation bar and set the following options:
   a. Select **Encrypt Outbound Messages**.
   b. Set the Algorithm to **AES-128**.
8. Click **Keystore Options** in the Proxy Editor navigation bar and configure the keystore properties, as required.

> **Note:** Ensure that you are using keystore with v3 certificates. By default, the JDK 1.5 keytool generates keystores with v1 certificates.

9. Click **OK** to close the wizard.
10. In the Structure pane, click **<appname>Binding_Stub.xml** and edit the file as described in next section.
11. Invoke the web service method from the client.

To edit the <appname>Binding_Stub.xml File:

1. Provide the keystore password and sign and encryption key passwords.
2. In the inbound signature, specify the following:

```
<inbound><verify-signature><tbs-elements>
<tbs-element
name-space="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd" local-part="Timestamp"/>
```

```
...
```

3. In the outbound signature, specify that the timestamp should be signed, as follows:

```
<outbound>/<signature>/<tbs-elements>
<tbs-element
name-space="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity
-utility-1.0.xsd" local-part="Timestamp"/>
...
```

4. In the outbound encryption, specify the key transport algorithm, as follows:

```
<outbound><encrypt>
<keytransport-method>RSA-OAEP-MGF1P</keytransport-method>
...
```

3.2.2 Configuring an OC4J 10*g* Web Service and an OWSM 12*c* Client

The following instructions tell how to configure an OC4J 10*g* web service and an OWSM 12*c* client to implement anonymous authentication with message protection that conforms to the WS-Security 1.0 standard:

To configure the OC4J 10g Web Service:

1. Create and deploy a web service application.
2. Use Application Server Control to secure the deployed web service.
3. Click **Authentication** tab and ensure that no options are selected.
4. Click **Integrity** tab of the Inbound Policies page and set the following options:
   a. Select **Require Message Body to Be Signed**.
   b. Select **Verify Timestamp** and **Creation Time Required in Timestamp**.
   c. Enter the **Expiration Time** (in seconds).
5. Click **Integrity** tab of the Outbound Policies page and set the following options:
   a. Select **Sign Body Element of Message**.
   b. Set the **Signature Method** to **RSA-SHA1**.
   c. Select **Add Timestamp** and **Creation Time Required in Timestamp**.
   d. Enter the **Expiration Time** (in seconds).
6. Click **Confidentiality** tab of the Inbound Policies page and set the following options:
   a. Select **Require Encryption of Message Body**.
7. Click **Confidentiality** tab of the Outbound Policies page and set the following options:
   a. Select **Encrypt Body Element of Message**.
   b. Set the **Encryption Method** to **AES-128**.
   c. Set the public key to encrypt.
8. Configure the keystore properties and identity certificates.

Note: Ensure that you are using keystore with v3 certificates. By default, the JDK 1.5 keytool generates keystores with v1 certificates.

9. Edit the wsmgmt.xml deployment descriptor file, as described in Table 3–8, " Editing the wsmgmt.xml File".

To configure the OWSM 12c Client:

1. Create a client proxy for the OC4J 10*g* web service.
2. Attach the following policy: `oracle/wss10_message_protection_client_policy`.

   For more information, see "Attaching Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*
3. Configure the policy.

   For more information, see "oracle/wss10_username_token_with_message_protection_client_policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*
4. Invoke the web service method from the client.

To edit the wsmgmt.xml File:

1. Locate the `wsmgmt.xml` File under *`ORACLE_HOME`*`/j2ee/`*`oc4j_instance`*`/config`.

   > **Tip:** The `wsmgmt.xml` file is an instance-level configuration file, which holds the entire security configuration for the web services deployed in an OC4J instance.

   For more information, see "Understanding the Web Services Management Schema" in *Oracle® Application Server Advanced Web Services Developer's Guide*
2. In the inbound signature, specify the following:

```
<inbound><verify-signature><tbs-elements>
<tbs-element
name-space="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity
-utility-1.0.xsd" local-part="Timestamp"/>
...
```

3. In the outbound signature, specify that the timestamp should be signed, as follows:

```
<outbound>/<signature>/<tbs-elements>
<tbs-element
name-space="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity
-utility-1.0.xsd" local-part="Timestamp"/>
...
```

4. In the outbound encryption, specify the key transport algorithm, as follows:

```
<outbound><encrypt>
<keytransport-method>RSA-OAEP-MGF1P</keytransport-method>
...
```

3.3 Username Token with Message Protection (WS-Security 1.0)

This section tells how to implement username token with message protection that conforms to the WS-Security 1.0 standard:

- "Configuring an OWSM 12c Web Service and an OC4J 10g Client" on page 3-6
- "Configuring an OC4J 10g Web Service and an OWSM 12c Client" on page 3-8

3.3.1 Configuring an OWSM 12*c* Web Service and an OC4J 10*g* Client

The following instructions tell how to configure an OWSM 12*c* web service and an OC4J 10*g* client to implement username token with message protection that conforms to the WS-Security 1.0 standard:

To Configure the OWSM 12c Web Service:

1. Create an OWSM 12*c* web service.
2. Attach the following policy to the web service: `oracle/wss10_username_token_with_message_protection_service_policy`.

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

To configure the OC4J 10g Client:

1. Create a client proxy for the web service (above) using Oracle JDeveloper.

   For more information, see "Developing and Securing Web Services" in *Developing Applications with Oracle JDeveloper*.
2. Specify the username and password in the client proxy, as follows:

   ```
   port.setUsername(<username>)
   port.setPassword(<password>)
   ```
3. Use the Oracle JDeveloper wizard to secure the proxy by right-clicking on the proxy project and selecting **Secure Proxy**.
4. Click **Authentication** in the Proxy Editor navigation bar and set the following options:
   a. Select **Use Username to Authenticate**.
   b. Deselect **Add Nonce** and **Add Creation Time**.
5. Click **Inbound Integrity** in the Proxy Editor navigation bar and set the following options:
   a. Select **Verify Inbound Signed Request Body**.
   b. Select **Verify Timestamp** and **Creation Time Required in Timestamp**.
   c. Enter the **Expiration Time** (in seconds).
   d. Select all options under **Acceptable Signature Algorithms**.
6. Click **Outbound Integrity** in the Proxy Editor navigation bar and set the following options:
   a. Select **Sign Outbound Messages**.
   b. Select **Add Timestamp to Outbound Messages** and **Creation Time Required in Timestamp**.
   c. Enter the **Expiration Time** (in seconds).
7. Click **Inbound Confidentiality** in the Proxy Editor navigation bar and set the following options:
   a. Select **Decrypt Inbound Message Content**.
   b. Select all options under **Acceptable Signature Algorithms**.
8. Click **Outbound Confidentiality** in the Proxy Editor navigation bar and set the following options:
   a. Select **Encrypt Outbound Messages**.

   b. Set the Algorithm to **AES-128**.

9. Click **Keystore Options** in the Proxy Editor navigation bar and configure the keystore properties, as required.

   > **Tip:** Ensure that you are using keystore with v3 certificates. By default, the JDK 1.5 keytool generates keystores with v1 certificates.

10. Click **OK** to close the wizard.
11. In the Structure pane, click **<appname>Binding_Stub.xml** and edit the file, as described in Table 3–11, " Editing the <appname>Binding_Stub.xml File".
12. Invoke the web service.

To edit the <appname>Binding_Stub.xml File:

1. Provide the keystore password and sign and encryption key passwords.
2. In the inbound signature, specify the following:

```
<inbound><verify-signature><tbs-elements>
<tbs-element
name-space="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity
-utility-1.0.xsd" local-part="Timestamp" />
...
```

3. In the outbound signature, specify that the timestamp and UsernameToken should be signed, as follows:

```
<outbound>/<signature>/<tbs-elements>
<tbs-element
name-space="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd" local-part="Timestamp"/>
 <tbs-element
name-space="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity
-secext-1.0.xsd" local-part="UsernameToken"/>
...
```

4. In the outbound encryption, specify the key transport algorithm, as follows:

```
<outbound><encrypt>
<keytransport-method>RSA-OAEP-MGF1P</keytransport-method>
...
```

5. In the outbound encryption, specify that the UsernameToken should be encrypted, as follows:

```
<outbound>/<encrypt>/<tbe-elements>
<tbe-element local-part="UsernameToken"
name-space="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity
-secext-1.0.xsd" mode="CONTENT"/>
...
```

3.3.2 Configuring an OC4J 10*g* Web Service and an OWSM 12*c* Client

The following instructions tell how to configure an OC4J 10*g* web service and an OWSM 12*c* client to implement username token with message protection that conforms to the WS-Security 1.0 standard:

To configure the OC4J 10g Web Service:

1. Create and deploy a JAX-RPC web service on OC4J.
2. Use Application Server Control to secure the deployed web service.
3. Click **Authentication** tab and set the following options:

   a. Select **Use Username/Password Authentication**.

   b. Set **Password** to **Plain Text**.

4. Click **Integrity** tab in Inbound Policies page and set the following options:

   a. Select **Require Message Body to Be Signed**.

   b. Select **Verify Timestamp** and **Creation Time Required in Timestamp**.

   c. Enter the **Expiration Time** (in seconds).

5. Click **Integrity** tab in Outbound Policies page and set the following options:

   a. Select **Sign Body Element of Message**.

   b. Set the **Signature Method** to **RSA-SHA1**.

   c. Select **Add Timestamp** and **Creation Time Required in Timestamp**.

   d. Enter the **Expiration Time** (in seconds).

6. Click **Confidentiality** tab in the Inbound Policies page and set the following options:

   a. Select **Require Encryption of Message Body**.

7. Click **Confidentiality** tab in the Outbound Policies page and set the following options:

   a. Select **Encrypt Body Element of Message**.

   b. Set the **Encryption Method** to **AES-128**.

   c. Set the public key to encrypt.

8. Configure the keystore properties and identity certificates.

   > **Tip:** Ensure that you are using keystore with v3 certificates. By default, the JDK 1.5 keytool generates keystores with v1 certificates.

9. Edit the wsmgmt.xml deployment descriptor file, as described in Table 3–14, " Editing the wsmgmt.xml File".

To configure the OWSM 02c Client:

1. Create a client proxy for the OC4J 10*g* web service.

2. Attach the following policy: `oracle/wss10_username_token_with_message_protection_client_policy`.

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

3. Configure the policy.

   For more information, see "oracle/wss10_username_token_with_message_protection_client_policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*

4. Invoke the web service method from the client.

To edit the wsmgmt.xml File:

1. Find the `wsmgmt.xml` file under `*ORACLE_HOME*/j2ee/*oc4j_instance*/config/`.

2. In the inbound signature, specify the following:

   ```
   <inbound><verify-signature><tbs-elements>
   ```

```
<tbs-element
name-space="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity
-utility-1.0.xsd" local-part="Timestamp"/>
...
```

3. In the outbound signature, specify that the timestamp should be signed, as follows:

```
<outbound>/<signature>/<tbs-elements>
<tbs-element
name-space="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity
-utility-1.0.xsd" local-part="Timestamp"/>
...
```

4. In the outbound encryption, specify that the UsernameToken should be encrypted, as follows:

```
<outbound>/<encrypt>/<tbe-elements>
<tbe-element local-part="UsernameToken"
name-space="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity
-secext-1.0.xsd" mode="CONTENT"/>
...
```

3.4 SAML Token (Sender Vouches) with Message Protection (WS-Security 1.0)

This section tells how to implement SAML token sender vouches with message protection that conforms to the WS-Security 1.0 standard, the following interoperability scenarios:

- "Configuring an OWSM 12c Web Service and an OC4J 10g Client" on page 3-10
- "Configuring an OC4J 10g Web Service and an OWSM 12c Client" on page 3-12

3.4.1 Configuring an OWSM 12*c* Web Service and an OC4J 10*g* Client

The following instructions tell how to configure an OWSM 12*c* web service and an OC4J 10*g* client to implement SAML token sender vouches with message protection that conforms to the WS-Security 1.0 standard:

To configure the OWSM 12c Web Service:

1. Create an OWSM 12*c* web service.
2. Attach the following policy to the web service: `oracle/wss10_saml_token__with_message_protection_service_policy`

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

To configure the OC4J 10g client:

1. Create a client proxy for the web service (above) using Oracle JDeveloper.

   For more information, see Developing and Securing Web Services" in *Developing Applications with Oracle JDeveloper*.
2. Use the Oracle JDeveloper wizard to secure the proxy by right-clicking on the proxy project and selecting **Secure Proxy**.
3. Click **Authentication** in the Proxy Editor navigation bar and set the following options:
   a. Select **Use SAML Token**.

   b. Click **SAML Details**.
   c. Select **Sender Vouches Confirmation** and **Use Signature**.
   d. Enter the username that needs to be propagated as the **Default Subject Name**.
   e. Enter www.oracle.com as the **Default Issuer Name**.

4. Click **Inbound Integrity** in the Proxy Editor navigation bar and set the following options:
   a. Select **Verify Inbound Signed Request Body**.
   b. Select **Verify Timestamp** and **Creation Time Required in Timestamp**.
   c. Enter the **Expiration Time** (in seconds).
   d. Select all options under **Acceptable Signature Algorithms**.
5. Click **Outbound Integrity** in the Proxy Editor navigation bar and set the following options:
   a. Select **Sign Outbound Messages**.
   b. Select **Add Timestamp to Outbound Messages** and **Creation Time Required in Timestamp**.
   c. Enter the **Expiration Time** (in seconds).
6. Click **Inbound Confidentiality** in the Proxy Editor navigation bar and set the following options:
   a. Select **Decrypt Inbound Message Content**.
   b. Select all options under **Acceptable Signature Algorithms**.
7. Click **Outbound Confidentiality** in the Proxy Editor navigation bar and set the following options:
   a. Select **Encrypt Outbound Messages**.
   b. Set the Algorithm to **AES-128**.
8. Click **Keystore Options** in the Proxy Editor navigation bar and configure the keystore properties, as required.

   > **Note:** Ensure that you are using keystore with v3 certificates. By default, the JDK 1.5 keytool generates keystores with v1 certificates.

9. Click **OK** to close the wizard.
10. In the Structure pane, click **<appname>Binding_Stub.xml** and edit the file, as described in Table 3–17, " Editing the <appname>Binding_Stub.xml File".
11. Invoke the web service method.

To Edit the <appname>Binding_Stub.xml File:

1. Provide the keystore password and sign and encryption key passwords.
2. In the inbound signature, specify the following:

```
<inbound><verify-signature><tbs-elements>
<tbs-element
name-space="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity
-utility-1.0.xsd" local-part="Timestamp" />
...
```

3. In the outbound signature, specify that the timestamp should be signed, as follows:

   ```
   <outbound>/<signature>/<tbs-elements>
   <tbs-element
   name-space="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity
   -utility-1.0.xsd" local-part="Timestamp"/>
   ...
   ```

4. In the outbound encryption, specify the key transport algorithm, as follows:

   ```
   <outbound><encrypt>
   <keytransport-method>RSA-OAEP-MGF1P</keytransport-method>
   ...
   ```

3.4.2 Configuring an OC4J 10*g* Web Service and an OWSM 12*c* Client

The following instructions tell how to configure an OC4J 10*g* web service and an OWSM 12*c* client to implement SAML token sender vouches with message protection that conforms to the WS-Security 1.0 standard:

To configure the OC4J 10g Web Service:

1. Create and deploy a JAX-RPC web service on OC4J.
2. Use the Application Server Control to secure the deployed web service.
3. Click **Authentication** in navigation bar and set the following options:
   a. Select **Use SAML Authentication**.
   b. Select **Accept Sender Vouches**.
   c. Deselect **Verify Signature**.
4. Click **Inbound Integrity** in the navigation bar and set the following option:
   a. Select **Require Message Body To Be Signed**.
   b. Select **Verify Timestamp** and **Creation Time Required in Timestamp**.
   c. Enter the **Expiration Time** (in seconds).
5. Click **Outbound Integrity** in the navigation bar and select the following options:
   a. Select **Sign Body Element of Message**.
   b. Set the **Signature Method** to **RSA-SHA1**.
   c. Select **Add Timestamp** and **Creation Time Required in Timestamp**.
   d. Enter the **Expiration Time** (in seconds).
6. Click **Inbound Confidentiality** in the navigation bar and set the following option:
   a. Deselect **Require Encryption of Message Body**.
7. Click **Outbound Confidentiality** in the navigation bar and set the following option:
   1. Select **Encrypt Body Element of Message**.
   2. Set the **Encryption Method** to **AES-128**.
   3. Set the public key to encrypt.
8. Configure the keystore properties and identity certificates.

> **Note:** Ensure that you are using keystore with v3 certificates. By default, the JDK 1.5 keytool generates keystores with v1 certificates.

9. Edit the `wsmgmt.xml` deployment descriptor file, as described in Table 3–20, " Editing the wsmgmt.xml File".
10. Invoke the web service.

To configure the OWSM 12c Client:

1. Create a client proxy for the OC4J 10*g* web service.
2. Attach the following policy: `oracle/wss10_saml_token_with_message_protection_client_policy`.

   For more information, see "Attaching Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*
3. Configure the policy.

   For more information, see "oracle/wss10_saml_token_with_message_protection_client_policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*
4. Invoke the web service method from the client.

To edit the wsmgmt.xml File:

1. Find the `wsmgmt.xml` file in *`ORACLE_HOME`*`/j2ee/`*`oc4j_instance`*`/config`.
2. In the inbound signature, specify the following:

```
<inbound><verify-signature><tbs-elements>
<tbs-element
name-space="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity
-utility-1.0.xsd" local-part="Timestamp"/>
...
```

3. In the outbound signature, specify that the timestamp should be signed, as follows:

```
<outbound>/<signature>/<tbs-elements>
<tbs-element
name-space="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity
-utility-1.0.xsd" local-part="Timestamp"/>
...
```

4. In the outbound encryption, specify that the UsernameToken should be encrypted, as follows:

```
<outbound>/<encrypt>/<tbe-elements>
<tbe-element local-part="UsernameToken"
name-space="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity
-secext-1.0.xsd" mode="CONTENT"/>
...
```

3.5 Mutual Authentication with Message Protection (WS-Security 1.0)

This section tells how to implement mutual authentication with message protection that conforms to the WS-Security 1.0 standard, the following interoperability scenarios:

- "Configuring an OWSM 12c Web Service and an OC4J 10g Client" on page 3-13
- "Configuring an OC4J 10g Web Service and an OWSM 12c Client" on page 3-15

3.5.1 Configuring an OWSM 12*c* Web Service and an OC4J 10*g* Client

The following instructions tell how to configure an OWSM 12*c* web service and an OC4J 10*g* client to implement mutual authentication with message protection that conforms to the WS-Security 1.0 standard:

To configure the OWSM 12c Web Service:

1. Create a web service application.
2. Attach the following policy to the web service: `oracle/wss10_x509_token_with_message_protection_service_policy`.

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

To configure the OC4J 10g Client:

1. Create a client proxy for the web service (above) using Oracle JDeveloper.

   For more information, see Developing and Securing Web Services" in *Developing Applications with Oracle JDeveloper*.
2. Use the Oracle JDeveloper wizard to secure the proxy by right-clicking on the proxy project and selecting **Secure Proxy**.
3. Click **Authentication** in the Proxy Editor navigation bar and set the following options:
   a. Select **Use X509 To Authenticate**.
4. Click **Inbound Integrity** in the Proxy Editor navigation bar and set the following options:
   a. Select **Verify Inbound Signed Request Body**.
   b. Select **Verify Timestamp** and **Creation Time Required in Timestamp**.
   c. Enter the **Expiration Time** (in seconds).
   d. Select all options under **Acceptable Signature Algorithms**.
5. Click **Outbound Integrity** in the Proxy Editor navigation bar and set the following options:
   a. Select **Sign Outbound Messages**.
   b. Select **Add Timestamp to Outbound Messages** and **Creation Time Required in Timestamp**.
   c. Enter the **Expiration Time** (in seconds).
6. Click **Inbound Confidentiality** in the Proxy Editor navigation bar and set the following options:
   a. Select **Decrypt Inbound Message Content**.
   b. Select all options under **Acceptable Signature Algorithms**.
7. Click **Outbound Confidentiality** in the Proxy Editor navigation bar and set the following options:
   a. Select **Encrypt Outbound Messages**.
   b. Set the Algorithm to **AES-128**.
8. Click **Keystore Options** in the Proxy Editor navigation bar and configure the keystore properties, as required.

> **Note:** Ensure that you are using keystore with v3 certificates. By default, the JDK 1.5 keytool generates keystores with v1 certificates.

9. Click **OK** to close the wizard.
10. In the Structure pane, click **<appname>Binding_Stub.xml** and edit the file, as describe in Table 3–23, " Editing the <appname>Binding_Stub.xml File".
11. Invoke the web service.

To edit the <appname>Binding_Stub.xml file:

1. Provide the keystore password and sign and encryption key passwords.
2. In the inbound signature, specify the following:

   ```
   <inbound><verify-signature><tbs-elements>
   <tbs-element
   name-space="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity
   -utility-1.0.xsd" local-part="Timestamp" />
   ...
   ```

3. In the outbound signature, specify that the timestamp should be signed, as follows:

   ```
   <outbound>/<signature>/<tbs-elements>
   <tbs-element
   name-space="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity
   -utility-1.0.xsd" local-part="Timestamp"/>
   ...
   ```

4. In the outbound encryption, specify the key transport algorithm, as follows:

   ```
   <outbound><encrypt>
   <keytransport-method>RSA-OAEP-MGF1P</keytransport-method>
   ...
   ```

3.5.2 Configuring an OC4J 10*g* Web Service and an OWSM 12*c* Client

The following instructions tell how to configure an OC4J 10*g* web service and an OWSM 12*c* client to implement mutual authentication with message protection that conforms to the WS-Security 1.0 standard:

To configure the OC4J 10g Web Service:

1. Create and deploy a JAX-RPC web service on OC4J.
2. Use the Application Server Control to secure the deployed web service.
3. Click **Authentication** tab and set the following options:
   a. Select **Use X509 Certificate Authentication**.
4. Click **Integrity** tab of the Inbound Policies page and set the following options:
   a. Select **Require Message Body to Be Signed**.
   b. Select **Verify Timestamp** and **Creation Time Required in Timestamp**.
   c. Enter the **Expiration Time** (in seconds).
5. Click **Integrity** tab of the Outbound Policies page and set the following options:
   a. Select **Sign Body Element of Message**.
   b. Set the **Signature Method** to **RSA-SHA1**.

   c. Select **Add Timestamp** and **Creation Time Required in Timestamp**.

   d. Enter the **Expiration Time** (in seconds).

6. Click **Confidentiality** tab of the Inbound Policies page and set the following options:

   1. Select **Require Encryption of Message Body**.

7. Click **Confidentiality** tab of the Outbound Policies page and set the following options:

   a. Select **Encrypt Body Element of Message**.

   b. Set the **Encryption Method** to **AES-128**.

   c. Set the public key to encrypt.

8. Configure the keystore properties and identity certificates.

   > **Note:** Ensure that you are using keystore with v3 certificates. By default, the JDK 1.5 keytool generates keystores with v1 certificates.

9. Edit the `wsmgmt.xml` deployment descriptor file, as described in Table 3–26, " Editing the wsmgmt.xml File".

To configure the OWSM 12c Client:

1. Create a client proxy to the OC4J 10*g* web service.

2. Attach the following policy: `oracle/wss10_x509_token_with_message_protection_client_policy`.

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

3. Configure the policy.

   For more information, see "oracle/wss10_x509_token_with_message_protection_client_policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*

4. Invoke the web service.

To edit the wsmgmt.xml file:

1. Find the `wsmgmt.xml` file under *`ORACLE_HOME`*`/j2ee/`*`oc4j_instance`*`/config/`.

2. In the inbound signature, specify the following:

```
<inbound><verify-signature><tbs-elements>
<tbs-element
name-space="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity
-utility-1.0.xsd" local-part="Timestamp"/>
...
```

3. In the outbound signature, specify that the timestamp should be signed, as follows:

```
<outbound>/<signature>/<tbs-elements>
<tbs-element
name-space="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity
-utility-1.0.xsd" local-part="Timestamp"/>
...
```

4. In the outbound encryption, specify that the UsernameToken should be encrypted, as follows:

```
<outbound>/<encrypt>/<tbe-elements>
<tbe-element local-part="UsernameToken"
name-space="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity
-secext-1.0.xsd" mode="CONTENT"/>
...
```

3.6 Username Token Over SSL

This section tells how to implement username token over SSL, in the following interoperability scenarios:

- "Configuring an OWSM 12c Web Service and an OC4J 10g Client"
- "Configuring an OC4J 10g Web Service and an OWSM 12c Client"

For information about:

- Configuring SSL on WebLogic Server, see "Configuring Transport-Level Security (SSL)" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.
- Configuring SSL on OC4J, see http://download.oracle.com/docs/cd/B14099_19/web.1012/b14013/configssl.htm.

3.6.1 Configuring an OWSM 12*c* Web Service and an OC4J 10*g* Client

The following instructions tell how to configure an OWSM 12*c* web service and an OC4J 10*g* client to implement username token over SSL:

To configure the OWSM 12c Web Service:

1. Configure the server for SSL.

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

2. Attach one of the following policies to the web service:

```
oracle/wss_username_token_over_ssl_service_policy
```

```
oracle/wss_username_or_saml_token_over_ssl_service_policy
```

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

To configure the OC4J 10g Client:

1. Create a client proxy for the web service (above) using Oracle JDeveloper.

   Note: Ensure that the web service endpoint references the URL with HTTPS and SSL port configured on Oracle WebLogic Server.

   For more information, see Developing and Securing Web Services" in *Developing Applications with Oracle JDeveloper*.

2. Add the following code excerpt to initialize two-way SSL (at the beginning of the client proxy code):

```
HostnameVerifier hv = new HostnameVerifier()
httpsURLConnection.setDefaultHostnameVerifier(hv);
System.setProperty("javax.net.ssl.trustStore","<trust_store>");
System.setProperty("javax.net.ssl.trustStorePassword","<trust_store
_password>");
System.setProperty("javax.net.ssl.keyStore","<key_store>");
System.setProperty("javax.net.ssl.keyStorePassword","<key_store_password>");
System.setProperty("javax.net.ssl.keyStoreType","JKS");
```

3. Use the Oracle JDeveloper wizard to secure the proxy by right-clicking on the proxy project and selecting **Secure Proxy**.
4. Click **Authentication** in the Proxy Editor navigation bar and set the following options:
   a. Select **Use Username to Authenticate**.
   b. Deselect **Add Nonce** and **Add Creation Time**.
5. Click **Inbound Integrity** in the Proxy Editor navigation bar and deselect all options.
6. Click **Outbound Integrity** in the Proxy Editor navigation bar and deselect all options.
7. Click **Inbound Confidentiality** in the Proxy Editor navigation bar and deselect all options.
8. Click **Outbound Confidentiality** in the Proxy Editor navigation bar and deselect all options.
9. Click **Keystore Options** in the Proxy Editor navigation bar and configure the keystore properties, as required.

   > **Note:** Ensure that you are using keystore with v3 certificates. By default, the JDK 1.5 keytool generates keystores with v1 certificates.

10. Click **OK** to close the wizard.
11. In the Structure pane, click **<appname>Binding_Stub.xml** and edit the file. as described in. Table 3–29, " Editing the <appname>Binding_Stub.xml File"
12. Invoke the web service.

To edit the <appname>Binding_Stub.xml file:

1. Provide the keystore password and sign and encryption key passwords.
2. In the outbound signature, specify that the timestamp should be signed, as follows (and remove all other tags):

```
<outbound>
   <signature>
      <add-timestamp created="true" expiry="<Expiry_Time>"/>
   </signature>
...
```

3.6.2 Configuring an OC4J 10*g* Web Service and an OWSM 12*c* Client

The following instructions tell how to configure an OC4J 10*g* web service and an OWSM 12*c* client to implement username token over SSL:

To configure the OC4J 10g Web Service:

1. Configure the server for SSL.

For more information, see http://download.oracle.com/docs/cd/B14099_19/web.1012/b14013/configssl.htm

2. Use the Application Server Control to secure the deployed web service.
3. Click **Authentication** tab and set the following options:
   a. Select **Use Username/Password Authentication**.
4. Click **Integrity** tab of the Inbound Policies page and deselect all options.
5. Click **Integrity** tab of the Outbound Policies page and deselect all options.
6. Click **Confidentiality** tab of the Inbound Policies page and deselect all options.
7. Click **Confidentiality** tab of the Outbound Policies page and deselect all options.
8. Edit the `wsmgmt.xml` deployment descriptor file, as described in Table 3–32, " Editing the wsmgmt.xml File".

To configure the OWSM 12c client:

1. Create a client proxy to the OC4J 10*g* web service using `clientgen`.

   > **Note:** Ensure that the web service endpoint references the URL with HTTPS and SSL port configured on Oracle WebLogic Server.

2. Add the following code excerpt to initialize two-way SSL (at the beginning of the client proxy code):

```
HostnameVerifier hv = new HostnameVerifier()
httpsURLConnection.setDefaultHostnameVerifier(hv);
System.setProperty("javax.net.ssl.trustStore","<trust_store>");
System.setProperty("javax.net.ssl.trustStorePassword","<trust_store
_password>");
System.setProperty("javax.net.ssl.keyStore","<key_store>");
System.setProperty("javax.net.ssl.keyStorePassword","<key_store_password>");
System.setProperty("javax.net.ssl.keyStoreType","JKS");
```

3. Attach the following policy: `oracle/wss_username_token_over_ssl_client_policy`.

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

4. Configure the policy.

   For more information, see "oracle/wss_username_token_over_ssl_client_policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*

5. Invoke the web service.

To edit the wsmgmt.xml file:

1. Find the `wsmgmt.xml` file under *`ORACLE_HOME`*`/j2ee/`*`oc4j_instance`*`/config/`.
2. In the outbound signature, specify that the timestamp should be signed, as follows (and remove all other tags):

```
<outbound>
   <signature>
      <add-timestamp created="true" expiry="<Expiry_Time>"/>
   </signature>
...
```

3.7 SAML Token (Sender Vouches) Over SSL (WS-Security 1.0)

This section tells how to implement SAML token (sender vouches) over SSL that conforms to the WS-Security 1.0 standard, in the following interoperability scenarios:

- "Configuring an OWSM 12c Web Service and an OC4J 10g Client"
- "Configuring an OC4J 10g Web Service and an OWSM 12c Client"

For information about:

- Configuring SSL on WebLogic Server, see "Configuring Transport-Level Security (SSL)" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.
- Configuring SSL on OC4J, see `http://download.oracle.com/docs/cd/B14099_19/web.1012/b14013/configssl.htm`.

3.7.1 Configuring an OWSM 12*c* Web Service and an OC4J 10*g* Client

The following instructions tell how to configure an OWSM 12*c* web service and an OC4J 10*g* client to implement SAML token (sender vouches) over SSL that conforms to the WS-Security 1.0 standard:

To configure the OWSM 12c Web Service:

1. Configure the server for two-way SSL.

   For more information, see "Configuring SSL on WebLogic Server (Two-Way)" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

2. Attach the following policy to the web service:

```
oracle/wss_saml_token_over_ssl_service_policy
```

```
oracle/wss_username_or_saml_token_over_ssl_service_policy
```

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

To configure the OC4J 10g client:

1. Configure the server for two-way SSL.

   For more information, see `http://download.oracle.com/docs/cd/B14099_19/web.1012/b14013/configssl.htm`

2. Create a client proxy for the web service (above) using Oracle JDeveloper.

   > **Note:** Ensure that the web service endpoint references the URL with HTTPS and SSL port configured on Oracle WebLogic Server.

   For more information, see Developing and Securing Web Services" in *Developing Applications with Oracle JDeveloper*.

3. Add the following code excerpt to initialize two-way SSL (at the beginning of the client proxy code):

```
HostnameVerifier hv = new HostnameVerifier()
httpsURLConnection.setDefaultHostnameVerifier(hv);
System.setProperty("javax.net.ssl.trustStore","<trust_store>");
System.setProperty("javax.net.ssl.trustStorePassword","<trust_store
_password>");
System.setProperty("javax.net.ssl.keyStore","<key_store>");
System.setProperty("javax.net.ssl.keyStorePassword","<key_store_password>");
System.setProperty("javax.net.ssl.keyStoreType","JKS");
```

4. Use the Oracle JDeveloper wizard to secure the proxy by right-clicking on the proxy project and selecting **Secure Proxy**.
5. Click **Authentication** in the Proxy Editor navigation bar and set the following options:
   a. Select **Use SAML Token**.
   b. Click **SAML Details**.
   c. Select **Sender Vouches Confirmation**.
   d. Enter a valid username as the **Default Subject Name**.
6. Click **Inbound Integrity** in the Proxy Editor navigation bar and set the following option:
   a. Deselect **Verify Inbound Signed Message Body**.
7. Click **Outbound Integrity** in the Proxy Editor navigation bar and deselect all options.
8. Click **Inbound Confidentiality** in the Proxy Editor navigation bar and set the following option:
   a. Deselect **Decrypt Inbound Message Content**.
9. Click **Outbound Confidentiality** in the Proxy Editor navigation bar and set the following option:
   a. Deselect **Encrypt Outbound Message**.
10. Provide required information for the keystore to be used.
11. Click **OK** to close the wizard.
12. In the Structure pane, click **<appname>Binding_Stub.xml** and edit the file, as described in Table 3–35, " Editing the <appname>Binding_Stub.xml File".
13. Invoke the web service.

To edit the <appname>Binding_Stub.xml file:

1. Provide the keystore password and sign and encryption key passwords.
2. In the outbound signature, specify that the timestamp should be signed, as follows (and remove all other tags):

```
<outbound>
   <signature>
      <add-timestamp created="true" expiry="<Expiry_Time>"/>
   </signature>
...
```

3.7.2 Configuring an OC4J 10*g* Web Service and an OWSM 12*c* Client

The following instructions tell how to configure an OC4J 10*g* web service and an OWSM 12*c* client to implement SAML token (sender vouches) over SSL that conforms to the WS-Security 1.0 standard:

To configure the OC4J 10g Web Service:

1. Configure the server for two-way SSL.

For more information, see http://download.oracle.com/docs/cd/B14099_19/web.1012/b14013/configssl.htm

2. Use the Application Server Control to secure the deployed web service.
3. Click **Authentication** in navigation bar and set the following options:
   a. Select **Use SAML Authentication**.
   b. Select **Accept Sender Vouches**.
   c. Deselect **Verify Signature**.
4. Click **Integrity** tab of the Inbound Policies page and deselect all options.
5. Click **Integrity** tab of the Outbound Policies page and deselect all options.
6. Click **Confidentiality** tab of the Inbound Policies page and deselect all options.
7. Click **Confidentiality** tab of the Outbound Policies page and deselect all options.
8. Edit the wsmgmt.xml deployment descriptor file, as described in Table 3–38, " Edit the wsmgmt.xml File".

To configure the OWSM 12c Client:

1. Configure the server for two-way SSL.

   For more information, see "Configuring SSL on WebLogic Server (Two-Way)" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*

2. Create a client proxy to the OC4J 10*g* web service.

   For more information, see Ensure that the web service endpoint references the URL with HTTPS and SSL port configured on Oracle WebLogic Server.

3. Attach the following policy: `oracle/wss_saml_token_over_ssl_client_policy`.

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

4. Configure the policy.

   For more information, see "oracle/wss_saml_token_over_ssl_client_policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*

5. Invoke the web service.

To edit the wsmgmt.xml file:

1. Find the `wsmgmt.xml` file under *`ORACLE_HOME`*`/j2ee/`*`oc4j_instance`*`/config/`,.
2. In the outbound signature, specify that the timestamp should be signed, as follows (and remove all other tags):

```
<outbound>
   <signature>
      <add-timestamp created="true" expiry="<Expiry_Time>"/>
   </signature>
...
```

4

Interoperability with Oracle WebLogic Server 12*c* Web Service Security Environments

This chapter describes interoperability of Oracle Web Services Manager (OWSM) with Oracle WebLogic Server 12*c* web service security environments.

This chapter includes the following sections:

4.1 Overview of Interoperability with Oracle WebLogic Server 12*c* Web Service Security Environments

In Oracle Fusion Middleware 12*c*, you can attach both OWSM and Oracle WebLogic Server 12*c* web service policies to WebLogic Java EE web services.

For more information about:

- OWSM predefined policies, see "Predefined Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

- Configuring and attaching OWSM 12*c* policies, see "Securing Web Services" and "Attaching Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

For more details about the predefined Oracle WebLogic Server 12*c* web service policies, see:

- "Attaching Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*
- *Securing WebLogic Web Services for Oracle WebLogic Server*

Table 4–1 and Table 4–2 summarize the most common Oracle WebLogic Server 12*c* web service policy interoperability scenarios based on the following security requirements: authentication, message protection, and transport. The tables are organized as follows:

- Table 4–1 describes interoperability scenarios with WebLogic web service policies and OWSM client policies.
- Table 4–2 describes interoperability scenarios with OWSM web service policies and WebLogic web service client policies.

Table 4–1 WebLogic Web Service Policy and OWSM Client Policy Interoperability

Identity Token	WS-Security Version	Message Protection	Transport Security	Service Policy	Client Policy
Username	1.1	Yes	No	■ `Wssp1.2-2007-Wss1.1-UsernameToken-Plain-EncryptedKey-Basic128.xml` ■ `Wssp1.2-2007-SignBody.xml` ■ `Wssp1.2-2007-EncryptBody.xml`	`oracle/wss11_username_token_with_message_protection_client_policy`
Username and MTOM	1.1	Yes	No	■ `Wssp1.2-2007-Wss1.1-UsernameToken-Plain-EncryptedKey-Basic128.xml` ■ `Wssp1.2-2007-SignBody.xml` ■ `Wssp1.2-2007-EncryptBody.xml`	`oracle/wss11_username_token_with_message_protection_client_policy` `wsmtom_policy`
Username	1.0	Yes	No	■ `Wssp1.2-wss10_username_token_with_message_protection_owsm_policy.xml` ■ `Wssp1.2-2007-SignBody.xml` ■ `Wssp1.2-2007-EncryptBody.xml`	`oracle/wss10_username_token_with_message_protection_client_policy`

Table 4–1 (Cont.) WebLogic Web Service Policy and OWSM Client Policy Interoperability

Identity Token	WS-Security Version	Message Protection	Transport Security	Service Policy	Client Policy
SAML 2.0	1.1	Yes	No	Wssp1.2-wss11_saml_token_with_message_protection_owsm_policy.xml Wssp1.2-2007-SignBody.xml Wssp1.2-2007-EncryptBody.xml	oracle/wss11_saml_token_with_message_protection_client_policy
SAML	1.1	Yes	No	Wssp1.2-wss11_saml_token_with_message_protection_owsm_policy.xml Wssp1.2-2007-SignBody.xml Wssp1.2-2007-EncryptBody.xml	oracle/wss11_saml_token_with_message_protection_client_policy
SAML and MTOM	1.1	Yes	No	Wssp1.2-wss11_saml_token_with_message_protection_owsm_policy.xml Wssp1.2-2007-SignBody.xml Wssp1.2-2007-EncryptBody.xml	oracle/wss11_saml_token_with_message_protection_client_policy wsmtom_policy

Table 4–1 (Cont.) WebLogic Web Service Policy and OWSM Client Policy Interoperability

Identity Token	WS-Security Version	Message Protection	Transport Security	Service Policy	Client Policy
SAML	1.0	Yes	No	■ Wssp1.2-wss10_saml_token_with_message_protection_owsm_policy.xml ■ Wssp1.2-2007-SignBody.xml ■ Wssp1.2-2007-EncryptBody.xml	oracle/wss10_saml_token_with_message_protection_client_policy
Mutual Authentication	1.1	Yes	No	■ Wssp1.2-wss11_x509_token_with_message_protection_owsm_policy.xml ■ Wssp1.2-2007-SignBody.xml ■ Wssp1.2-2007-EncryptBody.xml	oracle/wss11_x509_token_with_message_protection_client_policy
Mutual Authentication	1.0	Yes	No	■ Wssp1.2-wss10_x509_token_with_message_protection_owsm_policy.xml ■ Wssp1.2-2007-SignBody.xml ■ Wssp1.2-2007-EncryptBody.xml	oracle/wss10_x509_token_with_message_protection_client_policy

Table 4–2 OWSM Web Service Policy and WebLogic Web Service Client Policy Interoperability

Identity Token	WS-Security Version	Message Protection	Transport Security	Service Policy	Client Policy
Username	1.1	Yes	No	oracle/wss11_username_token_with_message_protection_service_policy	▪ Wssp1.2-2007-Wss1.1-UsernameToken-Plain-EncryptedKey-Basic128.xml ▪ Wssp1.2-2007-SignBody.xml ▪ Wssp1.2-2007-EncryptBody.xml
Username and MTOM	1.1	Yes	No	oracle/wss11_username_token_with_message_protection_service_policy	▪ Wssp1.2-wss10_username_token_with_message_protection_owsm_policy.xml ▪ Wssp1.2-2007-SignBody.xml ▪ Wssp1.2-2007-EncryptBody.xml
Username	1.0	Yes	No	oracle/wss10_username_token_with_message_protection_service_policy	▪ Wssp1.2-wss10_username_token_with_message_protection_owsm_policy.xml ▪ Wssp1.2-2007-SignBody.xml ▪ Wssp1.2-2007-EncryptBody.xml
Username over SSL	1.0 and 1.1	No	Yes	oracle/wss_username_token_over_ssl_service_policy	Wssp1.2-2007-Https-UsernameToken-Plain.xml
Username over SSL with MTOM	1.0 and 1.1	No	Yes	oracle/wss_username_token_over_ssl_service_policy	Wssp1.2-2007-Https-UsernameToken-Plain.xml
SAML over SSL	1.0 and 1.1	No	Yes	oracle/wss_saml_token_over_ssl_service_policy	Wssp1.2-2007-Saml1.1-SenderVouches-Https.xml
SAML over SSL with MTOM	1.0 and 1.1	No	Yes	oracle/wss_saml_token_over_ssl_service_policy	Wssp1.2-2007-Saml1.1-SenderVouches-Https.xml
SAML 2.0	1.1	Yes	No	oracle/wss11_saml_token_with_message_protection_service_policy	▪ Wssp1.2-wss11_saml_token_with_message_protection_owsm_policy.xml ▪ Wssp1.2-2007-SignBody.xml ▪ Wssp1.2-2007-EncryptBody.xml

Table 4–2 (Cont.) OWSM Web Service Policy and WebLogic Web Service Client Policy Interoperability

Identity Token	WS-Security Version	Message Protection	Transport Security	Service Policy	Client Policy
SAML	1.1	Yes	No	`oracle/wss11_saml_token_with_message_protection_service_policy`	■ `Wssp1.2-wss11_saml_token_with_message_protection_owsm_policy.xml` ■ `Wssp1.2-2007-SignBody.xml` ■ `Wssp1.2-2007-EncryptBody.xml`
SAML with MTOM	1.1	Yes	No	`oracle/wss11_saml_token_with_message_protection_service_policy`	■ `Wssp1.2-wss11_saml_token_with_message_protection_owsm_policy.xml` ■ `Wssp1.2-2007-SignBody.xml` ■ `Wssp1.2-2007-EncryptBody.xml`
SAML	1.0	Yes	No	`oracle/wss10_saml_token_with_message_protection_service_policy`	■ `Wssp1.2-wss10_saml_token_with_message_protection_owsm_policy.xml` ■ `Wssp1.2-2007-SignBody.xml` ■ `Wssp1.2-2007-EncryptBody.xml`
Mutual Authentication	1.1	Yes	No	`oracle/wss11_x509_token_with_message_protection_service_policy`	■ `Wssp1.2-wss11_x509_token_with_message_protection_owsm_policy.xml` ■ `Wssp1.2-2007-SignBody.xml` ■ `Wssp1.2-2007-EncryptBody.xml`
Mutual Authentication	1.0	Yes	No	`oracle/wss10_x509_token_with_message_protection_service_policy`	■ `Wssp1.2-wss10_x509_token_with_message_protection_owsm_policy.xml` ■ `Wssp1.2-2007-SignBody.xml` ■ `Wssp1.2-2007-EncryptBody.xml`

4.2 Username Token With Message Protection (WS-Security 1.1)

This section tells how to implement username token with message protection that conforms to the WS-Security 1.1 standard, in the following interoperability scenarios:

- Interoperating with a WebLogic Web Service Policy
- Web Service Client Policy

4.2.1 Interoperating with a WebLogic Web Service Policy

The following instructions tell how to implement username token with message protection that conforms to the WS-Security 1.1 standard and ensure interoperability between the WebLogic web service policy and the OWSM client policy:

To Attach and Configure the WebLogic Web Service Policy:

1. Create a WebLogic web service.

   For more information, see "Roadmap for Implementing WebLogic (Java EE) Web Services" in *Understanding Web Services*

2. Attach the following policies:

   a. `Wssp1.2-2007-Wss1.1-UsernameToken-Plain-EncryptedKey-Basic128.xml`

   b. `Wssp1.2-2007-SignBody.xml`

   c. `Wssp1.2-2007-EncryptBody.xml`

   For more information, see "Updating the JWS File with @Policy and @Policies Annotations" in *Securing WebLogic Web Services for Oracle WebLogic Server*

3. Configure identity and trust stores.

   For more information, see "Configure identity and trust" in *Oracle WebLogic Server Administration Console Online Help*

4. Configure message-level security.

   Note: You only need to configure the Confidentiality Key for a WS-Security 1.1 policy.

   For more information, see:

   - "Configuring Message-Level Security" in *Securing WebLogic Web Services for Oracle WebLogic Server*
   - "Create a Web Service security configuration" in *Oracle WebLogic Server Administration Console Online Help*

5. Deploy the web service.

   For more information, see "Install a Web Service" in *Oracle WebLogic Server Administration Console Online Help*

To attach and Configure the OWSM Client Policy:

1. Create a client proxy for the web service created earlier using `clientgen` or some other mechanism.

   For more information, see "Roadmap for Implementing Oracle Fusion Middleware Web Services" in *Understanding Web Services*

2. Attach the following policy to the web service client: `oracle/wss11_username_token_with_message_protection_client_policy`.

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

3. Configure the policy.

For more information, see "oracle/wss11_username_token_with_message_protection_client_policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*

4. Specify `keystore.recipient.alias` in the client configuration.

   For more information, see "oracle/wss11_username_token_with_message_protection_client_policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*

5. Ensure that the `keystore.recipient.alias` keys specified for the client exist as trusted certificate entry in the trust store configured for the web service.

   For more information, see "oracle/wss11_username_token_with_message_protection_client_policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*

6. Provide a valid username and password as part of the configuration.

   For more information, see "oracle/wss11_username_token_with_message_protection_client_policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*

7. Invoke the web service method from the client.

   For more information, see "Roadmap for Implementing Oracle Fusion Middleware Web Services" in *Understanding Web Services*

4.2.2 Web Service Client Policy

The following instructions tell how to implement username token with message protection that conforms to the WS-Security 1.1 standard and ensure interoperability between the OWSM web service policy and the WebLogic web service client policy:

To attach and Configure the OWSM Policy:

1. Create and deploy a web service.

   For more information, see "Roadmap for Implementing Oracle Fusion Middleware Web Services" in *Understanding Web Services*

2. Attach the following policy to the web service:

   a. `oracle/wss11_username_token_with_message_protection_service_policy`.

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

To attach and Configure the WebLogic Web Service Client Policy:

1. Create a client proxy for the web service created using `clientgen`.

   For more information, see "Using the clientgen Ant Task to Generate Client Artifacts" in *Developing JAX-WS Web Services for Oracle WebLogic Server*

2. Attach the following policies:

   a. `Wssp1.2-2007-Wss1.1-UsernameToken-Plain-EncryptedKey-Basic128.xml`

   b. `Wssp1.2-2007-SignBody.xml`

   c. Wssp1.2-2007-EncryptBody.xml

      For more information, see "Updating the JWS File with @Policy and @Policies Annotations" in *Securing WebLogic Web Services for Oracle WebLogic Server*

3. Provide the configuration for the server (encryption key) in the client.

Note: Ensure that the encryption key specified is in accordance with the encryption key configured for the web service.

For more information, see "Updating a Client Application to Invoke a Message-Secured Web Service" in *Securing WebLogic Web Services for Oracle WebLogic Server*

4. Invoke the web service method from the client.

   For more information, see "Writing the Java Client Application Code to Invoke a Web Service" in *Developing JAX-WS Web Services for Oracle WebLogic Server*

4.3 Username Token With Message Protection (WS-Security 1.1) and MTOM

This section describes how to implement username token with message protection that conforms to the WS-Security 1.1 standard and uses Message Transmission Optimization Mechanism (MTOM), in the following interoperability scenarios:

- Interoperating with a WebLogic Web Service Policy
- Interoperating with a WebLogic Web Service Client Policy

4.3.1 Interoperating with a WebLogic Web Service Policy

The following instructions tell how to implement username token with message protection that conforms to the WS-Security 1.1 standard and uses Message Transmission Optimization Mechanism (MTOM), and to ensure interoperability between the WebLogic web service policy and the OWSM client policy:

To attach and Configure the WebLogic Web Service Policy:

1. Create a WebLogic web service.

   For more information, see "Roadmap for Implementing WebLogic (Java EE) Web Services" in *Understanding Web Services*.

2. Use the `@MTOM` annotation in the web service.

   For more information, see "Updating the JWS File with @Policy and @Policies Annotations" in *Securing WebLogic Web Services for Oracle WebLogic Server*

To attach and configure the OWSM Client Policy:

1. Configure the client proxy for the web service using `clientgen` or some other mechanism.

2. If you did not use the `@MTOM` annotation in the web services, attach `wsmtom_policy` from the Management tab.

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

4.3.2 Interoperating with a WebLogic Web Service Client Policy

The following instructions tell how to implement username token with message protection that conforms to the WS-Security 1.1 standard and uses Message Transmission Optimization Mechanism (MTOM), and to ensure interoperability between the OWSM web service policy and the WebLogic web service client policy:

To attach and configure the OWSM Policy:

1. Configure the OWSM web service.
2. Attach `wsmtom_policy` from the Management tab.

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

To attach and configure the WebLogic Web Service Client Policy:

1. Create a client proxy for the web service created using `clientgen`.
2. If you did not attach the `wsmtom_policy`, use the `@MTOM` annotation in the web service client.

4.4 Username Token With Message Protection (WS-Security 1.0)

This section describes how to implement username token with message protection that conforms to the WS-Security 1.0 standard, in the following interoperability scenarios:

- Interoperability with a WebLogic Web Service Policy
- Interoperability with a WebLogic Web Service Client Policy

Note: WS-Security 1.0 policy is supported for legacy applications only. Use WS-Security 1.1 policy for maximum performance. For more information, see "Username Token With Message Protection (WS-Security 1.1)" on page 4-6.

4.4.1 Interoperability with a WebLogic Web Service Policy

The following instructions tell how to implement username token with message protection that conforms to the WS-Security 1.0 standard and ensure interoperability between the WebLogic web service policy and the OWSM client policy:

To attach and configure the WebLogic Web Service Policy:

1. Create a WebLogic web service.

   For more information, see "Roadmap for Implementing WebLogic (Java EE) Web Services" in *Understanding Web Services*
2. Attach the following policies:

   a. `Wssp1.2-2007-SignBody.xml`

   b. `Wssp1.2-wss10_username_token_with_message_protection_owsm_policy.xml`

   c. Wssp1.2-2007-EncryptBody.xml

   For more information, see "Updating the JWS File with @Policy and @Policies Annotations" in *Securing WebLogic Web Services for Oracle WebLogic Server*
3. Configure identity and trust stores.

   For more information, see "Configure identity and trust" in *Oracle WebLogic Server Administration Console Online Help.*
4. Configure message-level security.

   For more information, see:

   - "Configuring Message-Level Security" in *Securing WebLogic Web Services for Oracle WebLogic Server*

- "Create a Web Service security configuration" in *Oracle WebLogic Server Administration Console Online Help*

5. Deploy the web service.

   For more information, see *Deploying Applications to Oracle WebLogic Server*.

To attach and configure the OWSM Client Policy:

1. Create a client proxy to the web service created using `clientgen` or some other mechanism.

   For more information, see "Roadmap for Implementing Oracle Fusion Middleware Web Services" in *Understanding Web Services*

2. Attach the following policy to the web service client:

   a. `oracle/wss10_username_token_with_message_protection_client_policy`.

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

3. Configure the policy.

   For more information, see "oracle/wss10_username_token_with_message_protection_client_policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*

   > **Note:** Ensure that you use different keys for client (sign and decrypt key) and keystore recipient alias (server public key used for encryption). Ensure that the recipient alias is in accordance with the keys defined in the web service policy security configuration.

4. Ensure that the signing and encryption keys specified for the client exist as trusted certificate entries in the trust store configured for the web service.

5. Provide a valid username and password as part of the configuration.

6. Invoke the web service method from the client.

   For more information, see "Roadmap for Implementing Oracle Fusion Middleware Web Services" in *Understanding Web Services*

4.4.2 Interoperability with a WebLogic Web Service Client Policy

The following instructions tell how to implement username token with message protection that conforms to the WS-Security 1.0 standard and ensure interoperability between the OWSM web service policy and the WebLogic web service client policy:

To attach and configure the OWSM Policy:

1. Create a web service.

   For more information, see "Roadmap for Implementing Oracle Fusion Middleware Web Services" in *Understanding Web Services*

2. Attach the following policy to the web service:

   a. `oracle/wss10_username_token_with_message_protection_service_policy`.

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

To attach and configure the WebLogic Web Service Client Policy:

1. Create a client proxy for the web service created using `clientgen`.

   For more information, see "Using the clientgen Ant Task to Generate Client Artifacts" in *Developing JAX-WS Web Services for Oracle WebLogic Server*

2. Attach the following policies:

   a. `Wssp1.2-wss10_username_token_with_message_protection_owsm_policy.xml`

   b. Wssp1.2-2007-SignBody.xml

   c. `Wssp1.2-2007-EncryptBody.xml`

   For more information, see "Updating the JWS File with @Policy and @Policies Annotations" in *Securing WebLogic Web Services for Oracle WebLogic Server*

3. Configure the client for server (encryption key) and client certificates.

   > **Note:** Ensure that the encryption key specified is in accordance with the encryption key configured for the web service.

   For more information, see "Updating a Client Application to Invoke a Message-Secured Web Service" in *Securing WebLogic Web Services for Oracle WebLogic Server*

4. Invoke the web service method from the client.

   For more information, see "Writing the Java Client Application Code to Invoke a Web Service" in *Developing JAX-WS Web Services for Oracle WebLogic Server*

4.5 Username Token Over SSL

This section how to implement username token over SSL, in the following interoperability scenario:

- Interoperating with a WebLogic Web Service Client Policy

4.5.1 Interoperating with a WebLogic Web Service Client Policy

The following instructions tell how to implement username token over SSL and ensure interoperability between the OWSM web service policy and the WebLogic web service client policy:

To attach and configure the OWSM Policy:

1. Configure the server for one-way SSL.

   For more information, see "Configuring SSL on WebLogic Server (One-Way)" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*

2. Create a web service.

   For more information, see "Roadmap for Implementing Oracle Fusion Middleware Web Services" in *Understanding Web Services*

3. Attach the following policy:

   a. `oracle/wss_username_token_over_ssl_service_policy`.

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

To attach and configure the WebLogic Web Service Client Policy:

1. Create a client proxy for the web service created using `clientgen`. Provide a valid username and password as part of the configuration for this policy in the client proxy.

   For more information, see "Using the clientgen Ant Task to Generate Client Artifacts" in *Developing JAX-WS Web Services for Oracle WebLogic Server*

2. Configure WebLogic Server for SSL.

   For more information, see "Configuring SSL on WebLogic Server (One-Way)" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*

3. Configure identity and trust stores.

   For more information, see "Configure identity and trust" in *Oracle WebLogic Server Administration Console Online Help*

4. Attach `Wssp1.2-2007-Https-UsernameToken-Plain.xml` to the web service client.

   For more information, see "Updating the JWS File with @Policy and @Policies Annotations" in *Securing WebLogic Web Services for Oracle WebLogic Server*

5. Provide the truststore and other required System properties in the SSL client.

   For more information, see "Using SSL Authentication in Java Clients" in *Developing Applications with the WebLogic Security Service*

6. Invoke the web service.

   For more information, see "Writing the Java Client Application Code to Invoke a Web Service" in *Developing JAX-WS Web Services for Oracle WebLogic Server*

4.6 Username Token Over SSL with MTOM

This section describes how to implement username token over SSL with Message Transmission Optimization Mechanism (MTOM), in the following interoperability scenario:

- Interoperating with a WebLogic Web Service Client Policy

4.6.1 Interoperating with a WebLogic Web Service Client Policy

The following instructions tell how to implement username token over SSL with MTOM and ensure interoperability between the OWSM web service policy and the WebLogic web service client policy:

To attach and configure the OWSM Policy:

1. Configure the OWSM web service.

To attach and configure the WebLogic Web Service Client Policy:

1. Create a client proxy for the web service created earlier.
2. Use the `@MTOM` annotation in the web service client.

   For more information, see "Updating the JWS File with @Policy and @Policies Annotations" in *Securing WebLogic Web Services for Oracle WebLogic Server*

4.7 SAML Token (Sender Vouches) Over SSL

This section describes how to implement SAML token sender vouches with SSL, in the following interoperability scenario:

- Interoperating with a WebLogic Web Service Client Policy

4.7.1 Interoperating with a WebLogic Web Service Client Policy

The following instructions tell how to implement SAML token sender vouches with SSL and ensure interoperability between the OWSM web service policy and the WebLogic web service client policy:

To attach and configure the OWSM Policy:

1. Configure the `oracle/wss_saml_token_over_ssl_service_policy` policy for two-way SSL.

   For more information, see "oracle/wss_saml_token_over_ssl_service_policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*

2. Create a web service.

   For more information, see "Roadmap for Implementing Oracle Fusion Middleware Web Services" in *Understanding Web Services*

3. Attach the following policy to the web service: `oracle/wss_saml_token_over_ssl_service_policy`.

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

To attach and configure the WebLogic Web Service Client Policy:

1. Configure the `oracle/wss_saml_token_over_ssl_service_policy` policy for two-way SSL.

   For more information, see "oracle/wss_saml_token_over_ssl_service_policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*

2. Create a web service.

   For more information, see "Roadmap for Implementing Oracle Fusion Middleware Web Services" in *Understanding Web Services*

3. Attach the following policy to the web service: `oracle/wss_saml_token_over_ssl_service_policy`.

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

To attach and configure the WebLogic Web Service Client Policy:

1. Create a client proxy for the web service created earlier using `clientgen`.

   For more information, see "Using the clientgen Ant Task to Generate Client Artifacts" in *Developing JAX-WS Web Services for Oracle WebLogic Server*

2. Configure Oracle WebLogic Server for two-way SSL.

   For more information, see "Configuring SSL on WebLogic Server (Two-Way)" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*

3. Configure identity and trust stores.

   For more information, see "Configure identity and trust" in *Oracle WebLogic Server Administration Console Online Help*

4. Attach `Wssp1.2-2007-Saml1.1-SenderVouches-Https.xml` to the web service client.

For more information, see "Updating the JWS File with @Policy and @Policies Annotations" in *Securing WebLogic Web Services for Oracle WebLogic Server*

5. Configure a SAML credential mapping provider.
6. In the WebLogic Server Administration Console, navigate to **Security Realms > RealmName > Providers > Credential Mapping** page and create a New Credential Mapping Provider of type SAMLCredentialMapperV2.
7. Select the new provider, click on Provider Specific, and configure it as follows:
   a. Set Issuer URI to `www.oracle.com`.
   b. Set Name Qualifier to `www.oracle.com`.
8. Restart Oracle WebLogic Server.

   For more information, see "Accessing Oracle WebLogic Administration Console" in *Administering Web Services*
9. Create a SAML relying party by setting the Profile to WSS/Sender-Vouches.
10. Configure the SAML relying party as follows (leave other values set to the defaults):
    a. Target URL: <*url_used_to_access_Web_service*>
    b. Description: <*your_description*>
    c. Select the Enabled checkbox and click **Save**.
    d. Ensure the Target URL is set to the URL used for the client web service.

       For more information, see "Create a SAML 1.1 Relying Party" in *Oracle WebLogic Server Administration Console Online Help*
11. Create a servlet and call the proxy code from the servlet.
12. Use BASIC authentication so that the authenticated subject can be created.
13. Provide the truststore and other required System properties in the SSL client.

    For more information, see "Using SSL Authentication in Java Clients" in *Developing Applications with the WebLogic Security Service*
14. Invoke the Web application client.
15. Enter the credentials of the user whose identity is to be propagated using the SAML token.

    For more information, see "Writing the Java Client Application Code to Invoke a Web Service" in *Developing JAX-WS Web Services for Oracle WebLogic Server*

4.8 SAML Token (Sender Vouches) Over SSL with MTOM

This section describes how to implement SAML token sender vouches over SSL with MTOM, in the following interoperability scenario:

- Interoperating with a WebLogic Web Service Client Policy

4.8.1 Interoperating with a WebLogic Web Service Client Policy

The following instructions tell how to implement SAML token vouches over SSL with MTOM and ensure interoperability between the OWSM web service policy and the WebLogic web service client policy:

To attach and configure the OWSM Policy:

1. Configure the OWSM web service.

   For more information, see "SAML Token (Sender Vouches) Over SSL"

To attach and configure the WebLogic Web Service Client Policy:

1. Configure the Oracle WebLogic web service client policy.

   For more information, see "SAML Token (Sender Vouches) Over SSL"

2. Use the `@MTOM` annotation in the web service client.

   For more information, see "Updating the JWS File with @Policy and @Policies Annotations" in *Securing WebLogic Web Services for Oracle WebLogic Server*.

4.9 SAML Token 2.0 (Sender Vouches) With Message Protection (WS-Security 1.1)

This section describes how to implement SAML 2.0 token sender vouches with message protection that conforms to the WS-Security 1.1 standard, in the following interoperability scenarios:

- Interoperating with a WebLogic Web Service Policy
- Interoperating with a WebLogic Web Service Client Policy

4.9.1 Interoperating with a WebLogic Web Service Policy

The following instructions tell how to implement SAML 2.0 token sender vouches with message protection that conforms to the WS-Security 1.1 standard and ensure interoperability between the WebLogic web service policy and the OWSM client policy:

To attach and configure the WebLogic Web Service Policy:

1. Create a WebLogic web service.

   For more information, see "Roadmap for Implementing WebLogic (Java EE) Web Services" in *Understanding Web Services*

2. Attach the following policies:

   - `Wssp1.2-2007-Saml2.0-SenderVouches-Wss1.1.xml`
   - `Wssp1.2-2007-SignBody.xml`
   - `Wssp1.2-2007-EncryptBody.xml`

   For more information, see "Updating the JWS File with @Policy and @Policies Annotations" in *Securing WebLogic Web Services for Oracle WebLogic Server*

3. Configure the keystore properties for message signing and encryption. The configuration should be in accordance with the keystore used on the server side. Create the trust store out of the keystore by exporting both keys, and trust both of them while importing into trust store. Configure identity and trust stores.

   For more information, see "Configure identity and trust" in *Oracle WebLogic Server Administration Console Online Help*.

4. Configure message-level security.

   For more information, see

- "Configuring Message-Level Security" in *Securing WebLogic Web Services for Oracle WebLogic Server*
- "Create a Web Service security configuration" in *Oracle WebLogic Server Administration Console Online Help*

5. Attach new configuration using the annotation:

   ```
   @WssConfiguration(value="<my_security_configuration>") where
   <my_security_configuration> is the name of the Web Security Configuration
   created in previous step.
   ```

   For more information, see "Configuring Message-Level Security" in *Securing WebLogic Web Services for Oracle WebLogic Server*

6. Deploy the web service.

   For more information, see *Deploying Applications to Oracle WebLogic Server*.

7. Create a SAML Identity Asserter.

   ```
   In the WebLogic Server Administration Console, navigate to Security Realms >
   RealmName > Providers > Credential Mapping page and create a New Credential
   Mapping Provider of type SAML2IdentityAsserter.
   ```

   For more information, see "Configure Authentication and Identity Assertion providers" in *Oracle WebLogic Server Administration Console Online Help*

8. Restart WebLogic Server.

   For more information, see "Start and stop servers" in the *Oracle WebLogic Server Administration Console Online Help*.

9. To add the identity provider to the identity asserter created in Step 7, perform the following steps:

   a. Select the identity asserter created in Step 7 in the WebLogic Administration Console.

   b. Create a new identity provider partner, select **New**, and then select **New Webservice Identity Provider Partner**.

   c. Provide a name, and select **Finish**.

10. Configure the identity provider as follows:

    a. Select the identity provide partner created in Step 9.

    b. Select the Enabled check box.

    c. Provide the Audience URI. For example:

    ```
    target:*:/saml20WLSWS-Project1-context-root/Class1Port
    ```

    d. Set Issuer URI to `www.oracle.com`.

    e. Set Target URL to <*url_used_to_access_Web_service*>.

    f. Set Profile to `WSS/Sender-Vouches`.

To attach and configure the OWSM Client Policy:

1. Generate a client using JDeveloper for the web service created earlier. Create a Web project and then select New, and create a client proxy using the WSDL.

   For more information, see

   - "Roadmap for Implementing Oracle Fusion Middleware Web Services" in *Understanding Web Services*

- "Developing and Securing Web Services" in *Developing Applications with Oracle JDeveloper*.

2. Add a servlet in the above project.

3. Attach the following policy to the web service client: `oracle/wss11_saml20_token_with_message_protection_client_policy`.

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

4. Specify `keystore.recipient.alias` in the client configuration.

   > **Note:** Ensure that `keystore.recipient.alias` is same as the decryption key specified for the web service.

   For more information, see "oracle/wss11_saml20_token_with_message_protection_cient_policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*

5. Ensure that the `keystore.recipient.alias` keys specified for the client exist as trusted certificate entry in the trust store configured for the web service.

   For more information, see "oracle/wss11_saml20_token_with_message_protection_cient_policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*

6. In JDeveloper, secure web project with Form-based authentication using the Configure ADF Security Wizard.

   For more information, see *Developing Applications with Oracle JDeveloper*

7. Invoke the Web application client.

   For more information, see "Roadmap for Implementing Oracle Fusion Middleware Web Services" in *Understanding Web Services*

4.9.2 Interoperating with a WebLogic Web Service Client Policy

The following instructions tell how to implement SAML 2.0 token sender vouches with message protection that conforms to the WS-Security 1.1 standard and ensure interoperability between the WebLogic web service client policy and the OWSM policy:

To attach and configure the OWSM Policy:

1. Create a web service.

   For more information, see "Roadmap for Implementing Oracle Fusion Middleware Web Services" in *Understanding Web Services*

2. Attach the following policy to the web service:

   `oracle/wss11_saml20_token_with_message_protection_service_policy.`
   For more information, see "Attaching Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*

To attach and configure the WebLogic Web Service Client:

1. Create a Java EE client for the deployed web service using JDeveloper. Create a Web project and create a proxy using WSDL proxy.

For more information, see "Creating JAX-WS Web Services and Clients" in *Developing Applications with Oracle JDeveloper*

2. Attach the following policies:

   - `Wssp1.2-2007-Saml2.0-SenderVouches-Wss1.1.xml`
   - `Wssp1.2-2007-SignBody.xml`
   - `Wssp1.2-2007-EncryptBody.xml`

   > **Note:** Extract `weblogic.jar` to a folder and provide the absolute path to the above policies files.

   For more information, see "Attaching Policies" in *Developing Applications with Oracle JDeveloper*

3. Add servlet to above web project.

4. Configure the client for server (encryption key) and client certificates.

   > **Note:** Ensure that the encryption key specified is in accordance with the decryption key configured for the web service.

   For more information, see "Updating a Client Application to Invoke a Message-Secured Web Service" in *Securing WebLogic Web Services for Oracle WebLogic Server*

5. Secure the Web application client using BASIC Authentication.

   For more information, see "Developing BASIC Authentication Web Applications" in *Developing Applications with the WebLogic Security Service*

6. Deploy the Java EE Web application client.

   For more information, see "Deploying Web Services Applications" in *Administering Web Services*

7. Configure a SAML credential mapping provider.

   - In the Oracle WebLogic Server Administration Console, navigate to **Security Realms > RealmName > Providers > Credential Mapping** page and create a New Credential Mapping Provider of type SAML2CredentialMapper.

8. Select the new provider, click on Provider Specific, and configure it as follows:

   1. Set Issuer URI to `www.oracle.com`.
   2. Set Name Qualifier to `www.oracle.com`.

   For more information, see "Configure Credential Mapping Providers" in *Oracle WebLogic Server Administration Console Online Help*

9. Restart WebLogic Server.

   For more information, see "Start and stop servers" in the *Oracle WebLogic Server Administration Console Online Help*.

10. To create a new service provider partner, perform the following steps:

    a. Select the credential mapper created in Step 7 in the WebLogic Administration Console, and then select the Management tab.

   b. Select **New**, and then select **New Webservice Service Provider Partner**.
   c. Provide a name, and select Finish.

11. Configure the service provider partner as follows:
    a. Select the service provide partner created in Step 10.
    b. Select the **Enabled** check box.
    c. Provide the Audience URI.
    d. Set Issuer URI to `www.oracle.com`.
    e. Set Target URL to <*url_used_to_access_Web_service*>.
    f. Set Profile to `WSS/Sender-Vouches`.
12. Invoke the Web application client.
    - Enter the credentials of the user whose identity is to be propagated using SAML token.

      For more information, see "Writing the Java Client Application Code to Invoke a Web Service" in *Developing JAX-WS Web Services for Oracle WebLogic Server*

4.10 SAML Token (Sender Vouches) with Message Protection (WS-Security 1.1)

This section describes how to implement SAML token sender vouches with message protection that conforms to the WS-Security 1.1 standard, in the following interoperability scenarios:

- Interoperating with a WebLogic Web Service Policy
- Interoperating with a WebLogic Web Service Client Policy

4.10.1 Interoperating with a WebLogic Web Service Policy

The following instructions tell how to implement SAML token sender vouches with message protection that conforms to the WS-Security 1.1 standard and ensure interoperability between the WebLogic web service policy and the OWSM client policy:

To attach and configure the WebLogic Web Service Policy:

1. Create a WebLogic web service.

   For more information, see "Roadmap for Implementing WebLogic (Java EE) Web Services" in *Understanding Web Services*
2. Attach the following policies:
   - `Wssp1.2-wss11_saml_token_with_message_protection_owsm_policy.xml`
   - `Wssp1.2-2007-SignBody.xml`
   - `Wssp1.2-2007-EncryptBody.xml`

     For more information, see "Updating the JWS File with @Policy and @Policies Annotations" in *Securing WebLogic Web Services for Oracle WebLogic Server*.
3. Configure identity and trust stores.

   For more information, see "Configure identity and trust" in *Oracle WebLogic Server Administration Console Online Help*

4. Configure message-level security.

   > **Note:** Since this is a WS-Security 1.1 policy, you need to configure Confidentiality Key only.

   For more information, see

   - "Configure Message-Level Security" in *Securing WebLogic Web Services for Oracle WebLogic Server*
   - "Create a Web Service security configuration" in *Oracle WebLogic Server Administration Console Online Help*.

5. Deploy the web service.

   For more information, see *Deploying Applications to Oracle WebLogic Server*.

6. Create a SAMLIdentityAsserterV2 authentication provider.

   - In the WebLogic Server Administration Console, navigate to **Security Realms > RealmName > Providers > Credential Mapping** page and create a New Credential Mapping Provider of type SAMLCredentialMapperV2.

     For more information, see "Configuring Authentication and Identity Assertion providers" in *Oracle WebLogic Server Administration Console Online Help*

7. Restart WebLogic Server.

   For more information, see "Start and stop servers" in the *Oracle WebLogic Server Administration Console Online Help*.

8. Select the authentication provider created in step 5.

9. Create a SAML asserting party.

   - Set Profile to `WSS/Sender-Vouches`.

   For more information, see "Create a SAML 1.1 Asserting Party" in *Oracle WebLogic Server Administration Console Online Help*

10. Configure the SAML asserting party as follows:

    a. Set Issuer URI to `www.oracle.com`.

    b. Set Target URL to <*url_used_to_access_Web_service*>.

       For more information, see "Create a SAML 1.1 Asserting Party" in *Oracle WebLogic Server Administration Console Online Help*

To attach and configure the OWSM Client Policy:

1. Create a client proxy to the web service created earlier using clientgen or some other mechanism.

   For more information, see "Roadmap for Implementing Oracle Fusion Middleware Web Services" in *Understanding Web Services*

2. Attach the following policy to the web service client:

   - `oracle/wss11_saml_token_with_message_protection_client_policy`.

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

3. Configure the policy, as described in `oracle/wss11_saml_token_with_message_protection_client_policy`.

For more information, see "oracle/wss11_saml_token_with_message_protection_client_policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*

4. Specify `keystore.recipient.alias` in the client configuration.

> **Note:** Ensure that `keystore.recipient.alias` is the same as the decryption key specified for the web service.

For more information, see "oracle/wss11_saml_token_with_message_protection_client_policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*

5. Ensure that the `keystore.recipient.alias` keys specified for the client exist as trusted certificate entry in the trust store configured for the web service.

   For more information, see "oracle/wss11_saml_token_with_message_protection_client_policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*

6. Provide a valid username whose identity needs to be propagated using SAML token in the client configuration.

   For more information, see "oracle/wss11_saml_token_with_message_protection_client_policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*

7. Invoke the Web application client.

   - Enter the credentials of the user whose identity is to be propagated using SAML token.

     For more information, see "Roadmap for Implementing Oracle Fusion Middleware Web Services" in *Understanding Web Services*

4.10.2 Interoperating with a WebLogic Web Service Client Policy

The following instructions tell how to implement SAML 2.0 sender vouches with message protection that conforms to the WS-Security 1.1 standard and ensure interoperability between the OWSM web service policy and the WebLogic web service client policy:

To attach and configure the OWSM Policy:

1. Create a web service.

   For more information, see "Roadmap for Implementing Oracle Fusion Middleware Web Services" in *Understanding Web Services*

2. Attach the following policy to the web service:

   - oracle/wss11_saml_token_with_message_protection_service_policy.

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

To attach and configure the WebLogic Web Service Client Policy:

1. Create a client proxy for the web service (above) using clientgen.

   For more information, see "Using the clientgen Ant Task to Generate Client Artifacts" in *Developing JAX-WS Web Services for Oracle WebLogic Server*

2. Attach the following policies:

   - `Wssp1.2-wss11_saml_token_with_message_protection_owsm_policy.xml`
   - `Wssp1.2-2007-SignBody.xml`
   - `Wssp1.2-2007-EncryptBody.xml`

     For more information, see "Updating the JWS File with @Policy and @Policies Annotations" in *Securing WebLogic Web Services for Oracle WebLogic Server*

3. Configure the client for server (encryption key) and client certificates.

   > **Note:** Ensure that the encryption key specified is in accordance with the decryption key configured for the web service.

   For more information, see "Updating a Client Application to Invoke a Message-Secured Web Service" in *Securing WebLogic Web Services for Oracle WebLogic Server*

4. Secure the Web application client using BASIC Authentication.

   For more information, see "Developing BASIC Authentication Web Applications" in *Developing Applications with the WebLogic Security Service*.

5. Deploy the web service client.

   For more information, see "Deploying Web Services Applications" in *Administering Web Services*

6. Configure a SAML credential mapping provider.

   - In the Oracle WebLogic Server Administration Console, navigate to **Security Realms > RealmName > Providers > Credential Mapping** page and create a New Credential Mapping Provider of type SAMLCredentialMapperV2.

7. Select the new provider, click on Provider Specific, and configure it as follows:

   a. Set Issuer URI to `www.oracle.com`.

   b. Set Name Qualifier to `www.oracle.com`.

   For more information, see "Configure Credential Mapping Providers" in *Oracle WebLogic Server Administration Console Online Help*

8. Restart WebLogic Server.

   For more information, see "Start and stop servers" in the *Oracle WebLogic Server Administration Console Online Help*.

9. Create a SAML relying party.

   - Set the Profile to **WSS/Sender-Vouches**.

     For more information, see "Create a SAML 1.1 Relying Party" and "Configure a SAML 1.1 Relying Party" in *Oracle WebLogic Server Administration Console Online Help*

10. Configure the SAML relying party.

    - Ensure the Target URL is set to the URL used for the client web service.

      For more information, see "Configure a SAML 1.1 Relying Party" in *Oracle WebLogic Server Administration Console Online Help*

11. Invoke the Web application client.

- Enter the credentials of the user whose identity is to be propagated using SAML token.

  For more information, see "Writing the Java Client Application Code to Invoke a Web Service" in *Developing JAX-WS Web Services for Oracle WebLogic Server*

4.11 SAML Token (Sender Vouches) with Message Protection (WS-Security 1.1) and MTOM

This section describes how to implement SAML token with sender vouches and message protection that conforms to the WS-Security 1.1 standard and uses Message Transmission Optimization Mechanism (MTOM), in the following interoperability scenarios:

- Interoperating with a WebLogic Web Service Policy
- Interoperating with a WebLogic Web Service Client Policy

4.11.1 Interoperating with a WebLogic Web Service Policy

The following instructions tell how to implement SAML token sender vouches with message protection that conforms to the WS-Security 1.1 standard and MTOM and ensure interoperability between the WebLogic web service policy and the OWSM client policy:

To attach and configure the WebLogic Web Service Policy:

1. Create a WebLogic web service, as described in Section 4.10, "SAML Token (Sender Vouches) with Message Protection (WS-Security 1.1)"

   For more information, see "Roadmap for Implementing WebLogic (Java EE) Web Services" in *Understanding Web Services*

2. Use the @MTOM annotation in the web service in Step 2 of " Attaching and Configuring the WebLogic Web Service Policy".

   For more information, see "Updating the JWS File with @Policy and @Policies Annotations" in *Securing WebLogic Web Services for Oracle WebLogic Server*

To attach and configure the OWSM Client Policy:

1. Create a client proxy to the web service created earlier, as described in Section 4.10, "SAML Token (Sender Vouches) with Message Protection (WS-Security 1.1)"

   For more information, see "Roadmap for Implementing Oracle Fusion Middleware Web Services" in *Understanding Web Services*

2. Attach `wsmtom_policy` from the Management tab.

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

4.11.2 Interoperating with a WebLogic Web Service Client Policy

The following instructions tell how to implement SAML token sender vouches with message protection that conforms to the WS-Security 1.1 standard and MTOM and ensure interoperability between the OWSM web service policy and the WebLogic web service client policy:

To attach and configure the OWSM Policy:

1. Create and deploy a web service.

For more information, see "Roadmap for Implementing Oracle Fusion Middleware Web Services" in *Understanding Web Services*

2. Attach the following policy to the web service:

   - `oracle/wss11_username_token_with_message_protection_service_policy`.

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

To attach and configure the WebLogic Web Service Client Policy:

1. Create a client proxy for the web service created earlier using clientgen.

   For more information, see "Using the clientgen Ant Task to Generate Client Artifacts" in *Developing JAX-WS Web Services for Oracle WebLogic Server*

2. Attach the following policies:

   - `Wssp1.2-2007-Wss1.1-UsernameToken-Plain-EncryptedKey-Basic128.xml`
   - `Wssp1.2-2007-SignBody.xml`
   - `Wssp1.2-2007-EncryptBody.xml`

   For more information, see "Updating the JWS File with @Policy and @Policies Annotations" in *Securing WebLogic Web Services for Oracle WebLogic Server*

3. Provide the configuration for the server (encryption key) in the client.

   > **Note:** Ensure that the encryption key specified is in accordance with the encryption key configured for the web service.

   For more information, see "Updating a Client Application to Invoke a Message-Secured Web Service" in *Securing WebLogic Web Services for Oracle WebLogic Server*

4. Invoke the web service method from the client.

   For more information, see "Writing the Java Client Application Code to Invoke a Web Service" in *Developing JAX-WS Web Services for Oracle WebLogic Server*

4.12 SAML Token (Sender Vouches) with Message Protection (WS-Security 1.0)

This section describes how to implement SAML token with sender vouches and message protection that conforms to the WS-Security 1.0 standard, in the following interoperability scenarios:

- Interoperating with a WebLogic Web Service Policy
- Interoperating with a WebLogic Web Service Client Policy

> **Note:** WS-Security 1.0 policy is supported for legacy applications only. Use WS-Security 1.1 policy for maximum performance. For more information, see "SAML Token (Sender Vouches) with Message Protection (WS-Security 1.1)" on page 4-20.

4.12.1 Interoperating with a WebLogic Web Service Policy

The following instructions tell how to implement SAML token with sender vouches and message protection that conforms to the WS-Security 1.0 standard and ensure interoperability between the WebLogic web service policy and the OWSM client policy:

To attach and configure the WebLogic Web Service Policy:

1. Create a WebLogic web service.

   For more information, see "Roadmap for Implementing WebLogic (Java EE) Web Services" in *Understanding Web Services*

2. Attach the following policies:

   - `Wssp1.2-wss10_saml_token_with_message_protection_owsm_policy.xml`
   - `Wssp1.2-2007-SignBody.xml`
   - Wssp1.2-2007-EncryptBody.xml

3. Configure identity and trust stores.

   For more information, see "Configure identity and trust" in *Oracle WebLogic Server Administration Console Online Help*

4. Configure message-level security.

   For more information, see

   - "Configuring Message-Level Security" in *Securing WebLogic Web Services for Oracle WebLogic Server*
   - "Create a Web Service security configuration" in *Oracle WebLogic Server Administration Console Online Help*

5. Deploy the web service.

   For more information, see *Deploying Applications to Oracle WebLogic Server*.

6. Create a SAMLIdentityAsserterV2 authentication provider.

   - In the WebLogic Server Administration Console, navigate to **Security Realms > RealmName > Providers > Credential Mapping** page and create a New Credential Mapping Provider of type SAMLCredentialMapperV2.

     For more information, see "Configure Authentication and Identity Assertion providers" in *Oracle WebLogic Server Administration Console Online Help*

7. Restart WebLogic Server.

   For more information, see "Start and stop servers" in the *Oracle WebLogic Server Administration Console Online Help*.

8. Select the authentication provider created in step 5.

9. Create a SAML asserting party.

   - Set Profile to **WSS/Sender-Vouches**.

     For more information, see "Create a SAML 1.1 Asserting Party" in *Oracle WebLogic Server Administration Console Online Help*

10. Configure the SAML asserting party as follows (leave other values set to the defaults):

    a. Set Issuer URI to `www.oracle.com`.

   b. Set Target URL to <*url_used_by_client*>.

      For more information, see "Configure a SAML 1.1 Asserting Party" in *Oracle WebLogic Server Administration Console Online Help*

To attach and configure the OWSM Client Policy:

1. Create a client proxy to the web service created earlier using clientgen or some other mechanism.

   For more information, see "Roadmap for Implementing Oracle Fusion Middleware Web Services" in *Understanding Web Services*

2. Attach the following policy to the web service client:

   - `oracle/wss10_saml_token_with_message_protection_client_policy`.

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

3. Configure the policy.

   For more information, see "oracle/wss10_saml_token_with_message_protection_client_policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*

4. Ensure that you use different keys for client (sign and decrypt key) and keystore recipient alias (server public key used for encryption). Ensure that the recipient alias is in accordance with the keys defined in the web service policy security configuration.

   For more information, see "oracle/wss10_saml_token_with_message_protection_client_policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*

5. Ensure that the signing and encryption keys specified for the client exist as trusted certificate entries in the trust store configured for the web service.

   For more information, see "oracle/wss10_saml_token_with_message_protection_client_policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*

6. Provide valid username whose identity needs to be propagated using SAML token in the client configuration.

   For more information, see "oracle/wss10_saml_token_with_message_protection_client_policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*

7. Invoke the web service method.

   For more information, see "Roadmap for Implementing Oracle Fusion Middleware Web Services" in *Understanding Web Services*

4.12.2 Interoperating with a WebLogic Web Service Client Policy

The following instructions tell how to implement SAML token with message protection that conforms to the WS-Security 1.0 standard and ensure interoperability between the OWSM web service policy and the WebLogic web service client policy:

To attach and configure the OWSM Policy:

1. Create a web service.

For more information, see "Roadmap for Implementing Oracle Fusion Middleware Web Services" in *Understanding Web Services*

2. Attach the following policy to the web service:

   - `oracle/wss10_saml_token_with_message_protection_service_policy`.

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

To attach and configure the WebLogic Web Service Client Policy:

1. Create a client proxy for the web service (above) using clientgen.

   For more information, see "Using the clientgen Ant Task to Generate Client Artifacts" in *Developing JAX-WS Web Services for Oracle WebLogic Server*

2. Attach the following policies:

   - `Wssp1.2-wss10_saml_token_with_message_protection_owsm_policy.xml`
   - `Wssp1.2-2007-SignBody.xml`
   - `Wssp1.2-2007-EncryptBody.xml`

     For more information, see "Updating the JWS File with @Policy and @Policies Annotations" in *Securing WebLogic Web Services for Oracle WebLogic Server*

3. Configure the client for server (encryption key) and client certificates.

   > **Note:** Ensure that the encryption key specified is in accordance with the decryption key configured for the web service.

   For more information, see "Updating a Client Application to Invoke a Message-Secured Web Service" in *Securing WebLogic Web Services for Oracle WebLogic Server*

4. Secure the Web application client using BASIC Authentication.

   For more information, see "Developing BASIC Authentication Web Applications" in *Developing Applications with the WebLogic Security Service*

5. Deploy the web service client.

   For more information, see "Deploying Web Services Applications" in *Administering Web Services*

6. Configure a SAML credential mapping provider.

   - In the WebLogic Server Administration Console, navigate to **Security Realms > RealmName > Providers > Credential Mapping** page and create a New Credential Mapping Provider of type SAMLCredentialMapperV2.

     For more information, see "Configure Credential Mapping Providers" in *Oracle WebLogic Server Administration Console Online Help*

7. Select the SAMLCredentialMapperV2, click on Provider Specific, and configure it as follows:

   a. Set Issuer URI to `www.oracle.com`.

   b. Set Name Qualifier to `www.oracle.com`.

8. Restart WebLogic Server.

For more information, see "Start and stop servers" in the *Oracle WebLogic Server Administration Console Online Help*.

9. Create a SAML relying party. Set the profile to `WSS/Sender-Vouches`.

   For more information, see "Create a SAML 1.1 Relying Party" in *Oracle WebLogic Server Administration Console Online Help*

10. Configure the SAML relying party.

    Note: Ensure the target URL is set to the URL used for the client web service.

    For more information, see "Configure a SAML 1.1 Relying Party" in *Oracle WebLogic Server Administration Console Online Help*

11. Invoke the Web application client and enter the appropriate credentials.

    For more information, see "Writing the Java Client Application Code to Invoke a Web Service" in *Developing JAX-WS Web Services for Oracle WebLogic Server*

4.13 Mutual Authentication with Message Protection (WS-Security 1.0)

This section tells how to implement mutual authentication with message protection that conform to the WS-Security 1.0 standard, in the following interoperability scenarios:

- Interoperating with a WebLogic Web Service Policy
- Interoperating with a WebLogic Web Service Client Policy

4.13.1 Interoperating with a WebLogic Web Service Policy

The following instructions tell how to implement mutual authentication with message protection that conforms to the WS-Security 1.0 standard and ensure interoperability between the WebLogic web service policy and the OWSM client policy:

To attach and configure the WebLogic Web Service Policy:

1. Create a WebLogic web service.

   For more information, see "Roadmap for Implementing WebLogic (Java EE) Web Services" in *Understanding Web Services*

2. Attach the following policies:

   a. `Wssp1.2-wss10_x509_token_with_message_protection_owsm_policy.xml`

   b. `Wssp1.2-2007-SignBody.xml`

   c. Wssp1.2-2007-EncryptBody.xml

      For more information, see "Updating the JWS File with @Policy and @Policies Annotations" in *Securing WebLogic Web Services for Oracle WebLogic Server*

3. Configure identity and trust stores.

   For more information, see "Configure identity and trust" in *Oracle WebLogic Server Administration Console Online Help*

4. Configure message-level security.

   For more information, see

- "Configuring Message-Level Security" in *Securing WebLogic Web Services for Oracle WebLogic Server*
- "Create a Web Service security configuration" in *Oracle WebLogic Server Administration Console Online Help*

5. Create and configure token handlers for X.509 and for username token. In WebLogic Administration Console, navigate to the Web Service Security page of the domain and create the token handlers as described below.
6. Create a token handle for username token and configure the following:
   a. Name: <*name*>
   b. Class name:

      `weblogic.xml.crypto.wss.UsernameTokenHandler`
   c. Token Type: `ut`
   d. Handling Order: `1`
7. Create a token handler for X.509 and configure the following:
   a. Name: <*name*>
   b. Class name:

      `weblogic.xml.crypto.wss.BinarySecurityTokenHandler`
   c. Token Type: `x509`
   d. Handling Order: `0`
8. For the X.509 token handler, add the following properties:
   a. Name: `UserX509ForIdentity`
   b. Value: `true`
   c. IsEncrypted: `False`
9. Configure a credential mapping provider. Create a PKICredentialMapper and configure it as follows (leave all other values set to the defaults):
   a. Keystore Provider: N/A
   b. Keystore Type: `jks`
   c. Keystore File Name: `default_keystore.jks`
   d. Keystore Pass Phrase: <*password*>
   e. Confirm Keystore Pass Phrase: <*password*>
10. Configure Authentication by Selecting the **Authentication** tab and configure as follows:
    a. Click **DefaultIdentityAsserter** and add **X.509** to **Chosen** active types
    b. Click **Provider Specific** and configure the following:

       Default User Name Mapper Attribute Type: `CN`

       Active Types: `X.509`

       Use Default User Name Mapper: `True`

       For more information, see "Configure Authentication and Identity Assertion providers" in *Oracle WebLogic Server Administration Console Online Help*.

11. If the users are not added, add the Common Name (CN) user specified in the certificate.

    For more information, see "Create users" in *Oracle WebLogic Server Administration Console Online Help*

12. Restart Oracle WebLogic Server.

13. Deploy the web service.

    For more information, see "Install a Web Service" in *Oracle WebLogic Server Administration Console Online Help*

To attach and configure the OWSM Client Policy:

1. Create a client proxy to the web service created earlier using `clientgen` or some other mechanism.

   For more information, see "Roadmap for Implementing Oracle Fusion Middleware Web Services" in *Understanding Web Services*

2. Attach the following policy to the client:

   `wss10_x509_token_with_message_protection_client_policy`
   For more information, see "Attaching Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*

3. Provide the configuration for the server (encryption key) in the client.

   > **Note:** Ensure that the encryption key specified is in accordance with the encryption key configured for the web service.

   For more information, see "Updating a Client Application to Invoke a Message-Secured Web Service" in *Securing WebLogic Web Services for Oracle WebLogic Server*

4. Invoke the web service method from the client.

   For more information, see "Roadmap for Implementing Oracle Fusion Middleware Web Services" in *Understanding Web Services*

4.13.2 Interoperating with a WebLogic Web Service Client Policy

The following instructions tell how to implement username token with message protection that conforms to the WS-Security 1.0 standard and ensure interoperability between the OWSM web service policy and the WebLogic web service client policy:

To attach and configure the OWSM Policy:

1. Create and deploy a web service application.

   For more information, see "Roadmap for Implementing Oracle Fusion Middleware Web Services" in *Understanding Web Services*

2. Attach the following policy to the web service:

   `oracle/wss10_x509_token_with_message_protection_service_policy.`

To attach and configure the WebLogic Web Service Client Policy:

1. Create a client proxy for the web service created earlier using clientgen.

   For more information, see "Using the clientgen Ant Task to Generate Client Artifacts" in *Developing JAX-WS Web Services for Oracle WebLogic Server*

2. Attach the following policies:

   - `Wssp1.2-wss10_x509_token_with_message_protection_owsm_policy.xml`
   - Wssp1.2-2007-SignBody.xml
   - `Wssp1.2-2007-EncryptBody.xml`

     For more information, see "Updating the JWS File with @Policy and @Policies Annotations" in *Securing WebLogic Web Services for Oracle WebLogic Server*

3. Provide the configuration for the server (encryption key) in the client.

   > **Note:** Ensure that the encryption key specified is in accordance with the encryption key configured for the web service.

   For more information, see "Updating a Client Application to Invoke a Message-Secured Web Service" in *Securing WebLogic Web Services for Oracle WebLogic Server*

4. Invoke the web service method from the client.

   For more information, see "Writing the Java Client Application Code to Invoke a Web Service" in *Developing JAX-WS Web Services for Oracle WebLogic Server*

4.14 Mutual Authentication with Message Protection (WS-Security 1.1)

This section tells how to implement mutual authentication with message protection that conform to the WS-Security 1.1 standards, in the following interoperability scenarios:

- Interoperating with a WebLogic Web Service Policy
- Interoperating with a WebLogic Web Service Client Policy

4.14.1 Interoperating with a WebLogic Web Service Policy

The following instructions tell how to implement mutual authentication with message protection that conforms to the WS-Security 1.1 standard and ensure interoperability between the WebLogic web service policy and the OWSM client policy:

To attach and configure the WebLogic Web Service Policy:

1. Create a WebLogic web service.

   For more information, see "Roadmap for Implementing WebLogic (Java EE) Web Services" in *Understanding Web Services*

2. Attach the following policies:

   - `Wssp1.2-wss11_x509_token_with_message_protection_owsm_policy.xml`
   - `Wssp1.2-2007-SignBody.xml`
   - `Wssp1.2-2007-EncryptBody.xml`

     For more information, see "Updating the JWS File with @Policy and @Policies Annotations" in *Securing WebLogic Web Services for Oracle WebLogic Server*

3. Configure identity and trust stores.

   For more information, see "Configure identity and trust" in *Oracle WebLogic Server Administration Console Online Help*

4. Configure message-level security.

   For more information, see

   - "Configuring Message-Level Security" in *Securing WebLogic Web Services for Oracle WebLogic Server*
   - "Create a Web Service security configuration" in *Oracle WebLogic Server Administration Console Online Help*

5. Create and configure token handlers for X.509 and for username token. In WebLogic Administration Console, navigate to the Web Service Security page of the domain and create the token handlers as described below.

6. Create a token handle for username token and configure the following:

   - Name: <*name*>
   - Class name: `weblogic.xml.crypto.wss.UsernameTokenHandler`
   - Token Type: `ut`
   - Handling Order: `1`
   - Create a token handle for username token and configure the following:

Create a token handler for X.509 and configure the following:

   - Name: <*name*>
   - Class name: `weblogic.xml.crypto.wss.BinarySecurityTokenHandler`
   - Token Type: `x509`
   - Handling Order: `0`

For the X.509 token handler, add the following properties:

   - Name: `UserX509ForIdentity`
   - Value: `true`
   - IsEncrypted: `False`

     For more information, see "Create a token handler of a Web Service security configuration" in *Oracle WebLogic Server Administration Console Online Help*.

7. Configure a credential mapping provider by creating a PKICredentialMapper and configure it as follows (leave all other values set to the defaults):

   - Keystore Provider: N/A
   - Keystore Type: `jks`
   - Keystore File Name: `default_keystore.jks`
   - Keystore Pass Phrase: <*password*>
   - Confirm Keystore Pass Phrase: <*password*>

     For more information, see "Configure Credential Mapping Providers" in *Oracle WebLogic Server Administration Console Online Help*

8. Configure Authentication by selecting the **Authentication** tab and configure as follows:

   - Click **DefaultIdentityAsserter** and add **X.509** to **Chosen** active types
   - Click **Provider Specific** and configure the following:
   - Default User Name Mapper Attribute Type: `CN`

- Active Types: `X.509`
- Use Default User Name Mapper: `True`

9. If the users are not added, add the Common Name (CN) user specified in the certificate.

   For more information, see "Create users" in *Oracle WebLogic Server Administration Console Online Help*

10. Restart Oracle WebLogic Server.

11. Deploy the web service.

    For more information, see "Install a Web Service" in *Oracle WebLogic Server Administration Console Online Help*

To attach and configure the OWSM Client Policy:

1. Create a client proxy for the web service created earlier using clientgen or some other mechanism.

   For more information, see "Roadmap for Implementing Oracle Fusion Middleware Web Services" in *Understanding Web Services*

2. Attach the following policy to the client

   ```
   wss11_x509_token_with_message_protection_client_policy
   ```

3. Edit the policy as follows:

   ```
   <orasp:x509-token
     orasp:sign-key-ref-mech="thumbprint"
     orasp:enc-key-ref-mech="thumbprint"/>
   ```

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

4. Provide the configuration for the server (encryption key) in the client.

   > **Note:** Ensure that the encryption key specified is in accordance with the encryption key configured for the web service.

   For more information, see "Updating a Client Application to Invoke a Message-Secured Web Service" in Securing WebLogic Web Services for Oracle WebLogic Server.

5. Invoke the web service method from the client.

   For more information, see "Roadmap for Implementing Oracle Fusion Middleware Web Services" in *Understanding Web Services*

4.14.2 Interoperating with a WebLogic Web Service Client Policy

The following instructions tell how to implement mutual authentication with message protection that conforms to the WS-Security 1.1 standard and ensure interoperability between the OWSM web service policy and the WebLogic web service client policy:

To attach and configure the OWSM Policy:

1. Create and deploy a web service.

   For more information, see "Roadmap for Implementing Oracle Fusion Middleware Web Services" in *Understanding Web Services*

2. Attach the following policy to the web service:

`oracle/wss11_x509_token_with_message_protection_service_policy.`
For more information, see "Attaching Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*

To attach and configure the WebLogic Web Service Client Policy:

1. Create a client proxy for the web service created earlier using clientgen.

   For more information, see "Using the clientgen Ant Task to Generate Client Artifacts" in *Developing JAX-WS Web Services for Oracle WebLogic Server*

2. Attach the following policies:

   - `Wssp1.2-wss11_x509_token_with_message_protection_owsm_policy.xml`
   - `Wssp1.2-2007-SignBody.xml`
   - `Wssp1.2-2007-EncryptBody.xml`

     For more information, see "Updating the JWS File with @Policy and @Policies Annotations" in *Securing WebLogic Web Services for Oracle WebLogic Server*

3. Provide the configuration for the server (encryption key) in the client.

   > **Note:** Ensure that the encryption key specified is in accordance with the encryption key configured for the web service.

   For more information, see "Updating a Client Application to Invoke a Message-Secured Web Service" in *Securing WebLogic Web Services for Oracle WebLogic Server*

4. Invoke the web service method from the client.

   For more information, see "Writing the Java Client Application Code to Invoke a Web Service" in *Developing JAX-WS Web Services for Oracle WebLogic Server*

5 Interoperability with Microsoft WCF/.NET 3.5 Security Environments

This chapter describes interoperability of Oracle Web Services Manager (OWSM) with Microsoft WCF/.NET 3.5 security environments.

This chapter includes the following sections:

5.1 Overview of Interoperability with Microsoft WCF/.NET 3.5 Security Environments

In conjunction with Microsoft, Oracle has performed interoperability testing to ensure that the web service security policies created using OWSM 12*c* can interoperate with web service policies configured using Microsoft Windows Communication Foundation (WCF)/.NET 3.5 Framework and vice versa.

For more information about Microsoft WCF/.NET 3.5 Framework, see `http://msdn.microsoft.com/en-us/netframework/aa663324.aspx`.

For more information about:

- OWSM predefined policies, see "Predefined Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.
- Configuring and attaching OWSM 12*c* policies, see "Securing Web Services" and "Attaching Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

Note: In most cases, you can attach OWSM policies in source code, before deploying an application, or you can attach policies post deployment, using WLST or Fusion Middleware Control. To simplify the instructions in this chapter, it is assumed that you are attaching policies at runtime. If a situation *requires* that you attach a policy before deploying, it is described that way in the instructions.

Note: Some of the procedures described in this chapter instruct you to use the Microsoft ServiceModel Metadata Utility Tool (`SvcUtil.exe`) to create a client proxy and configuration file from the deployed web service. However, `SvcUtil.exe` does not work with certain security policy assertions used with OWSM. As a workaround when generating a WCF proxy for a web service protected by an OWSM policy, do the following:

- Detach the policy.
- Generate the proxy using `SvcUtil.exe`.
- Re-attach the policy.

For more information about `SvcUtil.exe`, see `http://msdn.microsoft.com/en-us/library/aa347733%28v=vs.90%29.aspx`.

Table 5–1 and Table 5–2 summarize the most common Microsoft .NET 3.5 interoperability scenarios based on the following security requirements: authentication, message protection, and transport.

Note: In the following scenarios, ensure that you are using a keystore with v3 certificates. By default, the JDK 1.5 keytool generates keystores with v1 certificates.

In addition, ensure that the keys use the proper extensions, including `DigitalSignature`, `Non_repudiation`, `Key_Encipherment`, and `Data_Encipherment`.

Table 5–1 OWSM 12c Service Policy and Microsoft WCF/.NET 3.5 Client Policy Interoperability

Identity Token	WS-Security Version	Message Protection	Transport Security	Service Policy	Client Policy
MTOM	NA	NA	NA	`oracle/wsmtom_policy`	See Table 5–4, " Configuring the Microsoft WCF/.NET 3.5 Client" on page 5-4

Username or SAML	1.1	Yes	No	`oracle/wss11_username_token_with_message_protection_service_policy`
				OR
				`oracle/wss11_saml_or_username_token_with_message_protection_service_policy`
					See Table 5–8, " Configuring the Microsoft WCF/.NET 3.5 Client"See on page 5-8

Username	1.0 and 1.1	No	Yes	`oracle/wss_saml_or_username_token_over_ssl_service_policy`
				OR
				`oracle/wss_username_token_over_ssl_service_policy`
					See Table 5–12, " Configuring the Microsoft WCF/.NET 3.5 Client"See on page 5-16

Mutual Authentication	1.1	Yes	No	`oracle/wss11_x509_token_with_message_protection_service_policy`	See Table 5–15, " Configuring the Microsoft WCF/.NET 3.5 Client" on page 5-20
Kerberos	1.1	Yes	No	`oracle/wss11_kerberos_token_with_message_protection_service_policy`	See Table 5–21, " Configuration Prerequisites for Interoperability" on page 5-29

Table 5–2 Microsoft WCF/.NET 3.5 Service Policy and OWSM 12c Client Policy Interoperability

Identity Token	WS-Security Version	Message Protection	Transport Security	Service Policy	Client Policy
MTOM	NA	NA	NA	See Table 5–5, " Configuring the Microsoft WCF/.NET 3.5 Web Service" on page 5-5	`oracle/wsmtom_policy`
Username	1.1	Yes	No	See Table 5–9, " Configuring the Microsoft WCF/.NET 3.5 Web Service" on page 5-12	`oracle/wss11_username_token_with_message_protection_client_policy`
Mutual Authentication	1.1	Yes	No	See Table 5–10, " Configuring the OWSM 02c Client"	`oracle/wss11_x509_token_with_message_protection_client_policy`

5.2 Message Transmission Optimization Mechanism (MTOM)

This section describes how to implement MTOM in the following interoperability scenarios:

- Configuring an OWSM 12c Web Service and a Microsoft WCF/.NET 3.5 Client
- Configuring a Microsoft WCF/.NET 3.5 Web Service and an OWSM 12c Client

5.2.1 Configuring an OWSM 12*c* Web Service and a Microsoft WCF/.NET 3.5 Client

The following instructions tell how to configure an OWSM 12*c* web service and a Microsoft WCF/.NET 3.5 Client to implement Message Transmission Optimization Mechanism (MTOM).

To Configure the OWSM 12c Web Service:

1. Create and deploy a web service application.
2. Attach the following policy to the web service: `oracle/wsmtom_policy`.

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

To Configure the Microsoft WCF/.NET 3.5 Client:

1. Use the Microsoft SvcUtil utility to create a client proxy and configuration file from the deployed web service. See Example 5–1, "app.config File for MTOM Interoperability".

   `http://msdn.microsoft.com/en-us/library/aa347733%28v=vs.90%29.aspx`

2. Run the client program.

   See the *app.config File for MTOM Interoperability* sample:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <system.serviceModel>
        <bindings>
            <customBinding>
                <binding name="CustomBinding_IMTOMService">
                    <mtomMessageEncoding maxReadPoolSize="64"
                     maxWritePoolSize="16"
                        messageVersion="Soap12" maxBufferSize="65536"
                        writeEncoding="utf-8">
                        <readerQuotas maxDepth="32" maxStringContentLength=
                         "8192" maxArrayLength="16384"
                            maxBytesPerRead="4096"
maxNameTableCharCount="16384" />
                    </mtomMessageEncoding>
                    <httpTransport manualAddressing="false"
maxBufferPoolSize="524288"
                        maxReceivedMessageSize="65536" allowCookies="false"
                           authenticationScheme="Anonymous"
                        bypassProxyOnLocal="false"
hostNameComparisonMode="StrongWildcard"
                        keepAliveEnabled="true" maxBufferSize="65536"
                           proxyAuthenticationScheme="Anonymous"
                        realm="" transferMode="Buffered"
                           unsafeConnectionNtlmAuthentication="false"
                        useDefaultWebProxy="true" />
                </binding>
            </customBinding>
        </bindings>
        <client>
          <endpoint address="<endpoint_url>"
              binding="customBinding"
bindingConfiguration="CustomBinding_IMTOMService"
              contract="IMTOMService" name="CustomBinding_IMTOMService" >
          </endpoint>
        </client>
    </system.serviceModel>
```

```
</configuration>
```

5.2.2 Configuring a Microsoft WCF/.NET 3.5 Web Service and an OWSM 12*c* Client

The following instructions tell how to configure a Microsoft WCF/.NET 3.5 web service and an OWSM 12*c* client to implement Message Transmission Optimization Mechanism (MTOM).

To Configure the Microsoft WCF/.NET 3.5 Web Service:

1. Create a .NET web service.

   For an example, see Example 5–2, ".NET Web Service for MTOM Interoperability".

   For more information, see "How to: Define a Windows Communication Foundation Service Contract" at `http://msdn.microsoft.com/en-us/library/ms731835.aspx`.

2. Deploy the application.

To Configure the OWSM 12c Client:

1. Using JDeveloper, create a SOA composite that consumes the .NET web service.

   For more information, see Developer's Guide for SOA Suite.

2. Attach the following policy to the web service client:

   `oracle/wsmtom_policy`

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

   See the following *.NET Web Service for MTOM Interoperability* sample:

```
static void Main(string[] args)
{
    string uri = "http://host:port/TEST/MTOMService/SOA/MTOMService";
    // Step 1 of the address configuration procedure: Create a URI to serve as
the base address.
    Uri baseAddress = new Uri(uri);

    // Step 2 of the hosting procedure: Create ServiceHost
    ServiceHost selfHost = new ServiceHost(typeof(MTOMService), baseAddress);

    try {
        HttpTransportBindingElement hb = new HttpTransportBindingElement();
        hb.ManualAddressing = false;
        hb.MaxBufferPoolSize = 2147483647;
        hb.MaxReceivedMessageSize = 2147483647;
        hb.AllowCookies = false;
        hb.AuthenticationScheme = System.Net.AuthenticationSchemes.Anonymous;
        hb.KeepAliveEnabled = true;
        hb.MaxBufferSize = 2147483647;
        hb.ProxyAuthenticationScheme =
System.Net.AuthenticationSchemes.Anonymous;
        hb.Realm = "";
        hb.TransferMode = System.ServiceModel.TransferMode.Buffered;
        hb.UnsafeConnectionNtlmAuthentication = false;
        hb.UseDefaultWebProxy = true;
        MtomMessageEncodingBindingElement me = new
MtomMessageEncodingBindingElement();
```

```
        me.MaxReadPoolSize=64;
        me.MaxWritePoolSize=16;
        me.MessageVersion=System.ServiceModel.Channels.MessageVersion.Soap12;
        me.WriteEncoding = System.Text.Encoding.UTF8;
        me.MaxWritePoolSize = 2147483647;
        me.MaxBufferSize = 2147483647;
        me.ReaderQuotas.MaxArrayLength = 2147483647;
        CustomBinding binding1 = new CustomBinding();
        binding1.Elements.Add(me);
        binding1.Elements.Add(hb);
        ServiceEndpoint ep = selfHost.AddServiceEndpoint(typeof(IMTOMService),
binding1,
               "MTOMService");
        EndpointAddress myEndpointAdd = new EndpointAddress(new Uri(uri),
        EndpointIdentity.CreateDnsIdentity("WSMCert3"));
        ep.Address = myEndpointAdd;

        // Step 4 of the hosting procedure: Enable metadata exchange.
        ServiceMetadataBehavior smb = new ServiceMetadataBehavior();
        smb.HttpGetEnabled = true;
        selfHost.Description.Behaviors.Add(smb);
        using (ServiceHost host = new ServiceHost(typeof(MTOMService)))
        {
            System.ServiceModel.Description.ServiceDescription svcDesc =
                 selfHost.Description;
            ServiceDebugBehavior svcDebug =
                  svcDesc.Behaviors.Find<ServiceDebugBehavior>();
            svcDebug.IncludeExceptionDetailInFaults = true;
        }

        // Step 5 of the hosting procedure: Start (and then stop) the service.
        selfHost.Open();
        Console.WriteLine("The service " + uri + " is ready.");
        Console.WriteLine("Press <ENTER> to terminate service.");
        Console.WriteLine();
        Console.ReadLine();
        // Close the ServiceHostBase to shutdown the service.
        selfHost.Close();
    }
    catch (CommunicationException ce)
    {
        Console.WriteLine("An exception occurred: {0}", ce.Message);
        selfHost.Abort();
    }
}
```

5.3 Username Token With Message Protection (WS-Security 1.1)

This section describes how to implement username token with message protection that conforms to the WS-Security 1.1 standard--with or without secure conversation enabled--in the following interoperability scenarios:

- Configuring an OWSM 12c Web Service and a Microsoft WCF/.NET 3.5 Client
- Configuring a Microsoft WCF/.NET 3.5 Web Service and an OWSM 12c Client

5.3.1 Configuring an OWSM 12*c* Web Service and a Microsoft WCF/.NET 3.5 Client

The following instructions tell how to configure a OWSM 12*c* web service and a Microsoft WCF/.NET 3.5 client to implement username token with message protection that conforms to the WS-Security 1.1 standard, both with and without secure conversation enabled.

To Configure the OWSM 12c Web Service:

1. Create a web service application.
2. Select the policy to use based on whether or not you want to enable secure conversation:

   If you do not want to enable secure conversation, clone either of the following policies:

   ```
   oracle/wss11_username_token_with_message_protection_service_policy
   ```

   ```
   oracle/wss11_saml_or_username_token_with_message_protection_service_pol
   icy
   ```

   To enable secure conversation, clone the following policy:

   ```
   oracle/wss11_username_token_with_message_protection_wssc_service_policy
   ```

   > **Note:** In the case of secure conversation enabled, you will have to configure the `app.config` file somewhat differently, as described in Table 5–8.

   For more information, see "Attaching Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.
3. Export the X.509 certificate file from the keystore on the service side to a `.cer` file (for example, `alice.cer`) using the following command:

   ```
   keytool -export -alias alice -file C:\alice.cer -keystore default-keystore.jks
   ```

To Configure the Microsoft WCF/.NET 3.5 Client:

1. Import the certificate file (exported previously) to the keystore on the client server using Microsoft Management Console (mmc), as follows:

   a. Open a command prompt.

   b. Type **mmc** and press **Enter**.

   c. Select **File > Add/Remove snap-in**.

   d. Select **Add and Choose Certificates**.

   > **Note:** To view certificates in the local machine store, you must be in the Administrator role.

   e. Select **Add**.

   f. Select **My user account and finish**.

   g. Click **OK**.

   h. Expand **Console Root > Certificates -Current user > Personal > Certificates**.

   i. Right-click on **Certificates** and select **All tasks > Import** to launch Certificate import Wizard.

   j. Click **Next**, select **Browse**, and navigate to the `.cer` file that was exported previously.

k. Click **Next** and accept defaults and finish the wizard.

For more information, see "How to: View Certificates with the MMC Snap-in" at http://msdn.microsoft.com/en-us/library/ms788967.aspx.

2. Generate a .NET client using the WSDL of the web service.

   For more information, see "How to: Create a Windows Communication Foundation Client" at http://msdn.microsoft.com/en-us/library/ms733133(v=vs.90).aspx.

3. In the Solution Explorer of the client project, add a reference by right-clicking on references, selecting Add reference, and browsing to `C:\Windows\Microsoft.NET\framework\v3.0\Windows Communication Foundation\System.Runtime.Serialization.dll`.

4. Edit the `app.config` file in the .NET project to update the certificate file and disable replays, as shown in Example 5–3, "app.config File" (Changes are identified in **bold**.)

   If you follow the default key setup, then `<certificate_cn>` should be set to `alice`.

5. Edit the `app.config` file as needed to enable to enable secure conversation or not.

   If you do not want to enable secure conversation, edit the `app.config` as shown in Example 5–3:

   - Set the `authenticationMode` property of the `<security>` element to `UserNameOverTransport`.
   - Do not configure the properties of the `secureConversationBootstrap` element.

   To enable secure conversation, edit the `app.config` file as shown the comments in bold italics in Example 5–3:

   - Set the `authenticationMode` property of the `<security>` element to `SecureConversation`.
   - Configure the `secureConversationBootstrap` element with additional properties, as shown in the example.

6. Compile the project.

7. Open a command prompt and navigate to the project's Debug folder.

8. Enter `<client_project_name>.exe` and press **Enter**.

   See the following *app.config file* sample:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.serviceModel>
    <behaviors>
      <endpointBehaviors>
         <behavior name="secureBehaviour">
           <clientCredentials>
             <serviceCertificate>
               <defaultCertificate findValue="<certificate_cn>"
                storeLocation="CurrentUser" storeName="My"
                x509FindType="FindBySubjectName"/>
             </serviceCertificate>
           </clientCredentials>
         </behavior>
      </endpointBehaviors>
    </behaviors>
  <bindings>
    <customBinding>
      <binding name="HelloWorldSoapHttp">
      <!-- To enable secrure conversation, use
           authenticationMode="SecureConversation"
           instead of the value for authenticationMode shown below -->
      <security
        authenticationMode="UserNameOverTransport"
        defaultAlgorithmSuite="Basic128"
        requireDerivedKeys="false"
        securityHeaderLayout="Lax"
        includeTimestamp="true"
        keyEntropyMode="CombinedEntropy"
        messageProtectionOrder="SignBeforeEncrypt"

messageSecurityVersion="WSSecurity11WSTrustFebruary2005WSSecureConversationFebr
uary2005WSSecurityPolicy11BasicSecurityProfile10"
        requireSignatureConfirmation="true">
        <localClientSettings
          cacheCookies="true"
          detectReplays="false"
          replayCacheSize="900000"
          maxClockSkew="00:05:00"
          maxCookieCachingTime="Infinite"
          replayWindow="00:05:00"
          sessionKeyRenewalInterval="10:00:00"
          sessionKeyRolloverInterval="00:05:00"
          reconnectTransportOnFailure="true"
          timestampValidityDuration="00:05:00"
          cookieRenewalThresholdPercentage="60"/>
        <localServiceSettings detectReplays="true"
issuedCookieLifetime="10:00:00"
          maxStatefulNegotiations="128"
          replayCacheSize="900000"
          maxClockSkew="00:05:00"
          negotiationTimeout="00:01:00" replayWindow="00:05:00"
          inactivityTimeout="00:02:00"
          sessionKeyRenewalInterval="15:00:00"
          sessionKeyRolloverInterval="00:05:00"
          reconnectTransportOnFailure="true" maxPendingSessions="128"
          maxCachedCookies="1000" timestampValidityDuration="00:05:00" />
        <secureConversationBootstrap />
        <!--
        To enable secure conversation, add the following properties to
        the <secureConversationBootstrap> element:
             <secureConversationBootstrap
               authenticationMode="UserNameOverTransport"
               requireDerivedKeys="false"
               securityHeaderLayout="Lax"
               includeTimestamp="true"
               keyEntropyMode="CombinedEntropy"
               messageProtectionOrder="SignBeforeEncrypt"

messageSecurityVersion="WSSecurity11WSTrustFebruary2005WSSecureConversationFebr
uary2005WSSecurityPolicy11BasicSecurityProfile10"
               requireSignatureConfirmation="true"/> -->
         -->
      </security>
      <textMessageEncoding
      maxReadPoolSize="64"
      maxWritePoolSize="16"
      messageVersion="Soap11"
      writeEncoding="utf-8">
        <readerQuotas
         maxDepth="32"
         maxStringContentLength="8192"
         maxArrayLength="16384"
         maxBytesPerRead="4096"
         maxNameTableCharCount="16384" />
     </textMessageEncoding>
     <HttpTransport
      manualAddressing="false"
      maxBufferPoolSize="524288"
      maxReceivedMessageSize="65536"
      allowCookies="false"
      authenticationScheme="Anonymous"
      bypassProxyOnLocal="false"
      hostNameComparisonMode="StrongWildcard"
      keepAliveEnabled="true"
      maxBufferSize="65536"
      proxyAuthenticationScheme="Anonymous"
      realm=""
      transferMode="Buffered"
      unsafeConnectionNtlmAuthentication="false"
      useDefaultWebProxy="true" />
      </binding>
    </customBinding>
  </bindings>
    <client>
      <endpoint address="<endpoint_url>"
       binding="customBinding"
       bindingConfiguration="HelloWorldSoapHttp"
       contract="HelloWorld"
       name="HelloWorldPort"
       behaviorConfiguration="secureBehaviour" >
        <identity>
          <dns value="<certificate_cn>"/>
        </identity>
      </endpoint>
    </client>
  </system.serviceModel>
</configuration>
```

5.3.2 Configuring a Microsoft WCF/.NET 3.5 Web Service and an OWSM 12*c* Client

The following instructions tell how to configure a Microsoft WCF/.NET 3.5 web service and an OWSM 12*c* client to implement username token with message protection that conforms to the WS-Security 1.1 standard.

To Configure the Microsoft WCF/.NET 3.5 Web Service:

1. Create a .NET web service.

   Be sure to create a custom binding for the web service using the `SymmetricSecurityBindingElement`. For an example, see Example 5–4, "Example of .NET Web Service".

   For more information, see "How to: Define a Windows Communication Foundation Service Contract" at `http://msdn.microsoft.com/en-us/library/ms731835.aspx`.

2. Create and import a certificate file to the keystore on the web service server.

   Using Microsoft Visual Studio, the command would be similar to the following:

   ```
   makecert -r -pe -n "CN=wsmcert3" -sky exchange -ss my C:\wsmcert3.cer
   ```

   This command creates and imports a certificate in mmc.

   If the command does not provide expected results, then try the following sequence of commands. You need to download Windows Developer Kit (WDK) at `http://www.microsoft.com/whdc/devtools/WDK/default.mspx`.

   ```
   makecert -r -pe -n "CN=wsmcert3" -sky exchange -ss my -sv wscert3.pvk
   C:\wsmcert3.cer
   pvk2pfx.exe -pvk wscert3.pvk -spc wsmcert3.cer -pfx PRF_WSMCert3.pfx -pi
   welcome1
   ```

   Then, in mmc, import `PRF_WSMCert3.pfx`.

3. Import the certificate created on the web service server to the client server using the `keytool` command. For example:

   ```
   keytool -import -alias wsmcert3 -file C:\wsmcert3.cer -keystore
   <owsm_client_keystore>
   ```

4. Right-click on the web service Solution project in Solutions Explorer and click **Open Folder In Windows Explorer**.

5. Navigate to the `bin/Debug` folder.

6. Double-click the `<project>.exe` file. This command runs the web service at the URL provided.

To Configure the OWSM 12c Client:

1. Using JDeveloper, create a SOA composite that consumes the .NET web service.

   For more information, see *Developer's Guide for SOA Suite.*

2. In JDeveloper, create a partner link using the WSDL of the .NET service.

3. Attach the following policy to the web service client: `oracle/wss11_username_token_with_message_protection_client_policy`.

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

4. Provide configurations for the `csf-key` and `keystore.recipient.alias`.

   You can specify this information when attaching the policy, by overriding the policy configuration. For more information.

   Ensure that you configure the `keystore.recipient.alias` as the alias of the certificate imported in step 1 (`wsmcert3`). For example:

   ```
   <wsp:PolicyReference
         URI="oracle/wss11_username_token_with_message_protection_client_policy"
         orawsp:category="security"
         orawsp:status="enabled"/>
      <property
         name="csf-key"
         type="xs:string"
         many="false">
         basic.credentials
      </property>
      <property
         name="keystore.recipient.alias"
         type="xs:string"
   ```

```
         many="false">
         wsmcert3
      </property>
```

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

See the following *.NET Web Service* sample:

```
static void Main(string[] args)
{
    // Step 1 of the address configuration procedure: Create a URI to serve as
the
    // base address.
    // Step 2 of the hosting procedure: Create ServiceHost
    string uri = "http://host:port/TEST/NetService";
    Uri baseAddress = new Uri(uri);

    ServiceHost selfHost = new ServiceHost(typeof(CalculatorService),
baseAddress);

    try
    {
        SymmetricSecurityBindingElement sm =

SymmetricSecurityBindingElement.CreateUserNameForCertificateBindingElement();
        sm.DefaultAlgorithmSuite =
System.ServiceModel.Security.SecurityAlgorithmSuite.Basic128;
        sm.SetKeyDerivation(false);
        sm.SecurityHeaderLayout = SecurityHeaderLayout.Lax;
        sm.IncludeTimestamp = true;
        sm.KeyEntropyMode = SecurityKeyEntropyMode.CombinedEntropy;
        sm.MessageProtectionOrder = MessageProtectionOrder.SignBeforeEncrypt;
        sm.MessageSecurityVersion =

MessageSecurityVersion.WSSecurity11WSTrustFebruary2005WSSecureConversationFebru
ary2005
        WSSecurityPolicy11BasicSecurityProfile10;
        sm.RequireSignatureConfirmation = true;
        sm.LocalClientSettings.CacheCookies = true;
        sm.LocalClientSettings.DetectReplays = true;
        sm.LocalClientSettings.ReplayCacheSize = 900000;
        sm.LocalClientSettings.MaxClockSkew = new TimeSpan(00, 05, 00);
        sm.LocalClientSettings.MaxCookieCachingTime = TimeSpan.MaxValue;
        sm.LocalClientSettings.ReplayWindow = new TimeSpan(00, 05, 00); ;
       sm.LocalClientSettings.SessionKeyRenewalInterval = new TimeSpan(10, 00,
00);
        sm.LocalClientSettings.SessionKeyRolloverInterval = new TimeSpan(00,
05, 00); ;
        sm.LocalClientSettings.ReconnectTransportOnFailure = true;
       sm.LocalClientSettings.TimestampValidityDuration = new TimeSpan(00, 05,
00); ;
        sm.LocalClientSettings.CookieRenewalThresholdPercentage = 60;
        sm.LocalServiceSettings.DetectReplays = false;
        sm.LocalServiceSettings.IssuedCookieLifetime = new TimeSpan(10, 00,
00);
        sm.LocalServiceSettings.MaxStatefulNegotiations = 128;
        sm.LocalServiceSettings.ReplayCacheSize = 900000;
        sm.LocalServiceSettings.MaxClockSkew = new TimeSpan(00, 05, 00);
        sm.LocalServiceSettings.NegotiationTimeout = new TimeSpan(00, 01, 00);
        sm.LocalServiceSettings.ReplayWindow = new TimeSpan(00, 05, 00);
        sm.LocalServiceSettings.InactivityTimeout = new TimeSpan(00, 02, 00);
        sm.LocalServiceSettings.SessionKeyRenewalInterval = new TimeSpan(15,
00, 00);
        sm.LocalServiceSettings.SessionKeyRolloverInterval = new TimeSpan(00,
05, 00);
        sm.LocalServiceSettings.ReconnectTransportOnFailure = true;
        sm.LocalServiceSettings.MaxPendingSessions = 128;
        sm.LocalServiceSettings.MaxCachedCookies = 1000;
        sm.LocalServiceSettings.TimestampValidityDuration = new TimeSpan(15,
00, 00);
        HttpTransportBindingElement hb = new HttpTransportBindingElement();
        hb.ManualAddressing = false;
        hb.MaxBufferPoolSize = 524288;
        hb.MaxReceivedMessageSize = 65536;
        hb.AllowCookies = false;
        hb.AuthenticationScheme = System.Net.AuthenticationSchemes.Anonymous;
        hb.KeepAliveEnabled = true;
        hb.MaxBufferSize = 65536;
        hb.ProxyAuthenticationScheme =
System.Net.AuthenticationSchemes.Anonymous;
        hb.Realm = "";
        hb.TransferMode = System.ServiceModel.TransferMode.Buffered;
        hb.UnsafeConnectionNtlmAuthentication = false;
        hb.UseDefaultWebProxy = true;
        TextMessageEncodingBindingElement tb1 = new
TextMessageEncodingBindingElement();
        tb1.MaxReadPoolSize = 64;
        tb1.MaxWritePoolSize = 16;
        tb1.MessageVersion =
System.ServiceModel.Channels.MessageVersion.Soap12;
        tb1.WriteEncoding = System.Text.Encoding.UTF8;
        CustomBinding binding1 = new CustomBinding(sm);
        binding1.Elements.Add(tb1);
        binding1.Elements.Add(hb);
        ServiceEndpoint ep = selfHost.AddServiceEndpoint(typeof(ICalculator),
binding1,
          "CalculatorService");

        EndpointAddress myEndpointAdd = new EndpointAddress(
        new Uri(uri),
        EndpointIdentity.CreateDnsIdentity("WSMCert3"));
        ep.Address = myEndpointAdd;

        // Step 4 of the hosting procedure: Enable metadata exchange.
        ServiceMetadataBehavior smb = new ServiceMetadataBehavior();
        smb.HttpGetEnabled = true;
        selfHost.Description.Behaviors.Add(smb);

selfHost.Credentials.ServiceCertificate.SetCertificate(StoreLocation.CurrentUse
r,
           StoreName.My,
        X509FindType.FindBySubjectName, "WSMCert3");

selfHost.Credentials.ClientCertificate.Authentication.CertificateValidationMode
=
            X509CertificateValidationMode.PeerOrChainTrust;

selfHost.Credentials.UserNameAuthentication.UserNamePasswordValidationMode =
            UserNamePasswordValidationMode.Custom;
        CustomUserNameValidator cu = new CustomUserNameValidator();
```

```
selfHost.Credentials.UserNameAuthentication.CustomUserNamePasswordValidator =
cu;
        using (ServiceHost host = new ServiceHost(typeof(CalculatorService)))
        {
            System.ServiceModel.Description.ServiceDescription svcDesc =
selfHost.Description;
            ServiceDebugBehavior svcDebug =
svcDesc.Behaviors.Find<ServiceDebugBehavior>();
            svcDebug.IncludeExceptionDetailInFaults = true;
        }

        // Step 5 of the hosting procedure: Start (and then stop) the service.
        selfHost.Open();
        Console.WriteLine("The Calculator service is ready.");
        Console.WriteLine("Press <ENTER> to terminate service.");
        Console.WriteLine();
        Console.ReadLine();
        selfHost.Close();
    }
    catch (CommunicationException ce)
    {
         Console.WriteLine("An exception occurred: {0}", ce.Message);
         selfHost.Abort();
     }
}
```

5.4 Username Token Over SSL

This section describes how to implement username token over SSL in the following interoperability scenario:

- Configuring an OWSM 12c Web Service and a Microsoft WCF/.NET 3.5 Client

5.4.1 Configuring an OWSM 12*c* Web Service and a Microsoft WCF/.NET 3.5 Client

The following instructions tell how to configure an OWSM 12*c* web service and a Microsoft WCF/.NET 3.5 client to implement username token over SSL, both with and without secure conversation enabled.

To Configure the OWSM 12c Web Service:

1. Configure the server for SSL.

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

2. Create an OWSM web service.

3. Select the policy to use based on whether or not you want to enable secure conversation.

   If you do not want to enable secure conversation, use either of the following policies:

   `oracle/wss_username_token_over_ssl_service_policy`

   `oracle/wss_saml_or_username_token_over_ssl_service_policy`

   To enable secure conversation, use the following policy:

   `oracle/wss_username_token_over_ssl_wssc_service_policy`

> **Note:** In the case of secure conversation enabled, you will have to configure the `app.config` file somewhat differently, as described in Table 5–12:

For more information, see "Cloning a Web Service Policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

4. Edit the policy settings, as follows:

   a. Disable the Creation Time Required configuration setting.

   b. Disable the Nonce Required configuration setting.

   c. Leave the default configuration set for all other configuration settings.

5. Attach the policy.

   "Attaching Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

To Configure the Microsoft WCF/.NET 3.5 Client:

1. Generate a .NET client using the WSDL of the web service.

   For more information, see "How to: Create a Windows Communication Foundation Client" at `http://msdn.microsoft.com/en-us/library/ms733133(v=vs.90).aspx`.

2. In the Solution Explorer of the client project, add a reference by right-clicking on references, selecting Add reference, and browsing to `C:\Windows\Microsoft.NET\framework\v3.0\Windows Communication Foundation\System.Runtime.Serialization.dll`.

3. Edit the `app.config`, as shown in Example 5–5.

4. Edit the `app.config` file as needed to enable to enable secure conversation or not.

   If you do not want to enable secure conversation, edit the `app.config` as shown in regular typeface in Example 5–3:

   - Set the `authenticationMode` property of the `<security>` element to `UserNameOverTransport`.
   - Do not configure the properties of the `secureConversationBootstrap` element.

   To *enable* secure conversation, edit the `app.config` as shown the comments in bold italics in Example 5–3:

   - Set the `authenticationMode` property of the `<security>` element to `SecureConversation`.
   - Configure the `secureConversationBootstrap` element with additional properties, as shown in the example.

5. Compile the project.

6. Open a command prompt and navigate to the project's Debug folder.

7. Type `<client_project_name>.exe` and press **Enter**.

   See the following *app.config file* sample:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
```

```
  <system.serviceModel>
    <bindings>
      <customBinding>
        <binding name="BPELProcess1Binding">
          <!-- To enable secrure conversation, you must use
          authenticationMode="SecureConversation"
          instead of the value for authenticationMode shown below, under
<security -->
          <security defaultAlgorithmSuite="Basic128"
            authenticationMode="UserNameOverTransport"
            requireDerivedKeys="false" securityHeaderLayout="Lax"
includeTimestamp="true"
            keyEntropyMode="CombinedEntropy"
messageProtectionOrder="SignBeforeEncrypt"

messageSecurityVersion="WSSecurity11WSTrustFebruary2005WSSecureConversation
            February2005WSSecurityPolicy11BasicSecurityProfile10"
            requireSignatureConfirmation="true">
            <localClientSettings cacheCookies="true" detectReplays="false"
              replayCacheSize="900000" maxClockSkew="00:05:00"
              maxCookieCachingTime="Infinite"
              replayWindow="00:05:00" sessionKeyRenewalInterval="10:00:00"
              sessionKeyRolloverInterval="00:05:00"
reconnectTransportOnFailure="true"
              timestampValidityDuration="00:05:00"
              cookieRenewalThresholdPercentage="60"/>
            <localServiceSettings detectReplays="true"
issuedCookieLifetime="10:00:00"
              maxStatefulNegotiations="128" replayCacheSize="900000"
              maxClockSkew="00:05:00"
              negotiationTimeout="00:01:00" replayWindow="00:05:00"
              inactivityTimeout="00:02:00"
              sessionKeyRenewalInterval="15:00:00"
              sessionKeyRolloverInterval="00:05:00"
              reconnectTransportOnFailure="true" maxPendingSessions="128"
              maxCachedCookies="1000" timestampValidityDuration="00:05:00" />
            <secureConversationBootstrap />
           <!-- To enable secure conversation, add the following properties to
            the <secureConversationBootstrap> element:
            <secureConversationBootstrap
              authenticationMode="UserNameOverTransport"
              requireDerivedKeys="false"
              securityHeaderLayout="Lax"
              includeTimestamp="true"
              keyEntropyMode="CombinedEntropy"
              messageProtectionOrder="SignBeforeEncrypt"

messageSecurityVersion="WSSecurity11WSTrustFebruary2005WSSecureConversationFebr
uary2005WSSecurityPolicy11BasicSecurityProfile10"
              requireSignatureConfirmation="true"/> -->
          </security>
            <textMessageEncoding
              maxReadPoolSize="64"
              maxWritePoolSize="16"
              messageVersion="Soap11"
              writeEncoding="utf-8">
                <readerQuotas
                  maxDepth="32"
                  maxStringContentLength="8192"
                  maxArrayLength="16384"
```

```
                    maxBytesPerRead="4096"
                    maxNameTableCharCount="16384" />
                </textMessageEncoding>
                <httpsTransport
                  manualAddressing="false"
                  maxBufferPoolSize="524288"
                  maxReceivedMessageSize="65536"
                  allowCookies="false"
                  authenticationScheme="Anonymous"
                  bypassProxyOnLocal="false"
                  hostNameComparisonMode="StrongWildcard"
                  keepAliveEnabled="true"
                  maxBufferSize="65536"
                  proxyAuthenticationScheme="Anonymous"
                  realm=""
                  transferMode="Buffered"
                  unsafeConnectionNtlmAuthentication="false"
                  useDefaultWebProxy="true"  requireClientCertificate="false"/>
                </binding>
              </customBinding>
            </bindings>
          <client>
              <endpoint
                address="
https://host:port/soa-infra/services/default/IO_NET6/bpelprocess1_client_ep"
                binding="customBinding"
bindingConfiguration="BPELProcess1Binding"
                contract="BPELProcess1" name="BPELProcess1_pt" />
          </client>
    </system.serviceModel>
</configuration>
```

5.5 Mutual Authentication with Message Protection (WS-Security 1.1)

This section describes how to implement mutual authentication with message protection that conform to the WS-Security 1.1 standards in the following interoperability scenarios:

- Configuring an OWSM 12c Web Service and a Microsoft WCF/.NET 3.5 Client
- Configuring a Microsoft WCF/.NET 3.5 Web Service and an OWSM 12c Client

Before configuring the web service and client in either of the above scenarios, follow the instructions in "Configuration Prerequisites".

5.5.1 Configuration Prerequisites

The following procedure describes how to perform prerequisite configuration tasks for implementing mutual authentication with message protection that conform to the WS-Security 1.1 standards.

To Configure Prerequisites for Interoperability:

1. Export the X.509 certificate file from the keystore on the service side to a `.cer` file (for example, `alice.cer`) using the following command:

   ```
   keytool -export -alias alice -file C:\alice.cer -keystore default-keystore.jks
   ```

2. Import the certificate file (exported previously) to the keystore on the client server using Microsoft Management Console (mmc).

a. Open a command prompt.

b. Type **mmc** and press ENTER.

c. Select **File > Add/Remove snap-in**.

d. Select **Add and Choose Certificates**.

   Note: To view certificates in the local machine store, you must be in the Administrator role.

e. Select **Add**.

f. Select **My user account and finish**.

g. Click **OK**.

h. Expand **Console Root > Certificates -Current user > Personal > Certificates**.

i. Right-click on **Certificates** and select **All tasks > Import** to launch Certificate import Wizard.

j. Click **Next**, select **Browse**, and navigate to the `.cer` file that was exported previously.

k. Click **Next** and accept defaults and finish the wizard.

For more information, see "How to: View Certificates with the MMC Snap-in" at `http://msdn.microsoft.com/en-us/library/ms788967.aspx`.

5.5.2 Configuring an OWSM 12*c* Web Service and a Microsoft WCF/.NET 3.5 Client

The following instructions tell how to configure an OWSM 12*c* web service and a Microsoft WCF/.NET 3.5 client to implement mutual authentication with message protection that conform to the WS-Security 1.1 standards.

To Configure the OWSM 12c Web Service:

1. Create a SOA composite and deploy it.

2. Using Fusion Middleware Control, attach the following policy to the web service:

   `oracle/wss11_x509_token_with_message_protection_service_policy`.

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

To Configure the Microsoft WCF/.NET 3.5 Client:

1. Use the Microsoft SvcUtil utility to create a client proxy (see Table 5–7, "Client Program") and configuration file from the deployed web service.

   For more information, see `http://msdn.microsoft.com/en-us/library/aa347733%28v=vs.90%29.aspx`

2. In the Solution Explorer of the client project, add a reference by right-clicking on references, selecting **Add reference**, and browsing to `C:\Windows\Microsoft.NET\framework\v3.0\Windows Communication Foundation\System.Runtime.Serialization.dll`.

3. Create an `app.config` configuration file, including the following steps.

   An example of the complete file is shown in Example 5–6, "app.config File". The steps listed below are called out in **bold type** in the example.

   a. Define behaviors with credentials.

   b. Create a custom binding.

   c. Diable the message replay detection.
   d. Modify endpoint behavior.

4. Compile the project.
5. Open a command prompt and navigate to the project's Debug folder.
6. Enter `<client_project_name>.exe` and press **Enter**.

   See the following *app.config* file sample:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.serviceModel>


  <!-- 1. Define behaviors with credentials
------------------------------------------- -->
    <behaviors>
      <endpointBehaviors>
        <behavior name="secureBehaviour">
          <clientCredentials>
            <serviceCertificate>
              <defaultCertificate findValue="<certificate_cn>"
                                  storeLocation="CurrentUser"
                                  storeName="My"
                                  x509FindType="FindBySubjectName"/>
            </serviceCertificate>
          </clientCredentials>
        </behavior>
      </endpointBehaviors>
    </behaviors>
  <!--
--------------------------------------------------------------------------------
-->

     <bindings>
       <customBinding>
         <binding name="BPELProcess1Binding">


  <!-- --- 2. Create a custom binding
------------------------------------------------- -->
           <security defaultAlgorithmSuite="Basic128"
authenticationMode="MutualCertificate"
  <!--
--------------------------------------------------------------------------------
-->

              requireDerivedKeys="false" securityHeaderLayout="Lax"
includeTimestamp="true"
              keyEntropyMode="CombinedEntropy"
messageProtectionOrder="SignBeforeEncrypt"

messageSecurityVersion="WSSecurity11WSTrustFebruary2005WSSecureConversation
              February2005WSSecurityPolicy11BasicSecurityProfile10"
              requireSignatureConfirmation="true">


  <!-- --- 3. Disable the message replay detection
-----------------------------------  -->
```

```
           <localClientSettings cacheCookies="true" detectReplays="false"
                    replayCacheSize="900000" maxClockSkew="00:05:00"
                    maxCookieCachingTime="Infinite"
  <!--
-------------------------------------------------------------------------------
-->

                    replayWindow="00:05:00"
sessionKeyRenewalInterval="10:00:00"
                    sessionKeyRolloverInterval="00:05:00"
reconnectTransportOnFailure="true"
                    timestampValidityDuration="00:05:00"
cookieRenewalThresholdPercentage="60" />
                <localServiceSettings detectReplays="true"
                    issuedCookieLifetime="10:00:00"
                    maxStatefulNegotiations="128"
                    replayCacheSize="900000" maxClockSkew="00:05:00"
                    negotiationTimeout="00:01:00" replayWindow="00:05:00"
                    inactivityTimeout="00:02:00"
                    sessionKeyRenewalInterval="15:00:00"
sessionKeyRolloverInterval="00:05:00"
                    reconnectTransportOnFailure="true" maxPendingSessions="128"
                    maxCachedCookies="1000"
timestampValidityDuration="00:05:00" />
                <secureConversationBootstrap />
              </security>
              <textMessageEncoding maxReadPoolSize="64" maxWritePoolSize="16"
                  messageVersion="Soap11" writeEncoding="utf-8">
                <readerQuotas maxDepth="32" maxStringContentLength="8192"
maxArrayLength="16384"
                    maxBytesPerRead="4096" maxNameTableCharCount="16384" />
              </textMessageEncoding>
              <httpTransport manualAddressing="false"
maxBufferPoolSize="524288"
                  maxReceivedMessageSize="65536" allowCookies="false"
                  authenticationScheme="Anonymous"
                  bypassProxyOnLocal="false"
hostNameComparisonMode="StrongWildcard"
                  keepAliveEnabled="true" maxBufferSize="65536"
                  proxyAuthenticationScheme="Anonymous"
                  realm="" transferMode="Buffered"
unsafeConnectionNtlmAuthentication="false"
                  useDefaultWebProxy="true" />
            </binding>
          </customBinding>

        </bindings>
        <client>


  <!--  - 4. Modify endpoint behavior
------------------------------------------------- -->
          <endpoint address="http://<server>:<port>//MyWebService1SoapHttpPort"
                binding="customBinding"
bindingConfiguration="MyWebService1SoapHttp"
                contract="MyWebService1"
                name="MyWebService1SoapHttpPort"
                behaviorConfiguration="secureBehaviour" >
            <identity>
              <dns value="<certificate_cn>"/>
            </identity>
          </endpoint>
  <!--
-------------------------------------------------------------------------------
-->

        </client>
    </system.serviceModel>
</configuration>
```

See the following *Client Program* sample:

```
 namespace IO_NET10_client
{
    class Program
    {
        static void Main(string[] args)
        {

            BPELProcess1Client client = new BPELProcess1Client();

            client.ClientCredentials.ClientCertificate.SetCertificate(
                    StoreLocation.CurrentUser,
                    StoreName.My,
                    X509FindType.FindBySubjectName, "WSMCert3");

             client.ClientCredentials.ServiceCertificate.SetDefaultCertificate(
                       StoreLocation.CurrentUser,
                       StoreName.My,
                    X509FindType.FindBySubjectName, "Alice");

            process proc = new process();
            proc.input = "Test wss11_x509_token_with_message_protection_policy
- ";
            Console.WriteLine(proc.input);
            processResponse response = client.process(proc);

            Console.WriteLine(response.result.ToString());
            Console.WriteLine("Press <ENTER> to terminate Client.");
            Console.ReadLine();
          }
    }
}
```

5.5.3 Configuring a Microsoft WCF/.NET 3.5 Web Service and an OWSM 12*c* Client

The following instructions tell how to configure a Microsoft WCF/.NET 3.5 web service and an OWSM 12*c* client to implement mutual authentication with message protection that conform to the WS-Security 1.1 standards.

To Configure the Microsoft WCF/.NET 3.5 Web Service:

1. Create a .NET web service.

   For an example, see Example 5–4, "Example of .NET Web Service".

   For more information, see "How to: Define a Windows Communication Foundation Service Contract" at `http://msdn.microsoft.com/en-us/library/ms731835%28v=vs.90%29.aspx`

2. Create a custom binding for the web service using the SymmetricSecurityBindingElement.

The following is a sample of the SymmetricSecurityBindingElement object:

```
SymmetricSecurityBindingElement sm =
(SymmetricSecurityBindingElement)SecurityBindingElement.CreateMutualCertificate
BindingElement();

sm.DefaultAlgorithmSuite =
System.ServiceModel.Security.SecurityAlgorithmSuite.Basic128;sm.SetKeyDerivati
on(false);
sm.SecurityHeaderLayout = SecurityHeaderLayout.Lax;sm.IncludeTimestamp =
true;
sm.KeyEntropyMode = SecurityKeyEntropyMode.CombinedEntropy;
sm.MessageProtectionOrder =
MessageProtectionOrder.SignBeforeEncrypt;sm.MessageSecurityVersion =
MessageSecurityVersion.WSSecurity11WSTrustFebruary2005WSSecureConversation
February2005WSSecurityPolicy11BasicSecurityProfile10;
sm.RequireSignatureConfirmation =
true;
```

For more information, see "How to: Create a Custom Binding Using the SecurityBindingElement" at `http://msdn.microsoft.com/en-us/library/ms730305(v=vs.90).aspx`.

3. Deploy the application.

To Configure the OWSM 12c Client:

1. Using JDeveloper, create a SOA composite that consumes the .NET web service.

   For more information, see *Developer's Guide for SOA Suite*.

2. In JDeveloper, create a partner link using the WSDL of the .NET service and add the import as follows:

   ```
   <wsdl:import namespace="<namespace>" location="<WSDL location>"/>
   ```

3. In Fusion Middleware Control, attach the following policy to the web service client:

   `oracle/wss11_x509_token_with_message_protection_client_policy`.

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

4. Provide configurations for the `keystore.recipient.alias`.

   You can specify this information when attaching the policy, by overriding the policy configuration.

   Ensure that you configure the `keystore.recipient.alias` as the alias of the certificate imported in step 4 (`wsmcert3`).

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

5. Invoke the web service method from the client.

5.6 Kerberos with Message Protection

This section describes how to implement Kerberos with message protection in the following interoperability scenario:

- Configuring an OWSM 12c Web Service and a Microsoft WCF/.NET 3.5 Client

5.6.1 Performing Prerequisite Tasks for Interoperability

To Configure Prerequisites for Interoperability:

1. Configure the Key Distribution Center (KDC) and Active Directory (AD).

   For more information, see "To Configure Windows Active Directory and Domain Controller" (the domain controller can serve as KDC) at `http://download.oracle.com/docs/cd/E19316-01/820-3746/gisdn/index.html`.

2. Set up the Kerberos configuration file `krb5.conf` in `c:\winnt` as shown in Example 5–8, "Kerberos Configuration File".

   See the following *Kerberos Configuration File* sample:

```
[logging]
default = c:\log\krb5libs.log
kdc = c:\log\krb5kdc.log
admin_server = c:\log\kadmind.log
[libdefaults]
default_realm = MYCOMPANY.LOCAL
dns_lookup_realm = false
dns_lookup_kdc = false
default_tkt_enctypes = rc4-hmac
default_tgs_enctypes = rc4-hmac
permitted_enctypes = rc4-hmac
kdc = hostname
[realms]
MYCOMPANY.LOCAL =
{ kdc = host:port  admin_server = host:port
  default_domain = <domainname>
}
 [domain_realm]
.<domainname> = MYCOMPANY.LOCAL
 <domainname> = MYCOMPANY.LOCAL
[appdefaults]
pam =
{   debug = false  ticket_lifetime = 36000  renew_lifetime = 36000  forwardable
=
 true  krb4_convert = false }
```

5.6.2 Configuring an OWSM 12*c* Web Service and a Microsoft WCF/.NET 3.5 Client

The following instructions tell how to configure an OWSM 12*c* web service and a Microsoft WCF/.NET 3.5 client to implement Kerberos with message protection.

To Configure the OWSM 12c Web Service:

1. Create and deploy a web service application.

2. Clone the following policy: `oracle/wss11_kerberos_token_with_message_protection_service_policy`.

   For more information, see "Cloning a Web Service Policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

3. Edit the policy settings to set Algorithm Suite to `Basic128Rsa15`.

4. Attach the policy to the web service.

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

To Configure the Microsoft WCF/.NET 3.5 Client:

1. Create a user in AD to represent the host where the web service is hosted. By default the user account is created with RC4-HMAC encryption. For example, foobar with user name is `HTTP/foobar`.

2. Use the following ktpass command to create a keytab file on the Windows AD machine where the KDC is running:

   ```
   ktpass -princ HTTP/foobar@MYCOMPANY.LOCAL -pass Oracle123 -mapuser
   foobar -out foobar.keytab -ptype KRB5_NT_PRINCIPAL -kvno 4
   ```

   where `HTTP/foobar` is the SPN, mapped to a user "foobar". Do not set "/desonly or cyrpto as "des-cbc-crc". MYCOMPANY.LOCAL is the default Realm for the KDC and is available in the krb5.ini file. The pass password must match the password created during the user creation.

   Use FTP binary mode to move the generated keytab file to the machine where the SOA Composite web service is hosted.

3. Use the following `setSpn` command to map the service principal to the user:

   ```
   setSpn -A HTTP/foobar@MYCOMPANY.LOCAL foobar
   ```

   ```
   setSpn -L foobar
   ```

   Only one SPN must be mapped to the user. If there are multiple SPNs mapped to the user, remove them using the command `setSpn -D <spname> <username>`.

4. Use the Microsoft svcutil utility to create a client proxy and configuration file from the deployed web service.

   Add the files generatedProxy.cs and app.config by right clicking the application (in the Windows Explorer) and selecting **Add Existing Item**.

   In the endpoint element of the app.config, add an "identity" element with service principal name as "HTTP/foobar@MYCOMPANY.LOCAL" (the same value used for creating keytab).

   ```
   <client>
           <endpoint address="http://host:port/HelloServicePort"
               binding="customBinding"
   bindingConfiguration="NewHelloSoap12HttpPortBinding"
               contract="NewHello" name="HelloServicePort">
           <identity>
             <servicePrincipalName value ="HTTP/foobar@MYCOMPANY.LOCAL"/>
           </identity>
           </endpoint>

         </client>
   ```

   A sample binding is provided in Example 5–9, "Custom Binding".

   For more information, see `http://msdn.microsoft.com/en-us/library/aa347733%28v=vs.90%29.aspx`.

5. Run the client program.

   See the following *Custom Binding* sample:

   ```
   <customBinding>
     <binding name="NewHelloSoap12HttpPortBinding">
         <!--Added by User: Begin-->
         <security defaultAlgorithmSuite="Basic128"
           authenticationMode="Kerberos"
           requireDerivedKeys="false" securityHeaderLayout="Lax"
           includeTimestamp="true"
           keyEntropyMode="CombinedEntropy"
   ```

```
              messageProtectionOrder="SignBeforeEncrypt"
              messageSecurityVersion="WSSecurity11WSTrustFebruary2005
              WSSecureConversationFebruary2005WSSecurityPolicy11BasicSecurity
                Profile10"
              requireSignatureConfirmation="true">
            <localClientSettings cacheCookies="true" detectReplays="true"
                replayCacheSize="900000" maxClockSkew="00:05:00"
                maxCookieCachingTime="Infinite"
                replayWindow="00:05:00"
                sessionKeyRenewalInterval="10:00:00"
                sessionKeyRolloverInterval="00:05:00"
                reconnectTransportOnFailure="true"
                timestampValidityDuration="00:05:00"
                cookieRenewalThresholdPercentage="60" />
                      <localServiceSettings detectReplays="true"
                issuedCookieLifetime="10:00:00"
                maxStatefulNegotiations="128" replayCacheSize="900000"
                maxClockSkew="00:05:00"
                negotiationTimeout="00:01:00" replayWindow="00:05:00"
                inactivityTimeout="00:02:00"
                sessionKeyRenewalInterval="15:00:00"
                sessionKeyRolloverInterval="00:05:00"
                reconnectTransportOnFailure="true"
                maxPendingSessions="128"
                maxCachedCookies="1000"
                timestampValidityDuration="00:05:00" />
                        <secureConversationBootstrap />
                      </security>
                    <!--Added by User: End-->
                      <textMessageEncoding maxReadPoolSize="64"
                         maxWritePoolSize="16"
                         messageVersion="Soap12" writeEncoding="utf-8">
                      <readerQuotas maxDepth="32" maxStringContentLength="8192"
                         maxArrayLength="16384"
                         maxBytesPerRead="4096" maxNameTableCharCount="16384" />
                      </textMessageEncoding>
                    <!--Added by User: Begin-->
                    <httpTransport manualAddressing="false"
                         maxBufferPoolSize="524288"
                         maxReceivedMessageSize="65536" allowCookies="false"
                         authenticationScheme="Anonymous"
                         bypassProxyOnLocal="false"
                         hostNameComparisonMode="StrongWildcard"
                         keepAliveEnabled="true" maxBufferSize="65536"
                         proxyAuthenticationScheme="Anonymous"
                         realm="" transferMode="Buffered"
                         unsafeConnectionNtlmAuthentication="false"
                         useDefaultWebProxy="true" />
                      <!--Added by User: End-->
                 </binding>
      </customBinding>
```

5.7 Kerberos with Message Protection Using Derived Keys

This section describes how to implement Kerberos with message protection using derived keys in the following interoperability scenario:

- Configuring an OWSM 12c Web Service and a Microsoft WCF/.NET 3.5 Client

Before configuring the web service and client in the above scenario, follow the instructions in Section 5.7.1, "Configuration Prerequisites."

5.7.1 Configuration Prerequisites

The following procedure describes how to perform prerequisite configuration tasks for implementing Kerberos with message protection using derived keys.

To Configure Prerequisites for Interoperability:

1. Configure the Key Distribution Center (KDC) and Active Directory (AD).

   For more information, see the following topics:

   - "To Configure Windows Active Directory and Domain Controller" (the domain controller can serve as KDC) at `http://download.oracle.com/docs/cd/E19316-01/820-3746/gisdn/index.html`
   - "Configuring Kerberos Tokens" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*

2. Set up the Kerberos configuration file `krb5.conf` in `c:\winnt` as shown in the example "Kerberos Configuration File".

```
[logging]
default = c:\log\krb5libs.log
kdc = c:\log\krb5kdc.log
admin_server = c:\log\kadmind.log
[libdefaults]
default_realm = MYCOMPANY.LOCAL
dns_lookup_realm = false
dns_lookup_kdc = false
default_tkt_enctypes = rc4-hmac
default_tgs_enctypes = rc4-hmac
permitted_enctypes = rc4-hmac
kdc = hostname
[realms]
MYCOMPANY.LOCAL =
{ kdc = host:port  admin_server = host:port
  default_domain = <domainname>
}
 [domain_realm]
.<domainname> = MYCOMPANY.LOCAL
 <domainname> = MYCOMPANY.LOCAL
[appdefaults]
pam =
{   debug = false  ticket_lifetime = 36000  renew_lifetime = 36000  forwardable
=
 true  krb4_convert = false }
```

5.7.2 Configuring an OWSM 12*c* Web Service and a Microsoft WCF/.NET 3.5 Client

The following instructions tell how to configure an OWSM 12*c* web service and a Microsoft WCF/.NET 3.5 client to implement Kerberos with message protection.

To Configure the OWSM 12c Web Service:

1. Create and deploy a web service application.
2. Clone the following policy: `wss11_kerberos_token_with_message_protection_basic128_service_policy.`

For more information, see "Cloning a Web Service Policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

3. Edit the policy settings to enable the Derived Keys option.
4. Attach the policy to the web service.

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

To Configure the Microsoft WCF/.NET 3.5 Client:

1. Create a user in AD to represent the host where the web service is hosted. By default the user account is created with RC4-HMAC encryption. For example, foobar with user name as "HTTP/foobar".
2. Use the following ktpass command to create a keytab file on the Windows AD machine where the KDC is running:

   ```
   ktpass -princ HTTP/foobar@MYCOMPANY.LOCAL -pass Oracle123 -mapuser
   foobar -out foobar.keytab -ptype KRB5_NT_PRINCIPAL -kvno 4
   ```

   where HTTP/foobar is the SPN, mapped to a user "foobar". Do not set "/desonly or cyrpto as "des-cbc-crc". MYCOMPANY.LOCAL is the default Realm for the KDC and is available in the `krb5.ini` file. The pass password must match the password created during the user creation.

   Use FTP binary mode to move the generated keytab file to the machine where the SOA Composite web service is hosted.

3. Use the following `setSpn` command to map the service principal to the user:

   ```
   setSpn -A HTTP/foobar@MYCOMPANY.LOCAL foobar
   ```

   ```
   setSpn -L foobar
   ```

   Only one SPN must be mapped to the user. If there are multiple SPNs mapped to the user, remove them using the command `setSpn -D <spname> <username>`.

4. Use the Microsoft SvcUtil utility to create a client proxy and configuration file from the deployed web service.

   Add the files generatedProxy.cs and app.config by right clicking the application (in the Windows Explorer) and selecting **Add Existing Item**.

   In the endpoint element of the app.config, add an "identity" element with service principal name as "HTTP/foobar@MYCOMPANY.LOCAL" (the same value used for creating keytab).

   ```
   <client>
           <endpoint address="http://host:port/HelloServicePort"
               binding="customBinding"
   bindingConfiguration="NewHelloSoap12HttpPortBinding"
               contract="NewHello" name="HelloServicePort">
           <identity>
             <servicePrincipalName value ="HTTP/foobar@MYCOMPANY.LOCAL"/>
           </identity>
           </endpoint>

         </client>
   ```

   A sample binding is provided in Example 5–11, "Custom Binding".

5. Run the client program.

   See the following *Custom Binding* sample:

```
<customBinding>
  <binding name="NewHelloSoap12HttpPortBinding">
    <!--Added by User: Begin-->
    <security defaultAlgorithmSuite="Basic128"
      authenticationMode="Kerberos"
      requireDerivedKeys="true" securityHeaderLayout="Lax"
      includeTimestamp="true"
      keyEntropyMode="CombinedEntropy"
      messageProtectionOrder="SignBeforeEncrypt"
      messageSecurityVersion="WSSecurity11WSTrustFebruary2005
      WSSecureConversationFebruary2005WSSecurityPolicy11BasicSecurity
      Profile10"
      requireSignatureConfirmation="true">
    <localClientSettings cacheCookies="true" detectReplays="true"
        replayCacheSize="900000" maxClockSkew="00:05:00"
        maxCookieCachingTime="Infinite"
        replayWindow="00:05:00"
        sessionKeyRenewalInterval="10:00:00"
        sessionKeyRolloverInterval="00:05:00"
        reconnectTransportOnFailure="true"
        timestampValidityDuration="00:05:00"
        cookieRenewalThresholdPercentage="60" />
      <localServiceSettings detectReplays="true"
        issuedCookieLifetime="10:00:00"
        maxStatefulNegotiations="128" replayCacheSize="900000"
        maxClockSkew="00:05:00"
        negotiationTimeout="00:01:00" replayWindow="00:05:00"
        inactivityTimeout="00:02:00"
        sessionKeyRenewalInterval="15:00:00"
        sessionKeyRolloverInterval="00:05:00"
        reconnectTransportOnFailure="true"
        maxPendingSessions="128"
        maxCachedCookies="1000"
        timestampValidityDuration="00:05:00" />
      <secureConversationBootstrap />
    </security>
  <!--Added by User: End-->
      <textMessageEncoding maxReadPoolSize="64"
        maxWritePoolSize="16"
        messageVersion="Soap12" writeEncoding="utf-8">
          <readerQuotas maxDepth="32" maxStringContentLength="8192"
            maxArrayLength="16384"
            maxBytesPerRead="4096" maxNameTableCharCount="16384" />
      </textMessageEncoding>
          <!--Added by User: Begin-->
      <httpTransport manualAddressing="false"
        maxBufferPoolSize="524288"
        maxReceivedMessageSize="65536" allowCookies="false"
        authenticationScheme="Anonymous"
        bypassProxyOnLocal="false"
        hostNameComparisonMode="StrongWildcard"
        keepAliveEnabled="true" maxBufferSize="65536"
        proxyAuthenticationScheme="Anonymous"
        realm="" transferMode="Buffered"
        unsafeConnectionNtlmAuthentication="false"
        useDefaultWebProxy="true" />
      <!--Added by User: End-->
  </binding>
</customBinding>
```

5.8 Kerberos with SPNEGO Negotiation

This section describes how to implement Kerberos with SPNEGO negotiation in the following interoperability scenario:

- Configuring an OWSM 12c Web Service and a Microsoft WCF/.NET 3.5 Client

5.8.1 Configuring an OWSM 12*c* Web Service and a Microsoft WCF/.NET 3.5 Client

The following instructions tell how to configure an OWSM 12*c* web service and a Microsoft WCF/.NET 3.5 client to implement Kerberos with SPNEGO negotiation.

To Configure the OWSM 12c Web Service:

1. Create and deploy a web service application.
2. Create a policy that uses the `http_spnego_token_service_template` assertion template.

   For more information, see Configuring Kerberos With SPNEGO Negotiation" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.
3. Attach the policy to the web service.

To Configure the Microsoft WCF/.NET 3.5 Client:

1. Use the Microsoft SvcUtil utility to create a client proxy and configuration file from the deployed web service.

   For more information, see `http://msdn.microsoft.com/en-us/library/aa347733%28v=vs.90%29.aspx`.
2. Add the files generatedProxy.cs and app.config by right clicking the application (in the Windows Explorer) and selecting **Add Existing Item**.
3. Edit the `app.config` file as shown in Example 5–12, "app.config File".

   In this listing, note that the values of the contract and name attributes of the endpoint element are obtained from the generatedProxy.cs file.
4. Compile the client.
5. After attaching the OWSM policy to the deployed web service, run the client.

   See the following *app.config* file sample:

```
<configuration>
  <system.serviceModel>
    <bindings>
      <basicHttpBinding>
        <binding name="BPELProcessBinding">
          <security mode= "TransportCredentialOnly">
            <transport clientCredentialType="Windows"/>
          </security>
        </binding>
      </basicHttpBinding>
    </bindings>
    <client>
      <endpoint
          address="http://host:port/soa-infra/services/default/SOAProxy/bpelpro
cess_client_ep"
          binding="basicHttpBinding"
          bindingConfiguration="BPELProcessBinding"
          contract="BPELProcess" name="BPELProcess_pt"
        <identity>
```

```
          <servicePrincipalName value ="HTTP/host:port@MYCOMPANY.LOCAL" />
        </identity>
      </endpoint>
    </client>
  </system.serviceModel>
</configuration>
```

5.9 Kerberos with SPNEGO Negotiation and Credential Delegation

This section describes how to implement Kerberos with SPNEGO negotiation and credential delegation in the following interoperability scenario:

- Configuring an OWSM 12c Web Service and a Microsoft WCF/.NET 3.5 Client

5.9.1 Configuring an OWSM 12*c* Web Service and a Microsoft WCF/.NET 3.5 Client

The following instructions tell how to configure an OWSM 12*c* web service and a Microsoft WCF/.NET 3.5 client to implement Kerberos with SPNEGO negotiation and credential delegation.

To Configure the OWSM 12c Web Service:

1. Create and deploy a web service application.
2. Create a policy that uses the `http_spnego_token_service_template` assertion template.

   For more information, see "Configuring Kerberos With SPNEGO Negotiation" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.
3. Attach the policy to the web service.
4. Set the value of the `credential.delegation` configuration setting to `true`.

   You can specify this information when attaching the policy, by overriding the policy configuration.

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

To Configure the Microsoft WCF/.NET 3.5 Client:

1. Use the Microsoft SvcUtil utility to create a client proxy and configuration file from the deployed web service.

   For more information, see `http://msdn.microsoft.com/en-us/library/aa347733%28v=vs.90%29.aspx`.
2. Add the files `generatedProxy.cs` and `app.config` by right clicking the application (in the Windows Explorer) and selecting **Add Existing Item**.
3. Edit the `app.config` file as shown in Example 5–13, "app.config File".

   In the example, note that the values of the contract and name attributes of the endpoint element are obtained from the `generatedProxy.cs` file.
4. Compile the client.
5. After attaching the OWSM policy to the deployed web service, run the client.

   See the following *app.config file* sample:

   ```
   <configuration>
     <system.serviceModel>
   ```

```
    <bindings>
      <basicHttpBinding>
        <binding name="BPELProcess1Binding">
          <security mode= "TransportCredentialOnly">
            <transport clientCredentialType="Windows"/>
          </security>
        </binding>
      </basicHttpBinding>
    </bindings>
    <client>
      <endpoint
          address="http://host:port/soa-infra/services/default/SOAProxy/bpelpro
cess1_client_ep"
          binding="basicHttpBinding"
          bindingConfiguration="BPELProcess1Binding"
          contract="BPELProcess1" name="BPELProcess1_pt"
          behaviorConfiguration="CredentialDelegation">
        <identity>
          <servicePrincipalName value ="HTTP/host:port@MYCOMPANY.LOCAL" />
        </identity>
      </endpoint>
    </client>
    <behaviors>
      <endpointBehaviors>
        <behavior name="CredentialDelegation">
          <clientCredentials>
            <windows allowedImpersonationLevel="Delegation"
              allowNtlm="false"/>
          </clientCredentials>
        </behavior>
      </endpointBehaviors>
    </behaviors>
  </system.serviceModel>
</configuration>
```

5.10 WCF/.NET 3.5 Client with Microsoft Active Directory Federation Services 2.0 (ADFS 2.0) STS

This section tells how to secure a WCF/.NET 3.5 client with Microsoft Active Directory Federation Services 2.0 (ADFS 2.0) secure token service (STS), using a   policy utilizing SAML bearer token over one-way SSL.

Note: The SAML sender vouches token is not supported in this use case.

The procedure described in this section assumes that you install and configure ADFS 2.0 on a Windows Server 2008 or Windows Server 2008 R2 system. This system is set up in the STS role.

The section includes the following topics:

- Section 5.10.1, "Install and Configure Active Directory Federation Services (ADFS) 2.0"
- Section 5.10.2, "Configure ADFS 2.0 STS As Trusted SAML Token Issuer"
- Section 5.10.3, "Configure Users in Oracle Internet Directory"

- Section 5.10.4, "Attach the Policy"
- Section 5.10.5, "Register the Web Service as a Relying Party in ADFS 2.0"
- Section 5.10.6, "Secure WCF/.NET 3.5 Client with ADFS 2.0"

5.10.1 Install and Configure Active Directory Federation Services (ADFS) 2.0

The following instructions tell how to install and configure ADFS 2.0:

To Install and Configure Active Directory Federation Services (ADFS) 2.0:

1. Install and configure Active Directory.

   For more information, see `http://technet.microsoft.com/en-us/windowsserver`.

2. Install ADFS 2.0 and configure it using the wizard.

   As you configure ADFS 2.0 using the wizard, on the Server Role page be sure to click **Federation server**.

   For more information, see `http://technet.microsoft.com/en-us/windowsserver/dd448613`.

   For download information, see `http://go.microsoft.com/fwlink/?linkid=151338`.

3. Create and configure a self-signed server authentication certificate in IIS and bind it to the default Web site using the Internet Information Services (IIS) Manager console. When done, enable SSL server authentication.

   The AD FS 2.0 Setup Wizard automatically installed the Web server (IIS) server role on the system.

   Creating a self-signed server authentication certificate is described generally in `http://technet.microsoft.com/en-us/library/cc771041%28v=ws.10%29.aspx`. The steps in this section provides use case-specific information.

   a. Open the Internet Information Services (IIS) Manager console.

   b. On the Start menu, click **All Programs**, point to Administrative Tools, and then click **Internet Information Services (IIS) Manager**.

   c. In the console tree, click the root node that contains the name of the system, and then, in the details pane, double-click the icon named **Server Certificates** in the IIS grouping.

   d. In the Actions pane, click **Create Self-Signed Certificate**.

   e. In the console tree, click **Default Web Site**.

   f. In the Actions pane, click **Bindings**.

   g. In the Site Bindings dialog box, click **Add**.

   h. In the Add Site Binding dialog box, select **https** in the Type drop-down list. Select the certificate you just created in the SSL certificate drop-down list, click **OK**, and then click **Close**.

   i. Close the Internet Information Services (IIS) Manager console. Enable SSL Server Authentication.

4. Configure the system as a standalone federation server.

   For more information, see `http://technet.microsoft.com/en-us/library/ee913579%28v=ws.10%29.aspx`.

5. Export the ADFS 2.0 token-signing certificate.

   For a self-signed certificate, select DER encoded binary X.509 (`.cer`).

   If the signing certificate is not self-signed, select Cryptographic Message Syntax Standard – PKCS 7 certificates (.p7b) and check **Include all the certificates in the certification path if possible**.

   For more information, see `http://technet.microsoft.com/en-us/library/dd378922%28v=ws.10%29.aspx#BKMK_4`.

6. Create users and include an email address. You later enable the STS to send the email address as the subject name id in the outgoing SAML assertions for the service.

   Follow these steps to add a sample user to Active Directory. Make sure to set the email address for each user.

   a. Log in to the system with domain administrator credentials.

   b. Click **Start**, click **Administrative Tools**, and then click **Active Directory Users and Computers**.

   c. In the console tree, right-click the **Users** folder. Click **New**, and then click **User**.

   d. On the New Object – User page, add the user, and then click **Next**.

   e. Provide a password, clear the **User must change password at next logon** check box, and then click **Next**.

   f. Click **Finish**.

   g. In the right-most pane of Active Directory Users and Computers, right-click the new user object, and then click **Properties**.

   h. On the General tab, in the E-mail box, type the email address of the user, and then click **OK**.

5.10.2 Configure ADFS 2.0 STS As Trusted SAML Token Issuer

The following instructions tell how to configure OWSM to trust the SAML assertions issued by an ADFS 2.0 STS.

To Configure ADFS 2.0 STS As Trusted SAML Token Issuer:

1. Get the STS signing certificates you exported in Table 5–28, " Install and Configure Active Directory Federation Services (ADFS) 2.0".

   For a `.p7b` file for a certificate chain, open the file in IE and copy each certificate in the chain in a `.cer` file.

2. Import the certificates into the location of the default keystore using keytool.

   ```
   keytool -importcert -file <sts-signing-certs-file> -trustcacerts -alias
   <alias> -keystore default-keystore.jks
   ```

3. Add `http://`*`domain-name`*`/adfs/services/trust` as a SAML trusted issuer.

   For more information, see "Configuring SAML Trusted Issuers and DN Lists" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

4. Add the Subject DN (as defined in RFC 2253) of the STS certificate in the Trusted STS Servers section. Use a string that conforms to RFC 2253, such as `CN=abc`.   You can use the mechanism of your choice, such as keytool, to view the certificate and determine the Subject DN.

For more information, see "Configuring SAML Trusted Issuers and DN Lists" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

5.10.3 Configure Users in Oracle Internet Directory

For each user, configure the mail attribute to match the user email address set in ADFS.

See Managing Directory Entries for Creating a User in *Oracle Fusion Middleware Administrator's Guide for Oracle Internet Directory* for information on configuring users in Oracle Internet Directory.

5.10.4 Attach the Policy

Attach any of the following OWSM policies to the web service:

- `oracle/wss_sts_issued_saml_bearer_token_over_ssl_service_policy`
- `oracle/wss_saml_token_bearer_over_ssl_service_policy`
- `oracle/wss11_saml_or_username_token_with_message_protection_service_policy`

These policies enforce message protection (integrity and confidentiality) and SAML-based authentication using credentials provided in SAML tokens with the bearer confirmation method in the WS-Security SOAP header. They also verify that the transport protocol provides SSL message protection.

See "Attaching Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager* for information on attaching policies.

5.10.5 Register the Web Service as a Relying Party in ADFS 2.0

Configure ADFS 2.0 to issue the SAML assertion to the web service with the email address or the name ID (SAM-Account-Name) as the subject name id.

See `http://technet.microsoft.com/en-us/library/dd807108%28v=ws.10%29.aspx` for general information on relying parties.

This section provides use case-specific information.

To add the Web Service as a Relying Party:

1. In the AD FS 2.0 Management console, click **AD FS 2.0**.
2. In the details pane, click **Add a trusted relying party** to start the Add Relying Party Wizard.
3. On the Welcome page, click **Start** to begin.
4. Select **Enter data about the relying party manually**.
5. Provide a display name and enter any notes you want.
6. Select **ADFS 2.0 Profile**.
7. On the Configure Certificate page, click **Next**.

   Configuring a token encryption certificate on this page is optional. Configure one on this page if you require that the token be encrypted. If you do not configure a token encryption certificate, the token issued by STS is not encrypted for the service.
8. WS-Trust is always enabled. Click **Next**.

9. For the Relying Party Trust Identifier, enter the service URL and click **Add**.
10. Permit all users to access this relying party.
11. Click **Next** and then **Close**.

5.10.5.1 Configure the Claim Rules for the Service

To enable the STS to send the email address or the name ID as the `subject name id` in the outgoing SAML assertions for the service, use the steps in this section to create a chain of two claim rules with different templates.

See `http://technet.microsoft.com/en-us/library/ee913578%28v=ws.10%29.aspx` for general information on claim rules. See `http://technet.microsoft.com/en-us/library/dd807115%28v=ws.10%29.aspx` to create a rule to send LDAP attributes as claims.

This section provides use case-specific information.

1. Right-click on the Relying Party for the service and select **Edit Claim Rules**.
2. On the Issuance Transform Rules tab select Add Rule.
3. Select **Send LDAP Attribute as Claims** as the claim rule template to use.
4. Give the Claim a name, such as Get LDAP Attributes.
5. Set the Attribute Store to Active Directory, the LDAP Attribute to E-Mail-Addresses, and the Outgoing Claim Type to E-mail Address.

   If you want to instead use the name ID as the subject name ID, under LDAP Attribute, select SAM-Account-Name.
6. Select **Finish**.
7. If you use the name ID as the subject name ID, click **OK** to close the property page and save the changes to the relying party trust.

   If you use the email address as the subject name ID, continue to add a rule.
8. Select **Add Rule**.
9. Select **Transform an Incoming Claim** as the claim rule template to use.
10. Give it a name, such as Email to Name ID.
11. Set the Incoming claim type as E-mail Address. (It must match the Outgoing Claim Type in the previous rule.)
12. Set the Outgoing claim type as Name ID and the Outgoing name ID format as Email (`urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress`).
13. Pass through all claim values and click **Finish**.
14. Click **OK** to close the property page and save the changes to the relying party trust.

5.10.6 Secure WCF/.NET 3.5 Client with ADFS 2.0

Perform the following steps to secure WCF/.NET 3.5 Client with ADFS 2.0:

1. Install .NET 3.5 and Microsoft Visual Studio 2008.
2. Import the SSL server certificates for STS and the service into Windows.

If the SSL server certificate for STS or the service is not issued from a trusted CA, or self-signed, then it needs to be imported with MMC tool, as described in Table 5–18, " Configuration Prerequisites for Interoperability".

3. Create and Configure the WCF Client.

   ADFS 2.0 STS supports multiple security and authentication mechanisms for token insurance. Each is exposed as a separate endpoint. For username/password authentication, two endpoints are provided:

   - `http://<adfs.domain>/adfs/services/trust/13/username` — This endpoint is for username token with message protection.
   - `https://<adfs.domain>/adfs/services/trust/13/usernamemixed` — This endpoint is for username token with transport protection (SSL).

   The WCF client uses the `https://<adfs.domain>/adfs/services/trust/13/usernamemixed` endpoint for username token on SSL to obtain the SAML bearer token for the service.

   a. Generate the WCF Client with the service WSDL.

      See http://msdn.microsoft.com/en-us/library/ms733133(v=vs.90) for information on creating a Windows Communication Foundation client.

   b. Configure the client with `ws2007FederationHttpBinding`:

      In the Solution Explorer of the client project, add a reference by right-clicking on references, selecting **Add reference**, and browsing to *C:*`\Windows\Microsoft.NET\framework\v3.0\Windows Communication Foundation\System.Runtime.Serialization.dll`.

      Edit the `app.config` file. (See http://msdn.microsoft.com/en-us/library/bb472490.aspx for information on WS 2007 Federation HTTP Binding.)   Consider the following sample:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <system.serviceModel>
        <behaviors>
          <endpointBehaviors>
            <behavior name="secureBehaviour">
              <clientCredentials>
                <serviceCertificate>
      <defaultCertificate findValue="weblogic"
            storeLocation="LocalMachine"
            storeName="My"
            x509FindType="FindBySubjectName"/>
                </serviceCertificate>
              </clientCredentials>
            </behavior>
          </endpointBehaviors>
        </behaviors>
      <bindings>
        <ws2007FederationHttpBinding>
          <binding
name="JaxWsWss11SamlOrUsernameOrSamlBearerOverSSLSoapHttp">
            <security mode="TransportWithMessageCredential">
              <message negotiateServiceCredential="false"
             algorithmSuite="Basic128"
                   issuedTokenType
="http://docs.oasis-open.org/wss/oasis-wss-saml-token-
```

```
profile-1.1#SAMLV1.1"
                   issuedKeyType="BearerKey">
                <issuer address
="https://domain-name/adfs/services/trust/13/usernamemixed"
              binding ="ws2007HttpBinding"

bindingConfiguration="ADFSUsernameMixed"/>
              </message>
            </security>
          </binding>
        </ws2007FederationHttpBinding>
        <ws2007HttpBinding>
          <binding name="ADFSUsernameMixed">
            <security mode="TransportWithMessageCredential">
              <message clientCredentialType="UserName"
establishSecurityContext="false" />
            </security>
          </binding>
        </ws2007HttpBinding>
      </bindings>
        <client>
          <endpoint

address="https://adc2170989:8002/JaxWsWss11SamlOrUsernameOrSamlBearerOverSS
L/JaxWsWss11Sam


lOrUsernameOrSamlBearerOverSSLService"
              binding="ws2007FederationHttpBinding"

bindingConfiguration="JaxWsWss11SamlOrUsernameOrSamlBearerOverSSLSoapHttp"
              contract="JaxWsWss11SamlOrUsernameOrSamlBearerOverSSL"


name="JaxWsWss11SamlOrUsernameOrSamlBearerOverSSLPort">
            <identity>
              <dns value="weblogic" />
            </identity>
          </endpoint>
      </client>
    </system.serviceModel>
</configuration>
```

c. Edit the `program.cs` file to make the service call.

If not already present, create a `.cs` file in the project and name it `program.cs` (or any name of your choice.) Edit it to match the following:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.ServiceModel;

namespace Client
{
    class Program
    {
        static void Main(string[] args)
        {
            JaxWsWss11SamlOrUsernameOrSamlBearerOverSSLClient client =
               New JaxWsWss11SamlOrUsernameOrSamlBearerOverSSLClient();
            client.ClientCredentials.UserName.UserName = "joe";
            client.ClientCredentials.UserName.Password = "eoj";




System.Net.ServicePointManager.ServerCertificateValidationCallback =
               ((sender, certificate, chain, sslPolicyErrors) => true);


            Console.WriteLine(client.echo("Hello"));
            Console.Read();
        }

    }
}
```

In this sample `program.cs` file:

`joe` is the username and *`eoj`* is the password used by the client to authenticate to the STS.

`System.Net.ServicePointManager.ServerCertificateValidationCallback =  ((sender, certificate, chain, sslPolicyErrors) => true);` has been added to validate the server side self-signed certificate. This is not required if the server certificate is issued by a trusted CA. If using a self-signed certificate for testing, add this method to validate the certificate on the client side.

6 Interoperability with Microsoft WCF/.NET 4.5 Security Environments

This chapter describes interoperability of Oracle Web Services Manager (OWSM) with Microsoft WCF/.NET 4.5 security environments.

This chapter includes the following sections:

6.1 Overview of Interoperability with Microsoft WCF/.NET 4.5 Security Environments

Oracle has performed interoperability testing to ensure that the web service security policies created using OWSM 12*c* can interoperate with web service policies configured using Microsoft Windows Communication Foundation (WCF)/.NET 4.5 Framework and vice versa.

For more information about the Microsoft .NET 4.5 (and earlier) Framework, see ".NET Development" at `http://msdn.microsoft.com/en-us/library/ff361664%28v=vs.110%29.aspx`.

For more information about:

- OWSM predefined policies, see "Predefined Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.
- Configuring and attaching OWSM 12*c* policies, see "Securing Web Services" and "Attaching Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

Note: In most cases, you can attach OWSM policies in source code, before deploying an application, or you can attach policies post deployment, using WLST or Fusion Middleware Control. To simplify the instructions in this chapter, it is assumed that you are attaching policies post deployment. If a situation *requires* that you attach a policy before deploying, it is described that way in the instructions.

Note: Some of the procedures described in this chapter instruct you to use the Microsoft ServiceModel Metadata Utility Tool (`SvcUtil.exe`) to create a client proxy and configuration file from the deployed web service. However, `SvcUtil.exe` does not work with certain security policy assertions used with OWSM. As a workaround when generating a WCF proxy for a web service protected by an OWSM policy, do the following:

- Detach the policy.
- Generate the proxy using `SvcUtil.exe`.
- Re-attach the policy.

For more information about `SvcUtil.exe`, see `http://msdn.microsoft.com/en-us/library/aa347733%28v=vs.110%29.aspx`.

Table 6–1 and Table 6–2 summarize the most common Microsoft .NET 4.5 interoperability scenarios based on the following security requirements: authentication, message protection, and transport.

Note: In the following scenarios, ensure that you are using a keystore with v3 certificates. By default, the JDK 1.5 keytool generates keystores with v1 certificates.

In addition, ensure that the keys use the proper extensions, including DigitalSignature, Non_repudiation, Key_Encipherment, and Data_Encipherment.

Table 6–1 OWSM 12c Service Policy and Microsoft WCF/.NET 4.5 Client Policy Interoperability

Identity Token	WS-Security Version	Message Protection	Transport Security	Service Policy	Client Policy
MTOM	NA	NA	NA	`oracle/wsmtom_policy`	See Table 6–4, " Configuring the Microsoft WCF/.NET 4.5 Client" on page 6-4

Username or SAML	1.1	Yes	No	`oracle/wss11_username_token_with_message_protection_service_policy`
				OR
				`oracle/wss11_saml_or_username_token_with_message_protection_service_policy`
					See Table 6–8, " Configuring the Microsoft WCF/.NET 4.5 Client" on page 9, Table 6–3, " Configuring the Microsoft WCF/.NET 4.5 Client" on page 15, and
					Section 6.10.6, "Step 6: Secure WCF/.NET 4.5 Client with ADFS 2.0," on page 6-34.

Username	1.0 and 1.1	No	Yes	`oracle/wss_saml_or_username_token_over_ssl_service_policy`
				OR
				`oracle/wss_username_token_over_ssl_service_policy`
					See Table 6–3, " Configuring the Microsoft WCF/.NET 4.5 Client" on page 6-15

Mutual Authentication	1.1	Yes	No	`oracle/wss11_x509_token_with_message_protection_service_policy`	See Table 6–5, " Configuring the Microsoft WCF/.NET 4.5 Client" on page 6-19
Kerberos	1.1	Yes	No	`oracle/wss11_kerberos_token_with_message_protection_service_policy`	See Table 6–9, " Configuring the Microsoft WCF/.NET 4.5 Client" on page 6-24

SAML Bearer	1.0	No	Yes	`oracle/wss_sts_issued_saml_bearer_token_over_ssl_service_policy`
				OR
				`oracle/wss_saml_token_bearer_over_ssl_service_policy`
					See Section 6.10.6, "Step 6: Secure WCF/.NET 4.5 Client with ADFS 2.0," on page 6-34

Table 6–2 Microsoft WCF/.NET 4.5 Service Policy and OWSM 12c Client Policy Interoperability

Identity Token	WS-Security Version	Message Protection	Transport Security	Service Policy	Client Policy
MTOM	NA	NA	NA	See Table 6–5, " Configuring the Microsoft WCF/.NET 4.5 Web Service" on page 6-5	`oracle/wsmtom_policy`

Table 6–2 (Cont.) Microsoft WCF/.NET 4.5 Service Policy and OWSM 12c Client Policy Interoperability

Identity Token	WS-Security Version	Message Protection	Transport Security	Service Policy	Client Policy
Username	1.1	Yes	No	See Table 6–9, " Configuring the Microsoft WCF/.NET 4.5 Web Service" on page 6-11	`oracle/wss11_username_token_with_message_protection_client_policy`
Username Token Over SSL	1.0	No	Yes	See Table 6–4, " Configuring the Microsoft WCF/.NET 4.5 Web Service" on page 6-16	`oracle/wss_username_token_over_ssl_client_policy`
Mutual Authentication	1.1	Yes	No	See Table 6–6, " Configuring the Microsoft WCF/.NET 4.5 Web Service" on page 6-21	`oracle/wss11_x509_token_with_message_protection_client_policy`

6.2 Message Transmission Optimization Mechanism (MTOM)

This section describes how to implement MTOM in the following interoperability scenarios:

- Configuring an OWSM 12c Web Service and a Microsoft WCF/.NET 4.5 Client
- Configuring a Microsoft WCF/.NET 4.5 Web Service and an OWSM 12c Client

6.2.1 Configuring an OWSM 12*c* Web Service and a Microsoft WCF/.NET 4.5 Client

The following instructions tell how to configure an OWSM 12*c* web service and a Microsoft WCF/.NET 4.5 Client to implement Message Transmission Optimization Mechanism (MTOM).

To Configure the OWSM 12c Web Service:

1. Create and deploy a web service application.

   For more information, see "Deploying Web Service Applications" in *Administering Web Services*.

2. Attach the following policy to the web service: `oracle/wsmtom_policy`.

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

To Configure the Microsoft WCF/.NET 4.5 Client:

1. Use the Microsoft SvcUtil utility to create a client proxy and configuration file from the deployed web service. See Example 6–1, "app.config File for MTOM Interoperability".

   For more information, see "ServiceModel Metadata Utility Tool (Svcutil.exe)" at `http://msdn.microsoft.com/en-us/library/aa347733%28v=vs.110%29.aspx`.

2. Run the client program.

   See the following *app.config File for MTOM Interoperability* sample:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <system.serviceModel>
        <bindings>
            <customBinding>
```

```
                <binding name="CustomBinding_IMTOMService">
                    <mtomMessageEncoding maxReadPoolSize="64"
                     maxWritePoolSize="16"
                        messageVersion="Soap12" maxBufferSize="65536"
                        writeEncoding="utf-8">
                        <readerQuotas maxDepth="32" maxStringContentLength=
                         "8192" maxArrayLength="16384"
                            maxBytesPerRead="4096"
maxNameTableCharCount="16384" />
                    </mtomMessageEncoding>
                    <httpTransport manualAddressing="false"
maxBufferPoolSize="524288"
                        maxReceivedMessageSize="65536" allowCookies="false"
                           authenticationScheme="Anonymous"
                        bypassProxyOnLocal="false"
hostNameComparisonMode="StrongWildcard"
                        keepAliveEnabled="true" maxBufferSize="65536"
                           proxyAuthenticationScheme="Anonymous"
                        realm="" transferMode="Buffered"
                           unsafeConnectionNtlmAuthentication="false"
                        useDefaultWebProxy="true" />
                </binding>
            </customBinding>
        </bindings>
        <client>
          <endpoint address="<endpoint_url>"
              binding="customBinding"
bindingConfiguration="CustomBinding_IMTOMService"
              contract="IMTOMService" name="CustomBinding_IMTOMService" >
          </endpoint>
        </client>
    </system.serviceModel>
</configuration>
```

6.2.2 Configuring a Microsoft WCF/.NET 4.5 Web Service and an OWSM 12*c* Client

The following instructions tell how to configure a Microsoft WCF/.NET 4.5 web service and an OWSM 12*c* client to implement Message Transmission Optimization Mechanism (MTOM).

To Configure the Microsoft WCF/.NET 4.5 Web Service:

1. Create a .NET web service.

   For an example, see Example 6–2, ".NET Web Service for MTOM Interoperability". For more information, see "How to: Define a Windows Communication Foundation Service Contract" at `http://msdn.microsoft.com/en-us/library/ms731835.aspx`.

2. Deploy the application.

   See the following *.NET Web Service for MTOM Interoperability* sample:

```
static void Main(string[] args)
{
    string uri = "http://host:port/TEST/MTOMService/SOA/MTOMService";
    // Step 1 of the address configuration procedure: Create a URI to serve as
the base address.
    Uri baseAddress = new Uri(uri);
    // Step 2 of the hosting procedure: Create ServiceHost
    ServiceHost selfHost = new ServiceHost(typeof(MTOMService), baseAddress);

    try {
        HttpTransportBindingElement hb = new HttpTransportBindingElement();
        hb.ManualAddressing = false;
        hb.MaxBufferPoolSize = 2147483647;
        hb.MaxReceivedMessageSize = 2147483647;
        hb.AllowCookies = false;
        hb.AuthenticationScheme = System.Net.AuthenticationSchemes.Anonymous;
        hb.KeepAliveEnabled = true;
        hb.MaxBufferSize = 2147483647;
        hb.ProxyAuthenticationScheme =
System.Net.AuthenticationSchemes.Anonymous;
        hb.Realm = "";
        hb.TransferMode = System.ServiceModel.TransferMode.Buffered;
        hb.UnsafeConnectionNtlmAuthentication = false;
        hb.UseDefaultWebProxy = true;
        MtomMessageEncodingBindingElement me = new
MtomMessageEncodingBindingElement();
        me.MaxReadPoolSize=64;
        me.MaxWritePoolSize=16;
        me.MessageVersion=System.ServiceModel.Channels.MessageVersion.Soap12;
        me.WriteEncoding = System.Text.Encoding.UTF8;
        me.MaxWritePoolSize = 2147483647;
        me.MaxBufferSize = 2147483647;
        me.ReaderQuotas.MaxArrayLength = 2147483647;
        CustomBinding binding1 = new CustomBinding();
        binding1.Elements.Add(me);
        binding1.Elements.Add(hb);
        ServiceEndpoint ep = selfHost.AddServiceEndpoint(typeof(IMTOMService),
binding1,
               "MTOMService");
        EndpointAddress myEndpointAdd = new EndpointAddress(new Uri(uri),
        EndpointIdentity.CreateDnsIdentity("WSMCert3"));
        ep.Address = myEndpointAdd;

        // Step 4 of the hosting procedure: Enable metadata exchange.
        ServiceMetadataBehavior smb = new ServiceMetadataBehavior();
        smb.HttpGetEnabled = true;
        selfHost.Description.Behaviors.Add(smb);
        using (ServiceHost host = new ServiceHost(typeof(MTOMService)))
        {
            System.ServiceModel.Description.ServiceDescription svcDesc =
                 selfHost.Description;
            ServiceDebugBehavior svcDebug =
                  svcDesc.Behaviors.Find<ServiceDebugBehavior>();
            svcDebug.IncludeExceptionDetailInFaults = true;
        }

        // Step 5 of the hosting procedure: Start (and then stop) the service.
        selfHost.Open();
        Console.WriteLine("The service " + uri + " is ready.");
        Console.WriteLine("Press <ENTER> to terminate service.");
        Console.WriteLine();
        Console.ReadLine();
        // Close the ServiceHostBase to shutdown the service.
        selfHost.Close();
    }
    catch (CommunicationException ce)
```

```
        {
            Console.WriteLine("An exception occurred: {0}", ce.Message);
            selfHost.Abort();
        }
    }
```

To Configure the OWSM 12c Client:

1. Using JDeveloper, create a SOA composite that consumes the .NET web service.

   For more information, see Developer's Guide for SOA Suite.

2. Attach the following policy to the web service client: `oracle/wsmtom_policy`.

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

6.3 Username Token With Message Protection (WS-Security 1.1)

This section describes how to implement username token with message protection that conforms to the WS-Security 1.1 standard--with or without secure conversation enabled--in the following interoperability scenarios:

- Configuring an OWSM 12c Web Service and a Microsoft WCF/.NET 4.5 Client
- Configuring a Microsoft WCF/.NET 4.5 Web Service and an OWSM 12c Client

6.3.1 Configuring an OWSM 12*c* Web Service and a Microsoft WCF/.NET 4.5 Client

The following instructions tell how to configure an OWSM 12*c* web service and a Microsoft WCF/.NET 4.5 client to implement username token with message protection that conforms to the WS-Security 1.1 standard, both with and without secure conversation enabled.

To Configure the OWSM 12c Web Service:

1. Create a SOAP 1.2 compliant web service application.
2. Select the policy to use based on whether or not you want to enable secure conversation:

   a. **If you do not want to enable secure conversation**, clone either of the following policies:

   `oracle/wss11_saml_or_username_token_with_message_protection_service_policy`

   `oracle/wss11_username_token_with_message_protection_service_policy`

   Note: In the case of secure conversation *not* enabled, you will have to set the `establishSecurityContext` property to `false` for the client, as described in Table 6–8, " Configuring the Microsoft WCF/.NET 4.5 Client"

   b. **To enable secure conversation,** clone the following policy:

   `oracle/wss11_username_token_with_message_protection_wssc_service_policy`

   For more information, see "Cloning a Web Service Policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

3. Edit the policy configuration settings of the cloned policy from step 2, above, as follows:

   a. Enable the X509 Token Derived Keys configuration setting.

   b. Enable the Encrypt Signature configuration setting.

   c. Disable the Confirm Signature configuration setting.

   d. Leave the default configuration set for all other configuration settings.

   Attach the policy to the web service. For more information, see "Attaching Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

4. Also attach the following policy:

   ```
   oracle/wsaddr_policy
   ```

5. Export the X.509 certificate file from the keystore on the service side to a `.cer` file (for example, `alice.cer`) using the following command:

   ```
   keytool -export -alias alice -file C:\alice.cer -keystore default-keystore.jks
   ```

   For more information, see "keytool - Key and Certificate Management Tool" at `http://docs.oracle.com/javase/6/docs/technotes/tools/windows/keytool.html`

To Configure the Microsoft WCF/.NET 4.5 Client:

1. Import the certificate file (exported previously) to the keystore on the client server using Microsoft Management Console (mmc), as follows:

   a. Open a command prompt.

   b. Type **mmc** and press **Enter**.

   c. Select **File > Add/Remove snap-in**.

   d. Select **Add and Choose Certificates**.

   > **Note:** To view certificates in the local machine store, you must be in the Administrator role.

   e. Select **Add**.

   f. Select **My user account and finish**.

   g. Click **OK**.

   h. Expand **Console Root > Certificates -Current user > Personal > Certificates**.

   i. Right-click on **Certificates** and select **All tasks > Import** to launch Certificate import Wizard.

   j. Click **Next**, select **Browse**, and navigate to the `.cer` file that was exported previously.

   k. Click **Next** and accept defaults and finish the wizard.

   For more information, see "How to: View Certificates with the MMC Snap-in" at `http://msdn.microsoft.com/en-us/library/ms788967.aspx`

2. Generate a .NET client using the WSDL of the web service.

> **Note:** You may have to set WS-Addressing action headers to prevent the client from sending implicit `wsa:Action` headers, as described in "Implicitly Associating WS-Addressing Action Properties" in *Developing JAX-WS Web Services for Oracle WebLogic Server*.

For more information, see "How to: Create a Windows Communication Foundation Client" at `http://msdn.microsoft.com/en-us/library/ms733133(v=vs.110).aspx`

3. Edit the `app.config` file in the .NET project to update the certificate file and disable replays, as shown in Example 6–3. (Changes are identified in **`bold`**). If you follow the default key setup, then `<certificate_cn>` should be set to `alice`.
4. The `establishSecurityContext` property in the `app.config` file must be set according to whether you are enabling secure conversation.

   By default, `establishSecurityContext` is set to `true`, enabling secure conversation. If you are *not* enabling secure conversation, set `establishSecurityContext` to `false`.

   For example, see Example 6–3 (lines in ***`bold italic`***).
5. Compile the project.
6. Open a command prompt and navigate to the project's Debug folder.
7. Enter `<client_project_name>.exe` and press **Enter**.

See the following *app.config File for Implementing Username Token With Message Protection (WS-Security 1.1)* sample:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.serviceModel>
    <behaviors>
      <endpointBehaviors>
         <behavior name="secureBehaviour">
           <clientCredentials>
             <serviceCertificate>
               <defaultCertificate findValue="<certificate_cn>"
                storeLocation="CurrentUser" storeName="My"
                x509FindType="FindBySubjectName"/>
             </serviceCertificate>
           </clientCredentials>
         </behavior>
      </endpointBehaviors>
    </behaviors>
    <bindings>
      <ws2007HttpBinding>
        <binding
name="Wss11UsernameTokenWithMessageProtectionWSSCServicePortBinding" >
          <security mode="Message">
            <message clientCredentialType="UserName"
                negotiateServiceCredential="false"
                algorithmSuite="Basic128"
                establishSecurityContext="true" />
                <!-- extablishSecurityContext is true by default and therefore
does not
                have to be specified to enable secure conversation.
               Set establishSecurityContext to false if secure conversation is
not enabled -->
```

```
          </security>
        </binding>
      </ws2007HttpBinding>
    </bindings>
  <client>
    <endpoint
address="http://10.244.167.70:7003/OWSMTestApp-Project1-context-root/ws11_usern
ame_token_with_message_protection_wsscPort?wsdl"
        behaviorConfiguration="PMCert"
        binding="ws2007HttpBinding"

bindingConfiguration="Wss11UsernameTokenWithMessageProtectionWSSCServicePortBin
ding"

contract="ServiceReference1.ws11_username_token_with_message_protection_wssc"
        name="ws11_username_token_with_message_protection_wsscPort">
      <identity>
        <dns value="orakey" />
      </identity>
    </endpoint>
  </client>
  </system.serviceModel>
</configuration>
```

6.3.2 Configuring a Microsoft WCF/.NET 4.5 Web Service and an OWSM 12*c* Client

The following instructions tell how to configure a Microsoft WCF/.NET 4.5 web service and an OWSM 12*c* client to implement username token with message protection that conforms to the WS-Security 1.1 standard.

To Configure the Microsoft WCF/.NET 4.5 Web Service:

1. Create a .NET web service.

   a. Create a custom binding for the web service using the `SymmetricSecurityBindingElement`, as shown in Example 6–4. This example shows a web service without secure conversation enabled.

   To enable secure conversation, make the following adjustments to the code in the example:

   a. Create another `SymmetricSecurityBindingElement` element based on the one created (`sm`), for example:

      ```
      SymmetricSecurityBindingElement scsm =
      SymmetricSecurityBindingElement.createSecureConversationBindingELement(sm,
      false)
      ```

   b. Create a new custom binding:

      ```
      CustomBinding binding1 = new CustomBinding(scsm);
      ```

   For more information, see "How to: Define a Windows Communication Foundation Service Contract" at http://msdn.microsoft.com/en-us/library/ms731835.aspx

2. Create and import a certificate file to the keystore on the web service server.

   Using Microsoft Visual Studio, the command would be similar to the following:

   ```
   makecert -r -pe -n "CN=wsmcert3" -sky exchange -ss my C:\wsmcert3.cer
   ```

   This command creates and imports a certificate in mmc. If the command does not provide expected results, then try the following sequence of commands. You need to download Windows Developer Kit (WDK) at http://www.microsoft.com/whdc/devtools/WDK/default.mspx

```
makecert -r -pe -n "CN=wsmcert3" -sky exchange -ss my -sv wscert3.pvk
C:\wsmcert3.cer
pvk2pfx.exe -pvk wscert3.pvk -spc wsmcert3.cer -pfx PRF_WSMCert3.pfx -pi
welcome1
```

Then, in mmc, import `PRF_WSMCert3.pfx`.

3. Import the certificate created on the web service server to the client server using the `keytool` command. For example:

```
keytool -import -alias wsmcert3 -file C:\wsmcert3.cer -keystore
<owsm_client_keystore>
```

For more information, see "keytool - Key and Certificate Management Tool" at http://docs.oracle.com/javase/6/docs/technotes/tools/windows/keytool.html

4. Right-click on the web service Solution project in Solutions Explorer and click **Open Folder In Windows Explorer**.

5. Navigate to the `bin/Debug` folder.

6. Double-click the `<project>.exe` file. This command runs the web service at the URL provided.

See the following *.NET Web Service* sample:

```
static void Main(string[] args)
{
    // Step 1 of the address configuration procedure: Create a URI to serve as
the
    // base address.
    // Step 2 of the hosting procedure: Create ServiceHost
    string uri = "http://host:port/TEST/NetService";
    Uri baseAddress = new Uri(uri);

    ServiceHost selfHost = new ServiceHost(typeof(CalculatorService),
baseAddress);

    try
    {
        SymmetricSecurityBindingElement sm =

SymmetricSecurityBindingElement.CreateUserNameForCertificateBindingElement();
        sm.DefaultAlgorithmSuite =
System.ServiceModel.Security.SecurityAlgorithmSuite.Basic128;
        sm.SetKeyDerivation(false);
        sm.SecurityHeaderLayout = SecurityHeaderLayout.Lax;
        sm.IncludeTimestamp = true;
        sm.KeyEntropyMode = SecurityKeyEntropyMode.CombinedEntropy;
        sm.MessageSecurityVersion =

MessageSecurityVersion.WSSecurity11WSTrustFebruary2005WSSecureConversationFebru
ary2005
        WSSecurityPolicy11BasicSecurityProfile10;
        sm.LocalClientSettings.CacheCookies = true;
        sm.LocalClientSettings.DetectReplays = true;
        sm.LocalClientSettings.ReplayCacheSize = 900000;
        sm.LocalClientSettings.MaxClockSkew = new TimeSpan(00, 05, 00);
        sm.LocalClientSettings.MaxCookieCachingTime = TimeSpan.MaxValue;
        sm.LocalClientSettings.ReplayWindow = new TimeSpan(00, 05, 00); ;
        sm.LocalClientSettings.SessionKeyRenewalInterval = new TimeSpan(10, 00,
00);
        sm.LocalClientSettings.SessionKeyRolloverInterval = new TimeSpan(00,
05, 00); ;
        sm.LocalClientSettings.ReconnectTransportOnFailure = true;
       sm.LocalClientSettings.TimestampValidityDuration = new TimeSpan(00, 05,
00); ;
        sm.LocalClientSettings.CookieRenewalThresholdPercentage = 60;
        sm.LocalServiceSettings.DetectReplays = false;
        sm.LocalServiceSettings.IssuedCookieLifetime = new TimeSpan(10, 00,
00);
        sm.LocalServiceSettings.MaxStatefulNegotiations = 128;
        sm.LocalServiceSettings.ReplayCacheSize = 900000;
        sm.LocalServiceSettings.MaxClockSkew = new TimeSpan(00, 05, 00);
        sm.LocalServiceSettings.NegotiationTimeout = new TimeSpan(00, 01, 00);
        sm.LocalServiceSettings.ReplayWindow = new TimeSpan(00, 05, 00);
        sm.LocalServiceSettings.InactivityTimeout = new TimeSpan(00, 02, 00);
        sm.LocalServiceSettings.SessionKeyRenewalInterval = new TimeSpan(15,
00, 00);
        sm.LocalServiceSettings.SessionKeyRolloverInterval = new TimeSpan(00,
05, 00);
        sm.LocalServiceSettings.ReconnectTransportOnFailure = true;
        sm.LocalServiceSettings.MaxPendingSessions = 128;
        sm.LocalServiceSettings.MaxCachedCookies = 1000;
        sm.LocalServiceSettings.TimestampValidityDuration = new TimeSpan(15,
00, 00);
        HttpTransportBindingElement hb = new HttpTransportBindingElement();
        hb.ManualAddressing = false;
        hb.MaxBufferPoolSize = 524288;
        hb.MaxReceivedMessageSize = 65536;
        hb.AllowCookies = false;
        hb.AuthenticationScheme = System.Net.AuthenticationSchemes.Anonymous;
        hb.KeepAliveEnabled = true;
        hb.MaxBufferSize = 65536;
        hb.ProxyAuthenticationScheme =
System.Net.AuthenticationSchemes.Anonymous;
        hb.Realm = "";
        hb.TransferMode = System.ServiceModel.TransferMode.Buffered;
        hb.UnsafeConnectionNtlmAuthentication = false;
        hb.UseDefaultWebProxy = true;
        TextMessageEncodingBindingElement tb1 = new
TextMessageEncodingBindingElement();
        tb1.MaxReadPoolSize = 64;
        tb1.MaxWritePoolSize = 16;
        tb1.MessageVersion =
System.ServiceModel.Channels.MessageVersion.Soap12;
        tb1.WriteEncoding = System.Text.Encoding.UTF8;
        CustomBinding binding1 = new CustomBinding(sm);
        binding1.Elements.Add(tb1);
        binding1.Elements.Add(hb);
        ServiceEndpoint ep = selfHost.AddServiceEndpoint(typeof(ICalculator),
binding1,
          "CalculatorService");

        EndpointAddress myEndpointAdd = new EndpointAddress(
        new Uri(uri),
        EndpointIdentity.CreateDnsIdentity("WSMCert3"));
        ep.Address = myEndpointAdd;

        // Step 4 of the hosting procedure: Enable metadata exchange.
        ServiceMetadataBehavior smb = new ServiceMetadataBehavior();
        smb.HttpGetEnabled = true;
        selfHost.Description.Behaviors.Add(smb);
selfHost.Credentials.ServiceCertificate.SetCertificate(StoreLocation.CurrentUse
r,
           StoreName.My,
        X509FindType.FindBySubjectName, "WSMCert3");

selfHost.Credentials.ClientCertificate.Authentication.CertificateValidationMode
=
            X509CertificateValidationMode.PeerOrChainTrust;

selfHost.Credentials.UserNameAuthentication.UserNamePasswordValidationMode =
            UserNamePasswordValidationMode.Custom;
        CustomUserNameValidator cu = new CustomUserNameValidator();

selfHost.Credentials.UserNameAuthentication.CustomUserNamePasswordValidator =
cu;
        using (ServiceHost host = new ServiceHost(typeof(CalculatorService)))
        {
            System.ServiceModel.Description.ServiceDescription svcDesc =
selfHost.Description;
            ServiceDebugBehavior svcDebug =
svcDesc.Behaviors.Find<ServiceDebugBehavior>();
            svcDebug.IncludeExceptionDetailInFaults = true;
        }

        // Step 5 of the hosting procedure: Start (and then stop) the service.
        selfHost.Open();
        Console.WriteLine("The Calculator service is ready.");
        Console.WriteLine("Press <ENTER> to terminate service.");
        Console.WriteLine();
        Console.ReadLine();
        selfHost.Close();
    }
    catch (CommunicationException ce)
    {
         Console.WriteLine("An exception occurred: {0}", ce.Message);
         selfHost.Abort();
     }
}
```

To Configure the OWSM 12c Client:

1. Using JDeveloper, create a SOA composite that consumes the .NET web service. For more information, see Developer's Guide for SOA Suite.
2. In JDeveloper, create a partner link using the WSDL of the .NET service.
3. Attach the following policy to the web service client: `oracle/wss11_username_token_with_message_protection_client_policy`.

   For more information, see "Attaching Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.
4. Provide configurations for the `csf-key` and `keystore.recipient.alias`.

   You can specify this information when attaching the policy, by overriding the policy configuration. For more information.

   Ensure that you configure the `keystore.recipient.alias` as the alias of the certificate imported in step 1 (`wsmcert3`). For example:

   ```
   <wsp:PolicyReference
         URI="oracle/wss11_username_token_with_message_protection_client_policy"
   ```

```
      orawsp:category="security"
      orawsp:status="enabled"/>
   <property
      name="csf-key"
      type="xs:string"
      many="false">
      basic.credentials
   </property>
   <property
      name="keystore.recipient.alias"
      type="xs:string"
      many="false">
      wsmcert3
   </property>
```

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

6.4 Username Token Over SSL

This section describes how to implement username token over SSL in the following interoperability scenario:

- Configuring an OWSM 12c Web Service and a Microsoft WCF/.NET 4.5 Client

6.4.1 Configuring an OWSM 12*c* Web Service and a Microsoft WCF/.NET 4.5 Client

The following instructions tell how to configure a OWSM 12*c* web service and a Microsoft WCF/.NET 4.5 client to implement username token over SSL, both with and without secure conversation enabled.

To Configure the OWSM 12c Web Service:

1. Configure the server for SSL.

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

2. Create an OWSM web service.

3. Select the policy to use based on whether or not you want to enable secure conversation:

   If you do not want to enable secure conversation, attach any of the following policies:

   `oracle/wss_username_token_over_ssl_service_policy`

   `oracle/wss_saml_or_username_token_over_ssl_service_policy`

   oracle/wss11_saml_or_username_token_with_message_protection_service_policy

   > **Note:** In the case of secure conversation *not* enabled, you will have to set the `establishSecurityContext` property to `false` for the client, as described in Table 6–8, " Configuring the Microsoft WCF/.NET 4.5 Client"

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

4. Specify that addressing is to be used, as follows:

   For an Oracle Infrastructure web service:

   Attach the following policy:

   `oracle/wssaddr_policy`

   For a Java EE web service:

   Only a subset of OWSM security policies are supported for Java EE web services and clients, so you cannot attach `oracle/wssaddr_policy` to a Java EE web service. Rather you must add addressing information using the `@Addressing` annotation in the source code for the service, as shown in Example 6–5.

   For more information, see the following:

   - "Attaching Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*
   - "Which OWSM Policies Are Supported for Java EE Web Services and Clients?" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*
   - "Attaching Policies to Java EE Web Services and Clients at Design TIme" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*

     See the following *Java EE Web Service with Addressing* sample:

     ```
     package oracle.wsm.qa.wls.service.soap12;
     import javax.jws.WebMethod;
     import javax.jws.WebParam;
     import javax.jws.WebService;
     import javax.xml.ws.BindingType;
     import javax.xml.ws.soap.Addressing;
     import javax.xml.ws.soap.SOAPBinding;
     import weblogic.wsee.jws.jaxws.owsm.SecurityPolicy;
     @WebService
     @BindingType(SOAPBinding.SOAP12HTTP_BINDING)
     @Addressing(enabled=true)
     public class wss_username_token_over_ssl {
       public wss_username_token_over_ssl() {
         super();
       }
       @WebMethod
       public String sayHello(@WebParam(name = "arg0") String name){
         return "hello "+ name;
       }
     }
     ```

To Configure the Microsoft WCF/.NET 4.5 Client:

1. Generate a .NET client using the WSDL of the web service.

   For more information, see "How to: Create a Windows Communication Foundation Client" at
   `http://msdn.microsoft.com/en-us/library/ms733133(v=vs.110).aspx`

2. The `establishSecurityContext` property in the `app.config` file must be set according to whether you are enabling secure conversation.

   By default, `establishSecurityContext` is set to `true`, enabling secure conversation. If you are *not* enabling secure conversation, set `establishSecurityContext` to `false`.

   For example, see Example 6–6 (lines in ***bold italic***).

3. Compile the project.
4. Open a command prompt and navigate to the project's Debug folder.
5. Type `<client_project_name>.exe` and press **Enter**.

   See the following *app.config file* sample:

```
<configuration>
  <system.serviceModel>
    <bindings>
      <ws2007HttpBinding>
        <binding name="wss_username_over_ssl_client">
          <security mode="TransportWithMessageCredential">
            <transport clientCredentialType="None" />
            <message clientCredentialType="UserName"
                negotiateServiceCredential="false"
                establishSecurityContext="true" />
                <!-- extablishSecurityContext is true by default and therefore
does not
                have to be specified to enable secure conversation.
               Set establishSecurityContext to false if secure conversation is
not enabled -->
          </security>
        </binding>
      </ws2007HttpBinding>
    </bindings>
    <client>
      <endpoint
address="https://10.244.167.70:7004/OWSMTestApp-Project1-context-root/wss_usern
ame_token_over_sslPort"
          binding="ws2007HttpBinding"
          bindingConfiguration="wss_username_over_ssl_client"
          contract="ServiceReference1.wss_username_token_over_ssl"
          name="wss_username_token_over_sslPort" />
    </client>
  </system.serviceModel>
</configuration>
```

6.4.2 Configuring a Microsoft WCF/.NET 4.5 Web Service and an OWSM 12*c* Client

The following instructions tell how to configure a Microsoft WCF/.NET 4.5 web service and an OWSM 12*c* client to implement username token over SSL.

Configuring the Microsoft WCF/.NET 4.5 Web Service:

1. Configure the server for SSL.

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

2. Create a .NET web service.

   a. Create a custom binding for the web service using the `SecurityBindingElement`, as shown in Example 6–7. This example shows a web service without secure conversation enabled.

   To enable secure conversation, make the following adjustments to the code in the example:

   a. Create another `SecurityBindingElement` element based on the one created (`sm`), for example:

   ```
   SecurityBindingElement scsm =
   ```

```
SecurityBindingElement.createSecureConversationBindingElement(sm)
```

b. Create the custom binding with scsm:

```
CustomBinding binding1 = new CustomBinding(scsm);
```

For more information, see "How to: Define a Windows Communication Foundation Service Contract" at http://msdn.microsoft.com/en-us/library/ms731835.aspx

See the following *.NET Web Service* sample:

```
static void Main(string[] args)
{
    // Step 1 of the address configuration procedure: Create a URI to serve as 
the
    // base address.
    // Step 2 of the hosting procedure: Create ServiceHost
    string uri = "http://host:port/TEST/NetService";
    Uri baseAddress = new Uri(uri);

    ServiceHost selfHost = new ServiceHost(typeof(CalculatorService), 
baseAddress);

    try
    {
        SecurityBindingElement sm =
            SecurityBindingElement.CreateUserNameOverTransportBindingElement();
        sm.DefaultAlgorithmSuite = 
System.ServiceModel.Security.SecurityAlgorithmSuite.Basic128;
        sm.SetKeyDerivation(false);
        sm.SecurityHeaderLayout = SecurityHeaderLayout.Lax;
        sm.IncludeTimestamp = true;
        sm.KeyEntropyMode = SecurityKeyEntropyMode.CombinedEntropy;
        sm.MessageSecurityVersion = 

MessageSecurityVersion.WSSecurity11WSTrustFebruary2005WSSecureConversationFebru
ary2005
        WSSecurityPolicy11BasicSecurityProfile10;
        sm.LocalClientSettings.CacheCookies = true;
        sm.LocalClientSettings.DetectReplays = true;
        sm.LocalClientSettings.ReplayCacheSize = 900000;
        sm.LocalClientSettings.MaxClockSkew = new TimeSpan(00, 05, 00);
        sm.LocalClientSettings.MaxCookieCachingTime = TimeSpan.MaxValue;
        sm.LocalClientSettings.ReplayWindow = new TimeSpan(00, 05, 00); ;
       sm.LocalClientSettings.SessionKeyRenewalInterval = new TimeSpan(10, 00, 
00);
        sm.LocalClientSettings.SessionKeyRolloverInterval = new TimeSpan(00, 
05, 00); ;
        sm.LocalClientSettings.ReconnectTransportOnFailure = true;
       sm.LocalClientSettings.TimestampValidityDuration = new TimeSpan(00, 05, 
00); ;
        sm.LocalClientSettings.CookieRenewalThresholdPercentage = 60;
        sm.LocalServiceSettings.DetectReplays = false;
        sm.LocalServiceSettings.IssuedCookieLifetime = new TimeSpan(10, 00, 
00);
        sm.LocalServiceSettings.MaxStatefulNegotiations = 128;
        sm.LocalServiceSettings.ReplayCacheSize = 900000;
        sm.LocalServiceSettings.MaxClockSkew = new TimeSpan(00, 05, 00);
        sm.LocalServiceSettings.NegotiationTimeout = new TimeSpan(00, 01, 00);
        sm.LocalServiceSettings.ReplayWindow = new TimeSpan(00, 05, 00);
        sm.LocalServiceSettings.InactivityTimeout = new TimeSpan(00, 02, 00);
        sm.LocalServiceSettings.SessionKeyRenewalInterval = new TimeSpan(15, 
00, 00);
        sm.LocalServiceSettings.SessionKeyRolloverInterval = new TimeSpan(00,
05, 00);
        sm.LocalServiceSettings.ReconnectTransportOnFailure = true;
        sm.LocalServiceSettings.MaxPendingSessions = 128;
        sm.LocalServiceSettings.MaxCachedCookies = 1000;
        sm.LocalServiceSettings.TimestampValidityDuration = new TimeSpan(15,
00, 00);
        HttpTransportBindingElement hb = new HttpTransportBindingElement();
        hb.ManualAddressing = false;
        hb.MaxBufferPoolSize = 524288;
        hb.MaxReceivedMessageSize = 65536;
        hb.AllowCookies = false;
        hb.AuthenticationScheme = System.Net.AuthenticationSchemes.Anonymous;
        hb.KeepAliveEnabled = true;
        hb.MaxBufferSize = 65536;
        hb.ProxyAuthenticationScheme =
System.Net.AuthenticationSchemes.Anonymous;
        hb.Realm = "";
        hb.TransferMode = System.ServiceModel.TransferMode.Buffered;
        hb.UnsafeConnectionNtlmAuthentication = false;
        hb.UseDefaultWebProxy = true;
        TextMessageEncodingBindingElement tb1 = new
TextMessageEncodingBindingElement();
        tb1.MaxReadPoolSize = 64;
        tb1.MaxWritePoolSize = 16;
        tb1.MessageVersion =
System.ServiceModel.Channels.MessageVersion.Soap12;
        tb1.WriteEncoding = System.Text.Encoding.UTF8;
        CustomBinding binding1 = new CustomBinding(sm);
        binding1.Elements.Add(tb1);
        binding1.Elements.Add(hb);
        ServiceEndpoint ep = selfHost.AddServiceEndpoint(typeof(ICalculator),
binding1,
          "CalculatorService");

        EndpointAddress myEndpointAdd = new EndpointAddress(
        new Uri(uri),
        EndpointIdentity.CreateDnsIdentity("WSMCert3"));
        ep.Address = myEndpointAdd;

        // Step 4 of the hosting procedure: Enable metadata exchange.
        ServiceMetadataBehavior smb = new ServiceMetadataBehavior();
        smb.HttpGetEnabled = true;
        selfHost.Description.Behaviors.Add(smb);

selfHost.Credentials.ServiceCertificate.SetCertificate(StoreLocation.CurrentUse
r,
           StoreName.My,
        X509FindType.FindBySubjectName, "WSMCert3");

selfHost.Credentials.ClientCertificate.Authentication.CertificateValidationMode
=
            X509CertificateValidationMode.PeerOrChainTrust;

selfHost.Credentials.UserNameAuthentication.UserNamePasswordValidationMode =
            UserNamePasswordValidationMode.Custom;
        CustomUserNameValidator cu = new CustomUserNameValidator();

selfHost.Credentials.UserNameAuthentication.CustomUserNamePasswordValidator =
cu;
        using (ServiceHost host = new ServiceHost(typeof(CalculatorService)))
        {
            System.ServiceModel.Description.ServiceDescription svcDesc =
selfHost.Description;
            ServiceDebugBehavior svcDebug =
svcDesc.Behaviors.Find<ServiceDebugBehavior>();
            svcDebug.IncludeExceptionDetailInFaults = true;
        }

        // Step 5 of the hosting procedure: Start (and then stop) the service.
        selfHost.Open();
        Console.WriteLine("The Calculator service is ready.");
        Console.WriteLine("Press <ENTER> to terminate service.");
        Console.WriteLine();
        Console.ReadLine();
        selfHost.Close();
    }
    catch (CommunicationException ce)
    {
         Console.WriteLine("An exception occurred: {0}", ce.Message);
         selfHost.Abort();
     }
}
```

To Configure the OWSM 12c Client:

1. Generate an OWSM client using the WSDL of the web service.

   For more information, see Developer's Guide for SOA Suite.

2. Attach the following policy to the client:

   ```
   oracle/wss_username_token_over_ssl_client_policy
   ```

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

6.5 Mutual Authentication with Message Protection (WS-Security 1.1)

This section describes how to implement mutual authentication with message protection that conform to the WS-Security 1.1 standards in the following interoperability scenarios:

- Configuring an OWSM 12c Web Service and a Microsoft WCF/.NET 4.5 Client
- Configuring a Microsoft WCF/.NET 4.5 Web Service and an OWSM 12c Client

Before configuring the web service and client in either of the above scenarios, follow the instructions in "Configuration Prerequisites".

6.5.1 Configuration Prerequisites

The following procedure describes how to perform prerequisite configuration tasks for implementing mutual authentication with message protection that conform to the WS-Security 1.1 standards.

To Configure Prerequisites for Interoperability:

1. Export the X.509 certificate file from the keystore on the service side to a `.cer` file (for example, `alice.cer`) using the following command:

   ```
   keytool -export -alias alice -file C:\alice.cer -keystore default-keystore.jks
   ```

For more information, see "keytool - Key and Certificate Management Tool" at http://docs.oracle.com/javase/6/docs/technotes/tools/windows/keytool.html

2. Import the certificate file (exported previously) to the keystore on the client server using Microsoft Management Console (mmc). See step 1 in Table 6–8, " Configuring the Microsoft WCF/.NET 4.5 Client" for specific instructions.

   For more information, "How to: View Certificates with the MMC Snap-in" at http://msdn.microsoft.com/en-us/library/ms788967.aspx.

6.5.2 Configuring an OWSM 12*c* Web Service and a Microsoft WCF/.NET 4.5 Client

The following instructions tell how to configure an OWSM 12*c* web service and a Microsoft WCF/.NET 4.5 client to implement mutual authentication with message protection that conform to the WS-Security 1.1 standards.

To Configure the OWSM 12c Web Service:

1. Create a SOAP 1.2 compliant SOA composite and deploy it.
2. Using Fusion Middleware Control, attach the following policy to the web service:

   `oracle/wss11_x509_token_with_message_protection_service_policy`

   For more information, see "Attaching Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.
3. Export `wss11_x509_token_with_message_protection_service_policy_net`. Change `encrypted="true"` to `"false"`, and import it back.

   ```
   <orasp:x509-token
      orasp:enc-key-ref-mech="thumbprint"
      orasp:is-encrypted="false"
      orasp:is-signed="false"
      orasp:sign-key-ref-mech="direct"/>
   ```

   For more information, see the following links:

   - "Exporting Web Service Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.
   - "Importing Web Service Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*
4. Attach the policy to the web service.

   For more information, see "Attaching Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.
5. Attach the following policy:

   `oracle/wsaddr_policy`

To Configure the Microsoft WCF/.NET 4.5 Client:

1. Use the Microsoft SvcUtil utility to create a client proxy (see Example 6–9, "Client Program") and configuration file from the deployed web service.

   For more information, see http://msdn.microsoft.com/en-us/library/aa347733%28v=vs.110%29.aspx
2. Create a `app.config` configuration file, as shown in Example 6–8, "app.config File".
3. Compile the project.

4. Open a command prompt and navigate to the project's Debug folder.
5. Enter `<client_project_name>.exe` and press **Enter**.

   See the following *app.config file* sample:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.serviceModel>
    <behaviors>
      <endpointBehaviors>
        <behavior name="secureBehaviour">
          <clientCredentials>
            <serviceCertificate>
              <defaultCertificate findValue="<certificate_cn>"
                                  storeLocation="CurrentUser"
                                  storeName="My"
                                  x509FindType="FindBySubjectName"/>
            </serviceCertificate>
          </clientCredentials>
        </behavior>
      </endpointBehaviors>
    </behaviors>
      <bindings>
        <ws2007HttpBinding>
          <binding name="wss_username_over_ssl_client">
            <security mode="TransportWithMessageCredential">
            <transport clientCredentialType="None" />
            <message clientCredentialType="UserName"
                negotiateServiceCredential="false"
                establishSecurityContext="false" />
            </security>
          </binding>
        </ws2007HttpBinding>
     </bindings>
        <client>
          <endpoint address="http://<server>:<port>//MyWebService1SoapHttpPort"
                binding="ws2007HttpBinding"
                contract="MyWebService1"
                name="MyWebService1SoapHttpPort"
                behaviorConfiguration="secureBehaviour" >
            <identity>
              <dns value="<certificate_cn>"/>
            </identity>
          </endpoint>
        </client>
    </system.serviceModel>
</configuration>
```

   See the following *Client Program* sample:

```
 namespace IO_NET10_client
{
    class Program
    {
        static void Main(string[] args)
        {

            BPELProcess1Client client = new BPELProcess1Client();

            client.ClientCredentials.ClientCertificate.SetCertificate(
                    StoreLocation.CurrentUser,
                    StoreName.My,
                    X509FindType.FindBySubjectName, "WSMCert3");

             client.ClientCredentials.ServiceCertificate.SetDefaultCertificate(
                       StoreLocation.CurrentUser,
                       StoreName.My,
                    X509FindType.FindBySubjectName, "Alice");

            process proc = new process();
            proc.input = "Test wss11_x509_token_with_message_protection_policy
- ";
            Console.WriteLine(proc.input);
            processResponse response = client.process(proc);

            Console.WriteLine(response.result.ToString());
            Console.WriteLine("Press <ENTER> to terminate Client.");
            Console.ReadLine();
          }
    }
}
```

6.5.3 Configuring a Microsoft WCF/.NET 4.5 Web Service and an OWSM 12*c* Client

The following instructions tell how to configure a Microsoft WCF/.NET 4.5 web service and an OWSM 12*c* client to implement mutual authentication with message protection that conform to the WS-Security 1.1 standards.

To Configure the Microsoft WCF/.NET 4.5 Web Service:

1. Create a .NET web service.

   For an example, see Example 6–4, "Example of .NET Web Service".

   For more information, see How to: Define a Windows Communication Foundation Service Contract" at http://msdn.microsoft.com/en-us/library/ms731835%28v=vs.90%29.aspx

2. Create a custom binding for the web service using the SymmetricSecurityBindingElement.

   The following is a sample of the SymmetricSecurityBindingElement object:

   ```
   SymmetricSecurityBindingElement sm =
   (SymmetricSecurityBindingElement)SecurityBindingElement.CreateMutualCertificate
   BindingElement();
   sm.DefaultAlgorithmSuite =
   System.ServiceModel.Security.SecurityAlgorithmSuite.Basic128;sm.SetKeyDerivati
   on(false);
   sm.SecurityHeaderLayout = SecurityHeaderLayout.Lax;sm.IncludeTimestamp =
   true;
   sm.KeyEntropyMode = SecurityKeyEntropyMode.CombinedEntropy;
   sm.MessageProtectionOrder =
   MessageProtectionOrder.SignBeforeEncrypt;sm.MessageSecurityVersion =
   MessageSecurityVersion.WSSecurity11WSTrustFebruary2005WSSecureConversation
   February2005WSSecurityPolicy11BasicSecurityProfile10;
   sm.RequireSignatureConfirmation =
   true;
   ```

   For more information, see "How to: Create a Custom Binding Using the SecurityBindingElement" at http://msdn.microsoft.com/en-us/library/ms730305%28v=vs.90%29.aspx

3. Deploy the application.

To Configure the OWSM 12c Client:

1. Using JDeveloper, create a SOA composite that consumes the .NET web service.

   For more information, see Developer's Guide for SOA Suite.

2. In JDeveloper, create a partner link using the WSDL of the .NET service and add the import as follows:

   ```
   <wsdl:import namespace="<namespace>" location="<WSDL location>"/>
   ```

3. In Fusion Middleware Control, attach the following policy to the web service client:

   `oracle/wss11_x509_token_with_message_protection_client_policy`.

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

4. Provide configurations for the `keystore.recipient.alias`.

   You can specify this information when attaching the policy, by overriding the policy configuration.

   Ensure that you configure the `keystore.recipient.alias` as the alias of the certificate imported in step 4 (`wsmcert3`).

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

5. Invoke the web service method from the client.

6.6 Kerberos with Message Protection

This section describes how to implement Kerberos with message protection in the following interoperability scenario:

- Configuring an OWSM 12c Web Service and a Microsoft WCF/.NET 4.5 Client

6.6.1 Performing Prerequisite Tasks for Interoperability

To Configure Prerequisites for Interoperability:

1. Configure the Key Distribution Center (KDC) and Active Directory (AD).

   For more information, see "To Configure Windows Active Directory and Domain Controller" (the domain controller can serve as KDC) at `http://download.oracle.com/docs/cd/E19316-01/820-3746/gisdn/index.html`.

2. Set up the Kerberos configuration file `krb5.conf` in `c:\winnt` as shown in the following "Kerberos Configuration File" sample.

   ```
   [logging]
   default = c:\log\krb5libs.log
   kdc = c:\log\krb5kdc.log
   admin_server = c:\log\kadmind.log
   [libdefaults]
   default_realm = MYCOMPANY.LOCAL
   dns_lookup_realm = false
   dns_lookup_kdc = false
   default_tkt_enctypes = rc4-hmac
   default_tgs_enctypes = rc4-hmac
   permitted_enctypes = rc4-hmac
   kdc = hostname
   ```

```
[realms]
MYCOMPANY.LOCAL =
{ kdc = host:port  admin_server = host:port
  default_domain = <domainname>
}
 [domain_realm]
.<domainname> = MYCOMPANY.LOCAL
 <domainname> = MYCOMPANY.LOCAL
[appdefaults]
pam =
{   debug = false  ticket_lifetime = 36000  renew_lifetime = 36000  forwardable
=
 true  krb4_convert = false }
```

6.6.2 Configuring an OWSM 12*c* Web Service and a Microsoft WCF/.NET 4.5 Client

The following instructions tell how to configure an OWSM 12*c* web service and a Microsoft WCF/.NET 4.5 client to implement Kerberos with message protection.

1. Create and deploy a web service application.

   For more information, see "Deploying Web Service Applications" in *Administering Web Services*.

2. Clone the following policy: `oracle/wss11_kerberos_token_with_message_protection_service_policy`.

   For more information, see "Cloning a Web Service Policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

3. Edit the policy settings to set Algorithm Suite to `Basic128Rsa15`.

4. Attach the policy to the web service.

   "Attaching Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

To Configure the Microsoft WCF/.NET 4.5 Client:

1. Create a user in AD to represent the host where the web service is hosted. By default the user account is created with RC4-HMAC encryption. For example, foobar with user name is `HTTP/foobar`.

2. Use the following ktpass command to create a keytab file on the Windows AD machine where the KDC is running:

   ```
   ktpass -princ HTTP/foobar@MYCOMPANY.LOCAL -pass Oracle123 -mapuser
   foobar -out foobar.keytab -ptype KRB5_NT_PRINCIPAL -kvno 4
   ```

   where `HTTP/foobar` is the SPN, mapped to a user "foobar". Do not set "/desonly or cyrpto as "des-cbc-crc". MYCOMPANY.LOCAL is the default Realm for the KDC and is available in the krb5.ini file. The pass password must match the password created during the user creation.

   Use FTP binary mode to move the generated keytab file to the machine where the SOA Composite web service is hosted.

3. `setSpn -L foobar`

   ```
   setSpn -A HTTP/foobar@MYCOMPANY.LOCAL foobar
   ```

   Only one SPN must be mapped to the user. If there are multiple SPNs mapped to the user, remove them using the command `setSpn -D <spname> <username>`.

   Use the following `setSpn` command to map the service principal to the user:

```
setSpn -A HTTP/foobar@MYCOMPANY.LOCAL foobar
```

```
setSpn -L foobar
```

Only one SPN must be mapped to the user. If there are multiple SPNs mapped to the user, remove them using the command `setSpn -D <spname> <username>`.

4. Use the Microsoft SvcUtil utility to create a client proxy and configuration file from the deployed web service.

   Add the files `generatedProxy.cs` and `app.config` by right clicking the application (in the Windows Explorer) and selecting **Add Existing Item**.

   In the endpoint element of the `app.config`, add an "identity" element with service principal name as "HTTP/foobar@MYCOMPANY.LOCAL" (the same value used for creating keytab).

```
<client>
        <endpoint address="http://host:port/HelloServicePort"
            binding="customBinding"
bindingConfiguration="NewHelloSoap12HttpPortBinding"
            contract="NewHello" name="HelloServicePort">
        <identity>
          <servicePrincipalName value ="HTTP/foobar@MYCOMPANY.LOCAL"/>
        </identity>
        </endpoint>

      </client>
```

   A sample binding is provided in Example 6–11, "Custom Binding".

   For more information, see
   http://msdn.microsoft.com/en-us/library/aa347733%28v=vs.110%29.aspx.

5. Run the client program.

   See the following *Custom Binding* sample:

```
<customBinding>
  <binding name="NewHelloSoap12HttpPortBinding">
      <!--Added by User: Begin-->
      <security defaultAlgorithmSuite="Basic128"
        authenticationMode="Kerberos"
        requireDerivedKeys="false" securityHeaderLayout="Lax"
        includeTimestamp="true"
        keyEntropyMode="CombinedEntropy"
        messageProtectionOrder="SignBeforeEncrypt"
        messageSecurityVersion="WSSecurity11WSTrustFebruary2005
        WSSecureConversationFebruary2005WSSecurityPolicy11BasicSecurity
          Profile10"
        requireSignatureConfirmation="true">
      <localClientSettings cacheCookies="true" detectReplays="true"
          replayCacheSize="900000" maxClockSkew="00:05:00"
          maxCookieCachingTime="Infinite"
          replayWindow="00:05:00"
          sessionKeyRenewalInterval="10:00:00"
          sessionKeyRolloverInterval="00:05:00"
          reconnectTransportOnFailure="true"
          timestampValidityDuration="00:05:00"
          cookieRenewalThresholdPercentage="60" />
                <localServiceSettings detectReplays="true"
          issuedCookieLifetime="10:00:00"
          maxStatefulNegotiations="128" replayCacheSize="900000"
          maxClockSkew="00:05:00"
          negotiationTimeout="00:01:00" replayWindow="00:05:00"
          inactivityTimeout="00:02:00"
          sessionKeyRenewalInterval="15:00:00"
          sessionKeyRolloverInterval="00:05:00"
          reconnectTransportOnFailure="true"
          maxPendingSessions="128"
          maxCachedCookies="1000"
          timestampValidityDuration="00:05:00" />
                  <secureConversationBootstrap />
                </security>
              <!--Added by User: End-->
                <textMessageEncoding maxReadPoolSize="64"
                   maxWritePoolSize="16"
                   messageVersion="Soap12" writeEncoding="utf-8">
                <readerQuotas maxDepth="32" maxStringContentLength="8192"
                   maxArrayLength="16384"
                   maxBytesPerRead="4096" maxNameTableCharCount="16384" />
                </textMessageEncoding>
              <!--Added by User: Begin-->
              <httpTransport manualAddressing="false"
                   maxBufferPoolSize="524288"
                   maxReceivedMessageSize="65536" allowCookies="false"
                   authenticationScheme="Anonymous"
                   bypassProxyOnLocal="false"
                   hostNameComparisonMode="StrongWildcard"
                   keepAliveEnabled="true" maxBufferSize="65536"
                   proxyAuthenticationScheme="Anonymous"
                   realm="" transferMode="Buffered"
                   unsafeConnectionNtlmAuthentication="false"
                   useDefaultWebProxy="true" />
                <!--Added by User: End-->
           </binding>
</customBinding>
```

6.7 Kerberos with Message Protection Using Derived Keys

This section describes how to implement Kerberos with message protection using derived keys in the following interoperability scenario:

- Configuring an OWSM 12c Web Service and a Microsoft WCF/.NET 4.5 Client

Before configuring the web service and client in the above scenario, follow the instructions in Section 6.7.1, "Configuration Prerequisites."

6.7.1 Configuration Prerequisites

The section below describes how to perform prerequisite configuration tasks for implementing Kerberos with message protection using derived keys.

To Configure Prerequisites for Interoperability:

1. Configure the Key Distribution Center (KDC) and Active Directory (AD).

   For more information, see the following topics:

   - "To Configure Windows Active Directory and Domain Controller" (the domain controller can serve as KDC) at http://download.oracle.com/docs/cd/E19316-01/820-3746/gisdn/index.html

- "Configuring Kerberos Tokens" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*

2. Set up the Kerberos configuration file `krb5.conf` in `c:\winnt` as shown in the following "Kerberos Configuration File" sample:

```
[logging]
default = c:\log\krb5libs.log
kdc = c:\log\krb5kdc.log
admin_server = c:\log\kadmind.log
[libdefaults]
default_realm = MYCOMPANY.LOCAL
dns_lookup_realm = false
dns_lookup_kdc = false
default_tkt_enctypes = rc4-hmac
default_tgs_enctypes = rc4-hmac
permitted_enctypes = rc4-hmac
kdc = hostname
[realms]
MYCOMPANY.LOCAL =
{ kdc = host:port  admin_server = host:port
  default_domain = <domainname>
}
 [domain_realm]
.<domainname> = MYCOMPANY.LOCAL
 <domainname> = MYCOMPANY.LOCAL
[appdefaults]
pam =
{   debug = false  ticket_lifetime = 36000  renew_lifetime = 36000  forwardable
=
 true  krb4_convert = false }
```

6.7.2 Configuring an OWSM 12*c* Web Service and a Microsoft WCF/.NET 4.5 Client

The following instructions tell how to configure an OWSM 12*c* web service and a Microsoft WCF/.NET 4.5 client to implement Kerberos with message protection.

To Configure the OWSM 12c Web Service:

1. Create and deploy a web service application.

   For more information, see "Deploying Web Service Applications" in *Administering Web Services*.

2. Clone the following policy: `wss11_kerberos_token_with_message_protection_basic128_service_policy.`

   For more information, see "Cloning a Web Service Policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

3. Edit the policy settings to enable the Derived Keys option.

4. Attach the policy to the web service.

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

To Configure the Microsoft WCF/.NET 4.5 Client:

1. Create a user in AD to represent the host where the web service is hosted. By default the user account is created with RC4-HMAC encryption. For example, foobar with user name as "HTTP/foobar".

2. Use the following ktpass command to create a keytab file on the Windows AD machine where the KDC is running:

   ```
   ktpass -princ HTTP/foobar@MYCOMPANY.LOCAL -pass Oracle123 -mapuser
   foobar -out foobar.keytab -ptype KRB5_NT_PRINCIPAL -kvno 4
   ```

   where HTTP/foobar is the SPN, mapped to a user "foobar". Do not set "/desonly or cyrpto as "des-cbc-crc". MYCOMPANY.LOCAL is the default Realm for the KDC and is available in the `krb5.ini` file. The pass password must match the password created during the user creation.

   Use FTP binary mode to move the generated keytab file to the machine where the SOA Composite web service is hosted.

3. Use the following `setSpn` command to map the service principal to the user:

   ```
   setSpn -A HTTP/foobar@MYCOMPANY.LOCAL foobar
   ```

   ```
   setSpn -L foobar
   ```

   Only one SPN must be mapped to the user. If there are multiple SPNs mapped to the user, remove them using the command `setSpn -D <spname> <username>`.

4. Use the Microsoft SvcUtil utility to create a client proxy and configuration file from the deployed web service.

   Add the files generatedProxy.cs and app.config by right clicking the application (in the Windows Explorer) and selecting **Add Existing Item**.

   In the endpoint element of the app.config, add an "identity" element with service principal name as "HTTP/foobar@MYCOMPANY.LOCAL" (the same value used for creating keytab).

   ```
   <client>
           <endpoint address="http://host:port/HelloServicePort"
               binding="customBinding"
   bindingConfiguration="NewHelloSoap12HttpPortBinding"
               contract="NewHello" name="HelloServicePort">
           <identity>
             <servicePrincipalName value ="HTTP/foobar@MYCOMPANY.LOCAL"/>
           </identity>
           </endpoint>
         </client>
   ```

   A sample binding is provided in Example 6–13, "Custom Binding".

5. Run the client program.

   See the following Custom Binding sample:

   ```
   <customBinding>
     <binding name="NewHelloSoap12HttpPortBinding">
       <!--Added by User: Begin-->
       <security defaultAlgorithmSuite="Basic128"
         authenticationMode="Kerberos"
         requireDerivedKeys="true" securityHeaderLayout="Lax"
         includeTimestamp="true"
         keyEntropyMode="CombinedEntropy"
         messageProtectionOrder="SignBeforeEncrypt"
         messageSecurityVersion="WSSecurity11WSTrustFebruary2005
         WSSecureConversationFebruary2005WSSecurityPolicy11BasicSecurity
         Profile10"
         requireSignatureConfirmation="true">
       <localClientSettings cacheCookies="true" detectReplays="true"
           replayCacheSize="900000" maxClockSkew="00:05:00"
   ```

```
        maxCookieCachingTime="Infinite"
        replayWindow="00:05:00"
        sessionKeyRenewalInterval="10:00:00"
        sessionKeyRolloverInterval="00:05:00"
        reconnectTransportOnFailure="true"
        timestampValidityDuration="00:05:00"
        cookieRenewalThresholdPercentage="60" />
      <localServiceSettings detectReplays="true"
        issuedCookieLifetime="10:00:00"
        maxStatefulNegotiations="128" replayCacheSize="900000"
        maxClockSkew="00:05:00"
        negotiationTimeout="00:01:00" replayWindow="00:05:00"
        inactivityTimeout="00:02:00"
        sessionKeyRenewalInterval="15:00:00"
        sessionKeyRolloverInterval="00:05:00"
        reconnectTransportOnFailure="true"
        maxPendingSessions="128"
        maxCachedCookies="1000"
        timestampValidityDuration="00:05:00" />
      <secureConversationBootstrap />
    </security>
  <!--Added by User: End-->
      <textMessageEncoding maxReadPoolSize="64"
        maxWritePoolSize="16"
        messageVersion="Soap12" writeEncoding="utf-8">
          <readerQuotas maxDepth="32" maxStringContentLength="8192"
            maxArrayLength="16384"
            maxBytesPerRead="4096" maxNameTableCharCount="16384" />
      </textMessageEncoding>
          <!--Added by User: Begin-->
      <httpTransport manualAddressing="false"
        maxBufferPoolSize="524288"
        maxReceivedMessageSize="65536" allowCookies="false"
        authenticationScheme="Anonymous"
        bypassProxyOnLocal="false"
        hostNameComparisonMode="StrongWildcard"
        keepAliveEnabled="true" maxBufferSize="65536"
        proxyAuthenticationScheme="Anonymous"
        realm="" transferMode="Buffered"
        unsafeConnectionNtlmAuthentication="false"
        useDefaultWebProxy="true" />
      <!--Added by User: End-->
  </binding>
</customBinding>
```

6.8 Kerberos with SPNEGO Negotiation

This section describes how to implement Kerberos with SPNEGO negotiation in the following interoperability scenario:

- Configuring an OWSM 12c Web Service and a Microsoft WCF/.NET 4.5 Client

6.8.1 Configuring an OWSM 12*c* Web Service and a Microsoft WCF/.NET 4.5 Client

The following instructions tell how to configure an OWSM 12*c* web service and a Microsoft WCF/.NET 4.5 client to implement Kerberos with SPNEGO negotiation.

To Configure the OWSM 12c Web Service:

1. Create and deploy a web service application.

   For more information, see "Deploying Web Service Applications" in *Administering Web Services*.

2. Create a policy that uses the `http_spnego_token_service_template` assertion template.

   For more information, see "Configuring Kerberos With SPNEGO Negotiation" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

3. Attach the policy to the web service.

To Configure the Microsoft WCF/.NET 4.5 Client:

1. Use the Microsoft SvcUtil utility to create a client proxy and configuration file from the deployed web service.

   For more information, see `http://msdn.microsoft.com/en-us/library/aa347733%28v=vs.110%29.aspx`.

2. Add the files generatedProxy.cs and app.config by right clicking the application (in the Windows Explorer) and selecting **Add Existing Item**.

3. Edit the `app.config` file as shown in Example 6–14, "app.config File".

   In this listing, note that the values of the contract and name attributes of the endpoint element are obtained from the generatedProxy.cs file.

4. Compile the client.

5. After attaching the OWSM policy to the deployed web service, run the client.

   See the following *app.config file* sample:

```
<configuration>
  <system.serviceModel>
    <bindings>
      <basicHttpBinding>
        <binding name="BPELProcessBinding">
          <security mode= "TransportCredentialOnly">
            <transport clientCredentialType="Windows"/>
          </security>
        </binding>
      </basicHttpBinding>
    </bindings>
    <client>
      <endpoint
          address="http://host:port/soa-infra/services/default/SOAProxy/bpelpro
cess_client_ep"
          binding="basicHttpBinding"
          bindingConfiguration="BPELProcessBinding"
          contract="BPELProcess" name="BPELProcess_pt"
        <identity>
          <servicePrincipalName value ="HTTP/host:port@MYCOMPANY.LOCAL" />
        </identity>
      </endpoint>
    </client>
  </system.serviceModel>
</configuration>
```

6.9 Kerberos with SPNEGO Negotiation and Credential Delegation

This section describes how to implement Kerberos with SPNEGO negotiation and credential delegation in the following interoperability scenario:

- Configuring an OWSM 12c Web Service and a Microsoft WCF/.NET 4.5 Client

6.9.1 Configuring an OWSM 12*c* Web Service and a Microsoft WCF/.NET 4.5 Client

The following instructions tell how to configure an OWSM 12*c* web service and a Microsoft WCF/.NET 4.5 client to implement Kerberos with SPNEGO negotiation and credential delegation.

To Configure the OWSM 12c Web Service:

1. Create and deploy a web service application.

   For more information, see "Deploying Web Service Applications" in *Administering Web Services*.

2. Create a policy that uses the `http_spnego_token_service_template` assertion template.

3. Attach the policy to the web service.

4. Set the value of the `credential.delegation` configuration setting to `true`.

   You can specify this information when attaching the policy, by overriding the policy configuration.

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

To Configure the Microsoft WCF/.NET 4.5 Client:

1. Use the Microsoft SvcUtil utility to create a client proxy and configuration file from the deployed web service.

   For more information, see `http://msdn.microsoft.com/en-us/library/aa347733%28v=vs.110%29.aspx`.

2. Add the files `generatedProxy.cs` and `app.config` by right clicking the application (in the Windows Explorer) and selecting **Add Existing Item**.

3. Edit the `app.config` file as shown in Example 6–15, "app.config File".

   In the example, note that the values of the contract and name attributes of the endpoint element are obtained from the `generatedProxy.cs` file.

4. Compile the client.

5. After attaching the OWSM policy to the deployed web service, run the client.

   See the following *app.config file* sample:

```
<configuration>
  <system.serviceModel>
    <bindings>
      <basicHttpBinding>
        <binding name="BPELProcess1Binding">
          <security mode= "TransportCredentialOnly">
            <transport clientCredentialType="Windows"/>
          </security>
        </binding>
      </basicHttpBinding>
    </bindings>
    <client>
      <endpoint
          address="http://host:port/soa-infra/services/default/SOAProxy/bpelpro
cess1_client_ep"
          binding="basicHttpBinding"
          bindingConfiguration="BPELProcess1Binding"
          contract="BPELProcess1" name="BPELProcess1_pt"
          behaviorConfiguration="CredentialDelegation">
        <identity>
          <servicePrincipalName value ="HTTP/host:port@MYCOMPANY.LOCAL" />
        </identity>
      </endpoint>
    </client>
    <behaviors>
      <endpointBehaviors>
        <behavior name="CredentialDelegation">
          <clientCredentials>
            <windows allowedImpersonationLevel="Delegation"
              allowNtlm="false"/>
          </clientCredentials>
        </behavior>
      </endpointBehaviors>
    </behaviors>
  </system.serviceModel>
</configuration>
```

6.10 WCF/.NET 4.5 Client with Microsoft Active Directory Federation Services 2.0 (ADFS 2.0) STS

This section tells how to secure a WCF/.NET 4.5 client with Microsoft Active Directory Federation Services 2.0 (ADFS 2.0) secure token service (STS), using the following policies:

- oracle/wss_sts_issued_saml_bearer_token_over_ssl_service_policy
- oracle/wss_saml_token_bearer_over_ssl_service_policy
- oracle/wss11_saml_or_username_token_with_message_protection_service_policy

> **Note:** The SAML sender vouches token is not supported in this use case.

The procedure described in this section are based on an ADFS 2.0 installation on Windows Server 2008 or Windows Server 2008 R2.

The section includes the following topics:

- Step 1: Install and Configure Active Directory Federation Services (ADFS) 2.0
- "Step 2: Configure OWSM to Trust SAML Assertions Issued by an ADFS 2.0 STS"
- Step 3: Configure Users in Oracle Internet Directory
- Step 4: Attach the Policy to the Web Service
- Step 5: Register the Web Service as a Relying Party in ADFS 2.0
- Step 6: Secure WCF/.NET 4.5 Client with ADFS 2.0

6.10.1 Step 1: Install and Configure Active Directory Federation Services (ADFS) 2.0

Install and configure ADFS 2.0 on a Windows Server 2008 or Windows Server 2008 R2 system.

Configure Active Directory and ADFS as shown.

To Install and Configure Active Directory Federation Services (ADFS) 2.0:

1. Set up the system in STS role.
2. Create and configure a self-signed server authentication certificate in Internet Information Services (IIS) and bind it to the default Web site using the IIS Manager console. When done, enable SSL server authentication.

   > **Note:** The ADFS 2.0 Setup Wizard automatically installs the web server (IIS) server role on the system.

3. Configure ADFS 2.0 as a stand-alone federation server.
4. Export the ADFS 2.0 token-signing certificate.

   For a self-signed certificate, select DER encoded binary X.509 (`.cer`).

   If the signing certificate is not self-signed, select Cryptographic Message Syntax Standard – PKCS 7 certificates (.p7b) and specify that all certificates in the certification path should be included.

5. Create users and include an e-mail address. You later enable the STS to send the e-mail address as the subject name id in the outgoing SAML assertions for the service.

For more information, see the following:

- "Windows Server 2008 R2 and Windows Server 2008" at `http://technet.microsoft.com/en-us/library/dd349801%28v=ws.10%29.aspx`.
- "Active Directory Services" at `http://technet.microsoft.com/en-us/library/dd578336%28v=ws.10%29.aspx`.
- "Active Directory Federation Services" at `http://technet.microsoft.com/library/cc772128%28WS.10%29.aspx`.
- "AD FS Step-by-Step Guide" at `http://technet.microsoft.com/en-us/library/cc731443%28v=ws.10%29.aspx`.
- "AD FS 2.0 Deployment Guide" at `http://technet.microsoft.com/en-us/library/dd807092%28v=ws.10%29.aspx`.

6.10.2 Step 2: Configure OWSM to Trust SAML Assertions Issued by an ADFS 2.0 STS

To Configure OWSM to trust the SAML assertions issued by an ADFS 2.0 STS:

1. Get the STS signing certificates you exported in "Step 1: Install and Configure Active Directory Federation Services (ADFS) 2.0.".

   For a `.p7b` file for a certificate chain, open the file in IE and copy each certificate in the chain in a `.cer` file.

2. Import the certificates into the location of the default keystore using keytool.

```
keytool -importcert -file <sts-signing-certs-file> -trustcacerts -alias
<alias> -keystore default-keystore.jks
```

For more information, see "keytool - Key and Certificate Management Tool" at http://docs.oracle.com/javase/6/docs/technotes/tools/windows/keytool.html

3. Add `http://`*`domain-name`*`/adfs/services/trust` as a SAML trusted issuer.
4. Add the Subject DN (as defined in RFC 2253) of the STS certificate in the Trusted STS Servers section. Use a string that conforms to RFC 2253, such as `CN=abc`. You can use the mechanism of your choice, such as keytool, to view the certificate and determine the Subject DN.

   For more information, refer to the following topics:

   - "Configuring SAML Trusted Issuers and DN Lists" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.
   - "keytool - Key and Certificate Management Tool" at http://docs.oracle.com/javase/6/docs/technotes/tools/windows/keytool.html.

6.10.3 Step 3: Configure Users in Oracle Internet Directory

For each user, configure the mail attribute to match the user e-mail address set in ADFS.

See Managing Directory Entries for Creating a User in *Oracle Fusion Middleware Administrator's Guide for Oracle Internet Directory* for information on configuring users in Oracle Internet Directory.

6.10.4 Step 4: Attach the Policy to the Web Service

Attach any of the following OWSM policies to the web service:

- `oracle/wss_sts_issued_saml_bearer_token_over_ssl_service_policy`
- `oracle/wss_saml_token_bearer_over_ssl_service_policy`
- `oracle/wss11_saml_or_username_token_with_message_protection_service_policy`

For more information, see:

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

6.10.5 Step 5: Register the Web Service as a Relying Party in ADFS 2.0

Configure ADFS 2.0 to issue the SAML assertion to the web service with the e-mail address or the name ID (SAM-Account-Name) as the subject name ID, as described below":

1. Add the web service as a relying party.

   For more information, see Create a Relying Party Trust Manually" at http://technet.microsoft.com/en-us/library/dd807108.aspx
2. Configure the claim rules for the service.

Enable the STS to send the e-mail address or the name ID as the `subject name id` in the outgoing SAML assertions for the service, create a chain of two claim rules with different templates.

To enable the STS to send the e-mail address or the name ID as the `subject name id` in the outgoing SAML assertions for the service, use the steps in this section to create a chain of two claim rules with different templates.

For more information, see the following topics:

- "Checklist: Creating Claim Rules for a Relying Party Trust" at `http://technet.microsoft.com/en-us/library/ee913578%28v=ws.10%29.aspx`
- "Create a Rule to Send LDAP Attributes as Claims" at `http://technet.microsoft.com/en-us/library/dd807115%28v=ws.10%29.aspx`

6.10.6 Step 6: Secure WCF/.NET 4.5 Client with ADFS 2.0

To Secure the WCF/.NET 4.5 client with ADFS 2.0:

1. Import the SSL server certificates for STS and the service into Windows.

   If the SSL server certificate for STS or the service is not issued from a trusted CA, or self-signed, then it needs to be imported with MMC tool, as described in step 1 in Table 6–8, " Configuring the Microsoft WCF/.NET 4.5 Client".

   For more information, see "How to: View Certificates with the MMC Snap-in" at `http://msdn.microsoft.com/en-us/library/ms788967.aspx`.

2. Create and configure the WCF./NET client, as described in steps 3 and 4, below.

   ADFS 2.0 STS supports multiple security and authentication mechanisms for token insurance. Each is exposed as a separate endpoint. For username/password authentication, two endpoints are provided:

   - `http://<adfs.domain>/adfs/services/trust/13/username` — This endpoint is for username token with message protection.
   - `https://<adfs.domain>/adfs/services/trust/13/usernamemixed` — This endpoint is for username token with transport protection (SSL).

   The WCF client uses the `https://<adfs.domain>/adfs/services/trust/13/usernamemixed` endpoint for username token on SSL to obtain the SAML bearer token for the service.

3. Generate the WCF Client with the service WSDL.

   For more information, see "How to: Create a Windows Communication Foundation Client" at `http://msdn.microsoft.com/en-us/library/ms733133(v=vs.110).aspx`.

4. Configure the client with `ws2007FederationHttpBinding`, and edit the `app.config` file, as follows.

   Example 6–16 shows a sample `app.config` for use with a web service using the following policies:

   - `oracle/wss_sts_issued_saml_bearer_token_over_ssl_service_policy`
   - `oracle/wss_saml_token_bearer_over_ssl_service_policy`
   - `oracle/wss11_saml_or_username_token_with_message_protection_service_policy`

For more information, see "WS 2007 Federation HTTP Binding" at http://msdn.microsoft.com/en-us/library/bb472490.aspx

5. Edit the `program.cs` file to make the service call.

   If not already present, create a `.cs` file in the project and name it `program.cs` (or any name of your choice.) Edit it to match the code in Example 6–17.

   In this example:

   `joe` is the username and *`eoj`* is the password used by the client to authenticate to the STS.

   `System.Net.ServicePointManager.ServerCertificateValidationCallback = ((sender, certificate, chain, sslPolicyErrors) => true);` has been added to validate the server side self-signed certificate. This is not required if the server certificate is issued by a trusted CA. If using a self-signed certificate for testing, add this method to validate the certificate on the client side.

   See the following *app.config File to Implement Varieties of SAML-Based Authentication* sample:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.serviceModel>
    <behaviors>
      <endpointBehaviors>
        <behavior name="secureBehaviour">
          <clientCredentials>
            <serviceCertificate>
              <defaultCertificate findValue="weblogic"
                  storeLocation="LocalMachine"
                  storeName="My"
                  x509FindType="FindBySubjectName"/>
            </serviceCertificate>
          </clientCredentials>
        </behavior>
      </endpointBehaviors>
    </behaviors>
    <bindings>
      <ws2007FederationHttpBinding>
        <binding name="JaxWsWss11SamlOrUsernameOrSamlBearerOverSSLSoapHttp">
          <security mode="TransportWithMessageCredential">
            <message negotiateServiceCredential="false"
                 algorithmSuite="Basic128"
                 issuedTokenType
="http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV1.1"
                 issuedKeyType="BearerKey">
              <issuer address
="https://domain-name/adfs/services/trust/13/usernamemixed"
                      binding ="ws2007HttpBinding"
                      bindingConfiguration="ADFSUsernameMixed"/>
             </message>
          </security>
        </binding>
      </ws2007FederationHttpBinding>
      <ws2007HttpBinding>
        <binding name="ADFSUsernameMixed">
          <security mode="TransportWithMessageCredential">
            <message clientCredentialType="UserName"
                     establishSecurityContext="false" />
          </security>
        </binding>
      </ws2007HttpBinding>
    </bindings>
    <client>
      <endpoint
address="https://host:8002/JaxWsWss11SamlOrUsernameOrSamlBearerOverSSL/JaxWsWss
11SamlOrUsernameOrSamlBearerOverSSLService"
          binding="ws2007FederationHttpBinding"

bindingConfiguration="JaxWsWss11SamlOrUsernameOrSamlBearerOverSSLSoapHttp"
          contract="JaxWsWss11SamlOrUsernameOrSamlBearerOverSSL"
          name="JaxWsWss11SamlOrUsernameOrSamlBearerOverSSLPort">
            <identity>
              <dns value="weblogic" />
            </identity>
          </endpoint>
    </client>
  </system.serviceModel>
</configuration>
```

See the following *pregram.cs File* sample:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.ServiceModel;

namespace Client
{
    class Program
    {
        static void Main(string[] args)
        {
            JaxWsWss11SamlOrUsernameOrSamlBearerOverSSLClient client =
               New JaxWsWss11SamlOrUsernameOrSamlBearerOverSSLClient();

            client.ClientCredentials.UserName.UserName = "joe";
            client.ClientCredentials.UserName.Password = "eoj";




System.Net.ServicePointManager.ServerCertificateValidationCallback =
               ((sender, certificate, chain, sslPolicyErrors) => true);


            Console.WriteLine(client.echo("Hello"));
            Console.Read();
        }

    }
}
```

7 Interoperability with Oracle Service Bus 10*g* Security Environments

This chapter describes interoperability of Oracle Web Services Manager (OWSM) with Oracle Service Bus 10*g* security environments.

This chapter includes the following sections:

- Overview of Interoperability with Oracle Service Bus 10g Security Environments
- Implementing a Username Token with WS-Security 1.0 Message Protection
- Implementing a SAML Sender Vouches Token with WS-Security 1.0 Message Protection
- Implementing a SAML or Username Token Over SSL
- Implementing Mutual Authentication with WS-Security 1.0 Message Protection

7.1 Overview of Interoperability with Oracle Service Bus 10*g* Security Environments

In Oracle Service Bus 10*g*, you attach policies to configure your security environment for inbound and outbound requests. Oracle Service Bus uses the underlying WebLogic security framework as building blocks for its security services. For information about configuring and attaching policies, see "Using WS-Policy in Oracle Service Bus Proxy and Business Services" in *Oracle Service Bus Security Guide* at `http://download.oracle.com/docs/cd/E13159_01/osb/docs10gr3/security/ws_policy.html`.

> **Note:** Ensure that you have downloaded and applied the TYBN and U37Z patches released for Oracle Service Bus 10.3 using the patch tool.

With OWSM 12*c*, you attach *policies* to web service endpoints. Each policy consists of one or more *assertions*, defined at the domain-level, that define the security requirements. A set of predefined policies and assertions are provided out-of-the-box.

Table 7–1 and Table 7–2 summarize the most common Oracle Service Bus 10*g* interoperability scenarios based on the following security requirements: authentication, message protection, and transport.

For more information about:

- OWSM predefined policies, see "Predefined Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

- Configuring and attaching OWSM 12*c* policies, see "Securing Web Services" and "Attaching Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.
- Configuring and attaching Oracle Service Bus 10*g* policies, see "Using WS-Policy in Oracle Service Bus Proxy and Business Services" in *Oracle Service Bus Security Guide* at `http://download.oracle.com/docs/cd/E13159_01/osb/docs10gr3/security/ws_policy.html`.

> **Note:** In the following scenarios, ensure that you are using a keystore with v3 certificates. By default, the JDK 1.5 keytool generates keystores with v1 certificates.
>
> In addition, ensure that the keys use the proper extensions, including DigitalSignature, Non_repudiation, Key_Encipherment, and Data_Encipherment.

Table 7–1 OWSM 12g Service Policy and Oracle Service Bus 10g Client Policy Interoperability

Identity Token	WS-Security Version	Message Protection	Transport Security	Service Policy	Client Policy
Username	1.0	Yes	No	`oracle/wss10_username_token_with_message_protection_service_policy`	`Encrypt.xml` `Sign.xml`
SAML	1.0	Yes	No	`oracle/wss10_saml_token_with_message_protection_service_policy`	`Encrypt.xml` `Sign.xml`
SAML or Username	1.0 and 1.1	No	Yes	`oracle/wss_saml_or_username_token_over_ssl_service_policy`	`Auth.xml`
Mutual Authentication	1.0	Yes	No	`oracle/wss10_x509_token_with_message_protection_service_policy`	`Encrypt.xml` `Sign.xml`

Table 7–2 Oracle Service Bus 10g Service Policy and OWSM 12c Client Policy Interoperability

Identity Token	WS-Security Version	Message Protection	Transport Security	Service Policy	Client Policy
Username	1.0	Yes	No	`Encrypt.xml` `Sign.xml`	`oracle/wss10_username_token_with_message_protection_client_policy`
SAML	1.0	Yes		`Encrypt.xml` `Sign.xml`	`oracle/wss10_saml_token_with_message_protection_client_policy`
Mutual Authentication	1.0	Yes	No	`Encrypt.xml` `Sign.xml`	`oracle/wss10_x509_token_with_message_protection_client_policy`

7.2 Implementing a Username Token with WS-Security 1.0 Message Protection

You can implement a username token with message WS-Security 1.0 protection. The following interoperability scenarios are supported:

- OWSM 12*c* web service with Oracle Service Bus 10*g* client
- Oracle Service Bus 10*g* web service with OWSM 12*c* client

For either scenario, you must perform prerequisite tasks for the WebLogic Server on which Oracle Service Bus is running. See "Overview of Prerequisites for Interoperabilty."

After completing the prerequisite tasks, see the detailed instructions for your supported scenario:

- "Configuring an OWSM 12c Web Service and an Oracle Service Bus 10g Client"
- "Configuring an Oracle Service Bus 10g Web Service and an OWSM 12c Client"

7.2.1 Overview of Prerequisites for Interoperabilty

Before you can implement a username token with WS-Security 1.0 message protection, you must complete a number of high-level tasks.

To configure prerequisites for interoperability:

1. Copy the `default-keystore.jks` and `trust.jks` files to your domain directory.

   The `default-keystore.jks` is used to store public and private keys for SOAP messages within the WebLogic Domain. The `trust.jks` is used to store private keys, digital certificates, and trusted certificate authority certificates that are used to establish and verify identity and trust in the WebLogic Server environment.

2. Invoke the WebLogic Administration Console.
3. Configure the Custom Identity and Custom Trust keystores.
4. Configure SSL.
5. Specify the private key alias, as required. For example: `oratest`.
6. Configure a credential mapping provider.

   Create a PKICredentialMapper and configure it as follows (leave all other values set to the defaults):

   a. Keystore Provider: N/A
   b. Keystore Type: jks
   c. Keystore File Name: default_keystore.jks
   d. Keystore Pass Phrase: <password>
   e. Confirm Keystore Pass Phrase: <password>

7. Restart Oracle WebLogic Server.
8. Invoke the OSB Console. For example:

   ```
   http://<host name>:<port number>/servicebus
   ```

9. Create a ServiceKeyProvider.
10. Specify Encryption Key and Digital Signature Key, as required.

    You must use different keys on the OWSM and Oracle Service Bus servers. You can use the same key for encryption and signing, if desired.

7.2.2 Configuring an OWSM 12*c* Web Service and an Oracle Service Bus 10*g* Client

You can implement a username token with message WS-Security 1.0 protection using an OWSM 12*c* web service and an Oracle Service Bus 10*g* client. Configure the web service, then configure the client.

To configure the OWSM 12c Web Service:

1. Clone the following policy: `wss10_username_token_with_message_protection_service_policy`.

   "Cloning a Web Service Policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*

2. Edit the policy settings, as follows:

   a. Set Encryption Key Reference Mechanism to `issuerserial`.

   b. Set Algorithm Suite to Basic128Rsa15 to match the algorithm suite used for Oracle Service Bus.

   c. Enable the Include Timestamp configuration setting.

   d. Set Is Encrypted to **false** for the Username token element only.

3. Attach the policy to the web service.

   "Attaching Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*

To configure the Oracle Service Bus 10g Client:

1. Clone the `Encrypt.xml` and `Sign.xml` policy files.

   For example, copy the files to `myEncrypt.xml` and `mySign.xml`. It is not recommended to edit the predefined policy files directly.

2. Edit the encryption algorithm in `myEncrypt.xml` file to prevent encryption compliance failure, as follows:

```
<wssp:Target>
   <wssp:EncryptionAlgorithm
     URI="http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
   <wssp:MessageParts
     Dialect="http://schemas.xmlsoap.org/2002/12/wsse#part">
      wsp:Body()
   </wssp:MessageParts>
</wssp:Target>
```

3. Edit the `mySign.xml` policy file attached to the Oracle Service Bus business service **request** only to sign the Username token by including the following target:

```
<wssp:Target>
   <wssp:DigestAlgorithm URI=
    "http://www.w3.org/2000/09/xmldsig#sha1" />
   <wssp:MessageParts Dialect=
    "http://www.bea.com/wls90/security/policy/wsee#part">
      wls:SecurityHeader(wsse:UsernameToken)
   </wssp:MessageParts>
</wssp:Target>
```

4. Edit the `mySign.xml` policy file attached to the Oracle Service Bus business service **response** only to specify that the security token is unsigned:

```
<wssp:Integrity SignToken="false">
```

Also, for SOA clients only, comment out the target for system headers, as shown:

```
<!-- wssp:Target>
  <wssp:DigestAlgorithm
   URI="http://www.w3.org/2000/09/xmldsig#sha1" />
  <wssp:MessageParts
   Dialect="http://www.bea.com/wls90/security/policy/wsee#part">
   wls:SystemHeaders()
  </wssp:MessageParts>
</wssp:Target -->
```

5. Invoke the web service method from the client.

Additional Information

"Cloning a Web Service Policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*

"Using WS-Policy in Oracle Service Bus Proxy and Business Services" in *Oracle Service Bus Security Guide* at http://download.oracle.com/docs/cd/E13159_01/osb/docs10gr3/security/ws_policy.html

7.2.3 Configuring an Oracle Service Bus 10*g* Web Service and an OWSM 12*c* Client

You can implement a username token with WS-Security 1.0 message protection using Oracle Service Bus 10*g* web service and an OWSM 12*c* client. Configure the web service, then configure the client.

To configure the Oracle Service Bus 10g Web Service:

1. Clone the `Encrypt.xml` and `Sign.xml` policy files.

   For example, copy the files to `myEncrypt.xml` and `mySign.xml`. It is not recommended to edit the predefined policy files directly.

2. Edit the encryption algorithm in the `myEncrypt.xml` file to prevent encryption compliance failure, as follows:

```
<wssp:Target>
   <wssp:EncryptionAlgorithm
     URI="http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
   <wssp:MessageParts
     Dialect="http://schemas.xmlsoap.org/2002/12/wsse#part">
      wsp:Body()
   </wssp:MessageParts>
</wssp:Target>
```

   For more information, see "Using WS-Policy in Oracle Service Bus Proxy and Business Services" in *Oracle Service Bus Security Guide* at http://download.oracle.com/docs/cd/E13159_01/osb/docs10gr3/security/ws_policy.html.

3. Edit the `mySign.xml` policy file attached to the proxy service **request** only to specify that the security token is unsigned:

```
<wssp:Integrity SignToken="false">
```

   Also, for SOA clients only, comment out the target for system headers, as shown:

```
<!-- wssp:Target>
  <wssp:DigestAlgorithm
   URI="http://www.w3.org/2000/09/xmldsig#sha1" />
  <wssp:MessageParts
   Dialect="http://www.bea.com/wls90/security/policy/wsee#part">
   wls:SystemHeaders()
  </wssp:MessageParts>
</wssp:Target -->
```

4. Create a web service application that invokes the Oracle Service Bus routing service.

To configure the OWSM 12c Client:

1. Clone the following policy: `wss10_username_token_with_message_protection_client_policy`.

   Edit the policy settings, as follows:

   a. Set Encryption Key Reference Mechanism to issuerserial.

   b. Set Recipient Encryption Key Reference Mechanism to issuerserial.

   c. Set Algorithm Suite to Basic128Rsa15 to match the algorithm suite used for Oracle Service Bus.

   d. Disable the Include Timestamp configuration setting.

   e. Set Is Encrypted to **false**.

   f. Leave the default configuration set for message signing and encryption.

   For more information, see "Cloning a Web Service Policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

2. Attach the policy to the web service client.

3. Invoke the web service from the client.

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

7.3 Implementing a SAML Sender Vouches Token with WS-Security 1.0 Message Protection

You can implement SAML sender vouches with WS-Security 1.0 message protection using an OWSM 12c Web Service with an Oracle Service Bus 10g Client.

The following are supported scenarios:

- OWSM 12c Web Service with Oracle Service Bus 10g Client
- Oracle Service Bus 10g Web Service with OWSM 12c Client

For either scenario, you must complete prerequisite tasks for the WebLogic Server on which Oracle Service Bus is running. See Table 7–8, " Configuration Prerequisites for Interoperability." After completing the prerequisite tasks, complete one of the following tasks depending upon your specific deployment:

- "Configuring an OWSM 12c Web Service and an Oracle Service Bus 10g Client" on page 7-8
- "Configuring an Oracle Service Bus 10g Web Service and an OWSM 12c Client" on page 7-9

To configure prerequisites for interoperability:

1. Copy the `default-keystore.jks` and `trust.jks` files to your domain directory.

The `default-keystore.jks` is used to store public and private keys for SOAP messages within the WebLogic Domain. The `trust.jks` is used to store private keys, digital certificates, and trusted certificate authority certificates that are used to establish and verify identity and trust in the WebLogic Server environment.

For more information, see "Configuring Keystores for Message Protection" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

2. Invoke the WebLogic Administration Console.

   For more information, see "Accessing Oracle WebLogic Administration Console" in *Administering Web Services*.

3. Create a SAMLIdentityAsserterV2 authentication provider.

   For more information, see "Configuring Authentication and Identity Assertion providers" in *Oracle WebLogic Server Administration Console Online Help*.

4. Restart WebLogic Server to add the new provider to the Administration Server's Runtime MBean server.

5. Select the authentication provider created in step 3.

6. Create and configure a SAML asserting party.

   Configure the SAML asserting party as follows (leave other values set to the defaults):

   - Profile: WSS/Sender-Vouches
   - Target URL: <OSB Proxy Service Endpoint URI>
   - Issuer URI: www.oracle.com

   Select the Enabled checkbox and click **Save**.

7. Create a SamlCredentialMapperV2 credential mapping provider.

   Select SamlCredentialMapperV2 from the drop-down list and name the credential mapper, for example, UC2_SamlCredentialMapperV2.

   For more information, see "SAML Identity Asserter V2: Create an Asserting Party" and "SAML Identity Asserter V2: Asserting Party: Configuration" in *Oracle WebLogic Server Administration Console Online Help*.

8. Restart WebLogic Server.

9. Configure the credential mapper as follows (leave other values set to the defaults):

   - Issuer URI: www.oracle.com

     Note: This value is specified in the policy file.

   - Name Qualifier: oracle.com

   For more information, see "Configure Credential Mapping Providers" in *Oracle WebLogic Server Administration Console Online Help*.

10. Create and configure a SAML relying party.

    Configure the SAML relying party as follows (leave other values set to the defaults):

    - Profile: WSS/Sender-Vouches
    - Target URL: <OWSM 12c Web Service>
    - Description: <your_description>

Select the Enabled checkbox and click **Save**.

For more information, see "SAML Credential Mapping Provider V2: Create a Relying Party" and "SAML Credential Mapping Provider V2: Relying Party: Configuration" in *Oracle WebLogic Server Administration Console Online Help*.

11. Restart WebLogic Server.

7.3.1 Configuring an OWSM 12*c* Web Service and an Oracle Service Bus 10*g* Client

You can configure implement SAML sender vouches with WS-Security 1.0 message protection using OWSM 12*c* web service and an Oracle Service Bus 10*g* client. Configure the web service, then configure the client.

To configure the OWSM 12*c* Web Service:

1. Clone the following policy: `oracle/wss10_saml_token_with_message_protection_service_policy`.

   For more information, see "Cloning a Web Service Policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

   a. Set Encryption Key Reference Mechanism to issuerserial.

   b. Set Algorithm Suite to Basic128Rsa15 to match the algorithm suite used for Oracle Service Bus.

   c. Set Is Encrypted to **false** for the Username token element only.

   d. Leave the default configuration set for message signing and encryption.

2. Attach the policy to the web service.

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

To Configure the Oracle Service Bus 10g Client:

1. Clone the `Encrypt.xml` and `Sign.xml` policy files.

   For example, to `myEncrypt.xml` and `mySign.xml`. It is not recommended to edit the predefined policy files directly.

2. Edit the encryption algorithm in the `myEncrypt.xml` file to prevent encryption compliance failure, as follows:

```
<wssp:Target>
   <wssp:EncryptionAlgorithm
     URI="http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
   <wssp:MessageParts
     Dialect="http://schemas.xmlsoap.org/2002/12/wsse#part">
      wsp:Body()
   </wssp:MessageParts>
</wssp:Target>
```

   For more information, see "Using WS-Policy in Oracle Service Bus Proxy and Business Services" in *Oracle Service Bus Security Guide* at http://download.oracle.com/docs/cd/E13159_01/osb/docs10gr3/security/ws_policy.html.

3. Edit the `mySign.xml` file attached to the Oracle Service Bus business service **request** only to sign the SAML assertion by including the following target:

```
<wssp:Target>
   <wssp:DigestAlgorithm URI="http://www.w3.org/2000/09/xmldsig#sha1" />
   <wssp:MessageParts Dialect=
    "http://www.bea.com/wls90/security/policy/wsee#part">
      wls:SecurityHeader(wsse:Assertion)
   </wssp:MessageParts>
</wssp:Target>
```

4. Edit the `mySign.xml` file attached to the Oracle Service Bus business service **response** only to specify that the security token is unsigned, as follows:

   ```
   <wssp:Integrity SignToken="false">
   ```

   Also, for SOA clients only, comment out the target for system headers, as shown:

   ```
   <!-- wssp:Target>
     <wssp:DigestAlgorithm
      URI="http://www.w3.org/2000/09/xmldsig#sha1" />
     <wssp:MessageParts
      Dialect="http://www.bea.com/wls90/security/policy/wsee#part">
      wls:SystemHeaders()
     </wssp:MessageParts>
   </wssp:Target -->
   ```

5. Use the custom SAML policy file shown in the following *Custom SAML Policy* sample:

   ```
   <?xml version="1.0"?>
   <wsp:Policy
      xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
      xmlns:wssp="http://www.bea.com/wls90/security/policy"
      xmlns:wsu="
   http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.
   xsd"
      xmlns:wls="http://www.bea.com/wls90/security/policy/wsee#part"
      wsu:Id="custom_saml">
      <wssp:Identity xmlns:wssp="http://www.bea.com/wls90/security/policy">
         <wssp:SupportedTokens>
            <wssp:SecurityToken
             TokenType=
   "http://docs.oasis-open.org/wss/2004/01/oasis-2004-01-saml-token-profile-1.0#SA
   MLAssertionID">
               <wssp:Claims>
                  <wssp:ConfirmationMethod>
                     sender-vouches
                  </wssp:ConfirmationMethod>
               </wssp:Claims>
            </wssp:SecurityToken>
         </wssp:SupportedTokens>
      </wssp:Identity>
      </wsp:Policy>
   ```

6. Invoke the web service from the client.

7.3.2 Configuring an Oracle Service Bus 10*g* Web Service and an OWSM 12*c* Client

You can implement SAMLsender vouches with WS-Security 1.0 message protection using an Oracle Service Bus 10*g* web service and an OWSM 12*c* client. Configure the web service, then configure the client.

To configure the Oracle Service Bus 10g Web Service:

1. Clone the `Encrypt.xml` and `Sign.xml` policy files.

For example, to `myEncrypt.xml` and `mySign.xml`. It is not recommended to edit the predefined policy files directly.

2. Edit the encryption algorithm in the `myEncrypt.xml` policy file to prevent encryption compliance failure, as follows:

```
<wssp:Target>
   <wssp:EncryptionAlgorithm
     URI="http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
   <wssp:MessageParts
     Dialect="http://schemas.xmlsoap.org/2002/12/wsse#part">
      wsp:Body()
   </wssp:MessageParts>
</wssp:Target>
```

For more information, see "Using WS-Policy in Oracle Service Bus Proxy and Business Services" in *Oracle Service Bus Security Guide* at `http://download.oracle.com/docs/cd/E13159_01/osb/docs10gr3/security/ws_policy.html`.

3. Edit the `mySign.xml` policy file attached to the proxy service **request** only to specify that the security token is unsigned:

```
<wssp:Integrity SignToken="false">
```

Also, for SOA clients only, comment out the target for system headers, as shown:

```
<!-- wssp:Target>
  <wssp:DigestAlgorithm
   URI="http://www.w3.org/2000/09/xmldsig#sha1" />
  <wssp:MessageParts
   Dialect="http://www.bea.com/wls90/security/policy/wsee#part">
   wls:SystemHeaders()
  </wssp:MessageParts>
</wssp:Target -->
```

4. Use the custom SAML policy file shown in the following *Custom SAML Policy* sample:

```
<?xml version="1.0"?>
<wsp:Policy
   xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
   xmlns:wssp="http://www.bea.com/wls90/security/policy"
   xmlns:wsu="
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.
xsd"
   xmlns:wls="http://www.bea.com/wls90/security/policy/wsee#part"
   wsu:Id="custom_saml">
   <wssp:Identity xmlns:wssp="http://www.bea.com/wls90/security/policy">
      <wssp:SupportedTokens>
         <wssp:SecurityToken
          TokenType=
"http://docs.oasis-open.org/wss/2004/01/oasis-2004-01-saml-token-profile-1.0#SA
MLAssertionID">
            <wssp:Claims>
               <wssp:ConfirmationMethod>
                  sender-vouches
               </wssp:ConfirmationMethod>
            </wssp:Claims>
         </wssp:SecurityToken>
      </wssp:SupportedTokens>
   </wssp:Identity>
   </wsp:Policy>
```

To configure the OWSM 12c client:

1. Clone the following policy: `wss10_saml_token_with_message_protection_client_policy`.

   Edit the policy settings, as follows:

   a. Set Encryption Key Reference Mechanism to issuerserial.

   b. Set Recipient Encryption Key Reference Mechanism to issuerserial.

   c. Set Algorithm Suite to Basic128Rsa15 to match the algorithm suite used for Oracle Service Bus.

   d. Disable the Include Timestamp configuration setting.

   e. Leave the default configuration set for message signing and encryption.

   For more information, see "Cloning a Web Service Policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

2. Attach the policy to the web service client.

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

3. Invoke the web service from the client.

7.4 Implementing a SAML or Username Token Over SSL

This section describes how to implement the SAML or username token over SSL policy, in the following interoperability scenario:

- Oracle Service Bus 10*g* client and OWSM 12*c* web service

> **Note:** The interoperability scenario described in this section also applies to the SAML Token Over SSL and Username Token Over SSL policies.

For either scenario, you must first perform prerequisite tasks for the WebLogic Server on which Oracle Service Bus is running, as described in the following sections:

- Configure the username token.
- Configure the SAML token.
- For SAML, perform the prerequisite steps for the WebLogic Server on which Oracle Service Bus is running.

Configuration instructions for the supported scenarios are in the following section:

- Section 7.4.1, "Configuring an OWSM 12c Web Service and an Oracle Service Bus 10g Client"

To configure SAML prerequisites for Interoperability:

1. Create a SamlCredentialMapperV2 credential mapping provider.

   Select SamlCredentialMapperV2 from the drop-down list and name the credential mapper; for example, UC2_SamlCredentialMapperV2.

   For more information, see "Configure Credential Mapping Providers" in *Oracle WebLogic Server Administration Console Online Help*.

2. Restart WebLogic Server.
3. Configure the credential mapper as follows (leave other values set to the defaults):
   - Issuer URI: www.oracle.com

     Note: This value is specified in the policy file.
   - Name Qualifier: oracle.com
4. Create and configure a SAML relying party.

   Configure the SAML relying party as follows (leave other values set to the defaults):
   - Profile: WSS/Sender-Vouches
   - Target URL: <OWSM 12c Web Service>
   - Description: <your_description>

   Select the Enabled checkbox and click **Save**.

   For more information, see "SAML Credential Mapping Provider V2: Create a Relying Party" and "SAML Credential Mapping Provider V2: Relying Party: Configuration" in *Oracle WebLogic Server Administration Console Online Help*
5. Restart WebLogic Server.

7.4.1 Configuring an OWSM 12*c* Web Service and an Oracle Service Bus 10*g* Client

You can implement the SAML or username token over SSL policy using an OWSM 12*c* web service and an Oracle Service Bus 10*g* client..

Both the SAML token client and the username token client are supported. Configure the web service, then configure the client.

To configure the OWSM 12c Web Service:

1. Configure the server for two-way SSL.
   - If the service policy is Username Token Over SSL, set **Two Way Client Cert Behavior** to "Client Certs Requested and Not Enforced."
   - If the service policy is SAML Token Over SSL, set **Two Way Client Cert Behavior** to "Client Certs Requested and Enforced."

   For more information, see "Configuring SSL on WebLogic Server (Two-Way)" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*
2. Clone the following policy: `wss_saml_or_username_token_over_ssl_service_policy`.
   - For `wss_username_token_over_ssl_service_policy`, disable the Create Element and Nonce configuration settings.
   - For `wss_saml_token_over_ssl_service_policy`, disable the Include Timestamp configuration setting.

   For more information, see "Cloning a Web Service Policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*
3. Use JDeveloper to create a simple SOA composite.
4. Attach the copy of the `wss_saml_or_username_token_over_ssl_service_policy` policy to the composite and deploy it.

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

To configure the Oracle Service Bus 10g client:

1. Configure the server for two-way SSL:

   - If the client policy is the equivalent of Username Token Over SSL, then set **Two Way Client Cert Behavior** to "Client Certs Requested and Not Enforced."
   - If the client policy is the equivalent of SAML Token Over SSL, then set **Two Way Client Cert Behavior** to "Client Certs Requested and Enforced."

   For more information, see "Configuring SSL on WebLogic Server (Two-Way)" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

2. In the Oracle Service Bus console, import the WSDL for the relying party.   Make sure that there is no policy attached. (Policy assertions are not allowed on this service.)

3. For SAML token, create a business service.

   a. Attach the policy to the request.

      Use the custom SAML policy file shown in the following *Custom SAML Policy* sample:

```
<?xml version="1.0"?>
<wsp:Policy
   xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
   xmlns:wssp="http://www.bea.com/wls90/security/policy"
   xmlns:wsu="
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
1.0.xsd"
   xmlns:wls="http://www.bea.com/wls90/security/policy/wsee#part"
   wsu:Id="custom_saml">
   <wssp:Identity xmlns:wssp="http://www.bea.com/wls90/security/policy">
      <wssp:SupportedTokens>
         <wssp:SecurityToken
          TokenType=
"http://docs.oasis-open.org/wss/2004/01/oasis-2004-01-saml-token-profile-1.
0#SAMLAssertionID">
            <wssp:Claims>
               <wssp:ConfirmationMethod>
                  sender-vouches
               </wssp:ConfirmationMethod>
            </wssp:Claims>
         </wssp:SecurityToken>
      </wssp:SupportedTokens>
   </wssp:Identity>
   </wsp:Policy>
```

   b. Change the WSDL from HTTP to HTTPS.

4. For username token, create a business service.

   a. Attach the `auth.xml` policy to the request.

   b. Change the WSDL from HTTP to HTTPS.

5. Create a proxy service, and create a route to the business service.

   In **HTTP Transport Configuration**, set Authentication to "basic."

On the **Security** page, associate the Service key provider. This is needed for Oracle Service Bus to send the client cert to SOA.

6. Run the proxy service from the Oracle Service Bus console with the username and password.

7.5 Implementing Mutual Authentication with WS-Security 1.0 Message Protection

This section describes how toYou can implement mutual authentication with WS-Security 1.0 message protection. The following scenarios are supported:

- OWSM 12c web service with Oracle Service Bus 10*g* client
- Oracle Service Bus 10*g* web service with OWSM 12*c* client

For either scenario, you must first perform prerequisite tasks:

- Configuration Prerequisites for OWSM
- Configuration Prerequisites for the Oracle WebLogic Server

After completing the prerequisite tasks, complete one of the following tasks depending upon your specific deployment:

- "Configuring an OWSM 12c Web Service and an Oracle Service Bus 10g Client" on page 7-16
- "Configuring an Oracle Service Bus 10g Web Service and an OWSM 12c Client" on page 7-18

To configure prerequisites for the Oracle WebLogic Server:

1. Copy the default-keystore.jks and trust.jks files to your domain directory.

   The `default-keystore.jks` is used to store public and private keys for SOAP messages within the WebLogic Domain. The `trust.jks` is used to store private keys, digital certificates, and trusted certificate authority certificates that are used to establish and verify identity and trust in the Oracle WebLogic Server environment.

   For more information, see "Configuring Keystores for Message Protection" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

2. Invoke the WebLogic Administration Console.

   For more information, see "Accessing Oracle WebLogic Administration Console" in *Administering Web Services*

3. Configure the Custom Identity and Custom Trust keystores.

   For more information, see "Configure keystores" in *Oracle WebLogic Server Administration Console Online Help*

4. Configure SSL.

   Specify the private key alias, as required. For example: `oratest`.

   For more information, see "Set up SSL" in *Oracle WebLogic Server Administration Console Online Help*

5. Configure a credential mapping provider.

6. Create a PKICredentialMapper and configure it as follows (leave all other values set to the defaults):

- Keystore Provider: N/A
- Keystore Type: jks
- Keystore File Name: default_keystore.jks
- Keystore Pass Phrase: <password>
- Confirm Keystore Pass Phrase: <password>

  For more information, see "Configure Credential Mapping Providers" in *Oracle WebLogic Server Administration Console Online Help*

7. Select the **Authentication** tab and configure as follows:
   - Click **DefaultIdentityAsserter** and add **X.509** to **Chosen** active types
   - Click **Provider Specific** and configure the following:
     - Default User Name Mapper Attribute Type: CN
     - Active Types: X.509
     - Use Default User Name Mapper: True

     For more information, see "Configure Authentication and Identity Assertion providers" in *Oracle WebLogic Server Administration Console Online Help*
8. Configure a token handler to specify that a client invoking a message-secured web service uses an X.509 certificate to establish their identity. In WebLogic Administration Console, navigate to the Web Service Security page of the domain and configure the inbound and outbound messages as follows:

   Note: Only username token with message protection or mutual authentication with message protection is available at any given time. Once you enable mutual authentication with message protection, username authentication will fail.
   - Click _SERVICE_BUS_INBOUND_WEB_SERVICE_SECURITY_MBEAN_ and select the Token Handler tab.
   - Click X.509 token handler and configure the following:
     - Name: UseX509ForIdentity
     - Value: True
   - Perform the same steps for the outbound Oracle Service Bus MBean: _SERVICE_BUS_OUTBOUND_WEB_SERVICE_SECURITY_MBEAN_
9. If the users are not added, add the Common Name (CN) user specified in the certificate.

   For more information, see "Create users" in *Oracle WebLogic Server Administration Console Online Help*.
10. Restart Oracle WebLogic Server.

To configure prerequisites for OWSM:

1. Configure authentication.

   Select the **Authentication** tab and configure as follows:
   - Click **DefaultIdentityAsserter** and add **X.509** to **Chosen** active types
   - Click **Provider Specific** and configure the following:
     - Default User Name Mapper Attribute Type: CN
     - Active Types: X.509

   - Use Default User Name Mapper: True

   For more information, see "Configure Authentication and Identity Assertion providers" in *Oracle WebLogic Server Administration Console Online Help*

2. If the users are not added, add the Common Name (CN) user specified in the certificate.

3. Restart Oracle WebLogic Server.

   For more information, see "Create users" in *Oracle WebLogic Server Administration Console Online Help*

7.5.1 Configuring an OWSM 12*c* Web Service and an Oracle Service Bus 10*g* Client

You can configure implement mutual authentication with WS-Security 1.0 message protection using an OWSM 12*c* web service and Oracle Service Bus 10*g* client. Configure the web service, then configure the client.

To configure the OWSM 12c Web Service:

1. Create and deploy a SOA composite.

2. Clone the following policy: `wss10_x509_token_with_message_protection_service_policy`.

   Edit the policy settings, as follows:

   a. Set Encryption Key Reference Mechanism to issuerserial.

   b. Set Algorithm Suite to Basic128Rsa15 to match the algorithm suite used for Oracle Service Bus.

   For more information, see "Cloning a Web Service Policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*

3. Attach the policy to the web service.

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

To configure Oracle Service Bus 10g Client:

1. Create an Oracle Service Bus business service.

2. Clone the `Encrypt.xml` and `Sign.xml` policy files.

   For example, copy the files to `myEncrypt.xml` and `mySign.xml`. It is not recommended to edit the predefined policy files directly.

3. Attach the X.509 policy shown in sample at the end of this procedure, to the Oracle Service Bus business service **request**.

4. Attach the `Sign.xml` policy file to the Oracle Service Bus business service **request**.

5. Edit the `myEncrypt.xml` policy, as shown in sample at the end of this procedure, and attach it to the Oracle Service Bus business service **request**.

   For more information, see "Using WS-Policy in Oracle Service Bus Proxy and Business Services" in *Oracle Service Bus Security Guide* at `http://download.oracle.com/docs/cd/E13159_01/osb/docs10gr3/security/ws_policy.html`.

6. Edit the `mySign.xml` policy file attached to the Oracle Service Bus business service **response** to specify that the security token is unsigned:

```
<wssp:Integrity SignToken="false">
```

Also, for SOA clients only, comment out the target for system headers, as shown in sample at the end of this procedure.

7. Attach the `myEncrypt.xml` policy file from Step 6 to the Oracle Service Bus business service **response**.

8. Create a ServiceKeyProvider.

9. Specify Encryption Key and Digital Signature Key, as required.

   You must use different keys on the OWSM and Oracle Service Bus servers. You can use the same key for encryption and signing, if desired.

10. Create a proxy service, and create a route to the business service.

    On the **Security** page, associate the Service key provider. This is needed for Oracle Service Bus to send the client certificate to SOA.

11. Run the proxy service from the Oracle Service Bus console.

    See the following *X.509 Policy* sample:

```
<wsp:Policy
  xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
  xmlns:wssp="http://www.bea.com/wls90/security/policy"

xmlns:s0="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-ut
ility-1.0.xsd"
  s0:Id="X509Auth">
        <wssp:Identity xmlns:wssp="http://www.bea.com/wls90/security/policy">
            <wssp:SupportedTokens>
                <wssp:SecurityToken
TokenType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-p
rofile-1.0#X509v3"/>
            </wssp:SupportedTokens>
        </wssp:Identity>
</wsp:Policy>
```

See the following *myEncrypt.xml Policy* sample:

```
<?xml version="1.0"?>
<wsp:Policy
 xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
 xmlns:wssp="http://www.bea.com/wls90/security/policy"

xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-u
tility-1.0.xsd"
 xmlns:wls="http://www.bea.com/wls90/security/policy/wsee#part"
 wsu:Id="X509Encrypt">
  <wssp:Confidentiality>
    <wssp:KeyWrappingAlgorithm URI="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
    <wssp:Target>
      <wssp:EncryptionAlgorithm
URI="http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
      <wssp:MessageParts
Dialect="http://schemas.xmlsoap.org/2002/12/wsse#part">wsp:Body()</wssp:Message
Parts>
    </wssp:Target>
    <wssp:KeyInfo/>
  </wssp:Confidentiality>
</wsp:Policy>
```

See the following *mySign Policy* sample:

```
  <?xml version="1.0"?>
<wsp:Policy
  xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
  xmlns:wssp="http://www.bea.com/wls90/security/policy"

xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd"
  xmlns:wls="http://www.bea.com/wls90/security/policy/wsee#part"
  wsu:Id="X509Sign">
  <wssp:Integrity SignToken="false">
    <wssp:SignatureAlgorithm URI="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    <wssp:CanonicalizationAlgorithm
 URI="http://www.w3.org/2001/10/xml-exc-c14n#"/>
    <!--wssp:Target>
      <wssp:DigestAlgorithm URI="http://www.w3.org/2000/09/xmldsig#sha1" />
      <wssp:MessageParts
 Dialect="http://www.bea.com/wls90/security/policy/wsee#part">
        wls:SystemHeaders()
      </wssp:MessageParts>
    </wssp:Target-->
    <wssp:Target>
      <wssp:DigestAlgorithm URI="http://www.w3.org/2000/09/xmldsig#sha1" />
      <wssp:MessageParts
 Dialect="http://www.bea.com/wls90/security/policy/wsee#part">
        wls:SecurityHeader(wsu:Timestamp)
      </wssp:MessageParts>
    </wssp:Target>
    <wssp:Target>
      <wssp:DigestAlgorithm URI="http://www.w3.org/2000/09/xmldsig#sha1" />
      <wssp:MessageParts
 Dialect="http://schemas.xmlsoap.org/2002/12/wsse#part">
      wsp:Body()
      </wssp:MessageParts>
    </wssp:Target>
   </wssp:Integrity>
  <wssp:MessageAge/>
</wsp:Policy>
```

7.5.2 Configuring an Oracle Service Bus 10*g* Web Service and an OWSM 12*c* Client

You can implement mutual authentication with WS-Security 1.0 message protection using Oracle Service Bus 10*g* web service and an OWSM 12*c* client. Configure the web service, then configure the client.

To configure the Oracle Service Bus 10g Web Service:

1. Create a Oracle Service Bus proxy service.
2. Clone the `Encrypt.xml` and `Sign.xml` policy files.

   For example, to `myEncrypt.xml` and `mySign.xml`. It is not recommended to edit the predefined policy files directly.
3. Attach the X.509 policy to the proxy service **request**.
4. Edit the `mySign.xml` policy file attached to the proxy service **request** and comment out the target for system headers and timestamp, as shown in the sample at the end of this procedure.

For more information, see "Using WS-Policy in Oracle Service Bus Proxy and Business Services" in *Oracle Service Bus Security Guide* at http://download.oracle.com/docs/cd/E13159_01/osb/docs10gr3/security/ws_policy.html.

5. Edit the encryption algorithm in the `myEncrypt.xml` file attached to the proxy service **request** as shown in the sample at the end of this procedure.
6. Attach `mySign.xml` and `myEncrypt.xml` policy files from the previous steps to the proxy service **response**.
7. Create a Service Key Provider.

mySign.xml Policy sample:

```
<?xml version="1.0"?>
<wsp:Policy
  xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
  xmlns:wssp="http://www.bea.com/wls90/security/policy"

xmlns:s0="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-ut
ility-1.0.xsd"
  s0:Id="X509SignRequest">
  <wssp:Integrity
 xmlns:wls="http://www.bea.com/wls90/security/policy/wsee#part"
 xmlns:wssp="http://www.bea.com/wls90/security/policy"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd">
  <wssp:SignatureAlgorithm URI="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
  <wssp:CanonicalizationAlgorithm URI="http://www.w3.org/2001/10/xml-exc-c14n#"
 />
  <!-- wssp:Target>
  <wssp:DigestAlgorithm URI="http://www.w3.org/2000/09/xmldsig#sha1" />
  <wssp:MessageParts
Dialect="http://www.bea.com/wls90/security/policy/wsee#part">wls:SystemHeaders
()</wssp:MessageParts>
  </wssp:Target -->
  <!-- wssp:Target>
  <wssp:DigestAlgorithm URI="http://www.w3.org/2000/09/xmldsig#sha1" />
  <wssp:MessageParts
Dialect="http://www.bea.com/wls90/security/policy/wsee#part">wls:SecurityHeader
(wsu:Timestamp)</wssp:MessageParts>
  </wssp:Target -->
  <wssp:Target>
  <wssp:DigestAlgorithm URI="http://www.w3.org/2000/09/xmldsig#sha1" />
  <wssp:MessageParts
Dialect="http://schemas.xmlsoap.org/2002/12/wsse#part">wsp:Body()</wssp:Message
Parts>
  </wssp:Target>
</wsp:Policy>
```

myEncrypt.xml sample:

```
<?xml version="1.0"?>
<wsp:Policy
 xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
 xmlns:wssp="http://www.bea.com/wls90/security/policy"

xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-u
tility-1.0.xsd"
 xmlns:wls="http://www.bea.com/wls90/security/policy/wsee#part"
 wsu:Id="X509Encrypt">
```

```
  <wssp:Confidentiality>
    <wssp:KeyWrappingAlgorithm URI="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
    <wssp:Target>
      <wssp:EncryptionAlgorithm
URI="http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
      <wssp:MessageParts
Dialect="http://schemas.xmlsoap.org/2002/12/wsse#part">wsp:Body()</wssp:Message
Parts>
    </wssp:Target>
    <wssp:KeyInfo/>
  </wssp:Confidentiality>

</wsp:Policy>
```

To configure the OWSM 12c client:

1. Clone the following policy: `wss10_x509_token_with_message_protection_client_policy`.

   In Fusion Middleware Control, edit the policy settings, as follows:

   a. Set Encryption Key Reference Mechanism to issuerserial.

   b. Set Recipient Encryption Key Reference Mechanism to issuerserial.

   c. Set Algorithm Suite to Basic128Rsa15 to match the algorithm suite used for Oracle Service Bus.

   d. Disable the Include Timestamp configuration setting.

   For more information, see "Cloning a Web Service Policy" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

2. In Fusion Middleware Control, specify keystore.recipient.alias in the client configuration. Ensure that the keystore.recipient.alias keys specified for the client exist as trusted certificate entry in the trust store configured for the web service.

3. Attach the policy to the web service client.

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

4. Invoke the web service from the client.

8 Interoperability with Axis 1.4 and WSS4J 1.5.8 Security Environments

This chapter describes interoperability of Oracle Web Services Manager (OWSM) with Axis 1.4 and WSS4J 1.5.8 security environments.

This chapter includes the following sections:

8.1 Overview of Interoperability With Axis 1.4 and WSS4J 1.5.8 Security Environments

In Axis 1.4 and WSS4J 1.5.8, you configure your security environment for inbound and outbound requests using handlers and deployment descriptors. For more information, see the *Axis Deployment Tutorial* at `http://ws.apache.org/wss4j/axis.html`.

With OWSM 12*c*, you attach *policies* to web service endpoints. Each policy consists of one or more *assertions*, defined at the domain-level, that define the security requirements. A set of predefined policies and assertions are provided out-of-the-box.

For more information about:

- OWSM predefined policies, see "Predefined Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.
- Configuring and attaching OWSM 12*c* policies, see "Securing Web Services" and "Attaching Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.
- Configuring and attaching policies on Axis and WSS4J, see the *Axis Deployment Tutorial* at `http://ws.apache.org/wss4j/axis.html`.

Table 8–1 OWSM 12c Service Policy and Axis WSS4J Client Policy Interoperability

Identity Token	WS-Security Version	Message Protection	Transport Security	Service Policy	Client Policy
Username	1.0	Yes	No	oracle/wss10_username_token_with_message_protection_service_policy	UsernameToken Timestamp Signature Encrypt
SAML	1.0	Yes	No	oracle/wss10_saml_token_with_message_protection_service_policy	SAMLTokenUnsigned Timestamp Signature Encrypt
Username	1.0 and 1.1	No	Yes	oracle/wss_username_token_over_ssl_service_policy	UsernameToken Timestamp
SAML	1.0 and 1.1	No	Yes	oracle/wss_saml_token_over_ssl_service_policy	SAMLTokenUnsigned Timestamp

Table 8–2 Axis WSS4J Service Policy and OWSM 12c Client Policy Interoperability

Identity Token	WS-Security Version	Message Protection	Transport Security	Service Policy	Client Policy
Username	1.0	Yes	No	UsernameToken Timestamp Signature Encrypt	oracle/wss10_username_token_with_message_protection_client_policy
SAML	1.0	Yes	No	SAMLTokenUnsigned Timestamp Signature Encrypt	oracle/wss10_saml_token_with_message_protection_client_policy
Username	1.0 and 1.1	No	Yes	Timestamp UsernameToken	oracle/wss_username_token_over_ssl_client_policy
SAML	1.0 and 1.1	No	Yes	Timestamp SAMLTokenUnsigned	oracle/wss_saml_token_over_ssl_client_policy

8.2 Creating Required Files for Interoperability With Axis and WSS4J

Perform the following steps to create the handler and property files that are required in each of the Axis and WSS4J interoperability scenarios:

1. Create and compile a password callback class, `PWCallback.java`, that can resolve passwords required by username and keystore aliases.

   The deployment descriptors defined in the following sections, contain username information, but not password information. As a best practice, you should not store sensitive information such as passwords in clear text within the deployment descriptor. To obtain the password, the Axis handler calls the password callback class. This mechanism is similar to JAAS. For more information, see the WSS4J documentation at `http://ws.apache.org/wss4j`.

2. Create the keystore properties file, `crypto.properties`, as shown below. Include this file in the classes directory.

   ```
   org.apache.ws.security.crypto.provider=org.apache.ws.security.components.crypto
   .Merlin
   org.apache.ws.security.crypto.merlin.keystore.type=jks
   org.apache.ws.security.crypto.merlin.keystore.password=welcome1
   org.apache.ws.security.crypto.merlin.file=default-keystore.jks
   ```

3. Create the `saml.properties` file, required for SAML interoperability scenarios only, as shown below.

```
org.apache.ws.security.saml.issuerClass=org.apache.ws.security.saml.SAMLIssuerI
mpl
org.apache.ws.security.saml.issuer.cryptoProp.file=crypto.properties
org.apache.ws.security.saml.issuer.key.name=orakey
org.apache.ws.security.saml.issuer.key.password=orakey
org.apache.ws.security.saml.issuer=www.oracle.com
org.apache.ws.security.saml.subjectNameId.name=weblogic
org.apache.ws.security.saml.authenticationMethod=password
org.apache.ws.security.saml.confirmationMethod=senderVouches
```

8.3 Username Token with Message Protection (WS-Security 1.0)

This section describes how to implement username token with message protection that conforms to the WS-Security 1.0 standard, in the following interoperability scenarios:

- Configuring an OWSM 12c Web Service and an Axis and WSS4J Client
- Configuring an Axis and WSS4J Web Service and an OWSM 12c Client

8.3.1 Configuring an OWSM 12*c* Web Service and an Axis and WSS4J Client

The following instructions tell how to configure an OWSM 12*c* web service and an Axis and WSS4J client to implement username token with message protection that conforms to the WS-Security 1.0 standard.

To configure the OWSM 12c Web Service:

1. Attach the following policy to the web service: `oracle/wss10_username_token_with_message_protection_service_policy`.

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

2. Deploy the web service.

To configure the Axis and WSS4J client:

1. Build your web service client proxy.

2. Create the password callback class, `PWCallback.java`, and keystore properties file, `crypto.properties`.

   For more information, see "Creating Required Files for Interoperability With Axis and WSS4J" on page 8-2

3. Include the keystore file (for example, `default-keystore.jks`) and `crypto.properties` file directly under the classes folder.

   Ensure that you are using keystore with v3 certificates. By default, the JDK 1.5 keytool generates keystores with v1 certificates.

4. Edit the deployment descriptor, `client_deploy.wsdd`, similar to the sample at the end of this procedure.

   In the example, the receiver decrypts, verifies, and validates the username token; the sender inserts a username token, timestamp, signs the body, username token, and timestamp, and encrypts the body and username token. As shown in the example, the encryption key transport is overridden to match the OWSM default requirements

5. Set the following property within the client code to use the deployment descriptor defined in the previous step.

```
System.setProperty("axis.ClientConfigFile", "client_deploy.wsdd");
```

6. Deploy the web service client.

   See the following *client_deploy.wsdd Deployment Descriptor* sample:

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
             xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
 <transport name="http"
  pivot="java:org.apache.axis.transport.http.HTTPSender"/>
  <globalConfiguration >
   <!-- wss10_username_token_with_message_protection -->
   <requestFlow>
     <handler type="java:org.apache.ws.axis.security.WSDoAllSender" >
       <parameter name="passwordCallbackClass"

value="com.oracle.xmlns.ConfigOverride_jws.CO_SOA.BPELProcess1.PWCallback"/>
       <parameter name="passwordType" value="PasswordText"/>
       <parameter name="user" value="weblogic"/>
       <parameter name="action" value="UsernameToken Timestamp Signature
Encrypt"/>
       <parameter name="encryptionKeyTransportAlgorithm"
        value="http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p"/>
       <parameter name="encryptionKeyIdentifier" value="DirectReference" />
       <parameter name="encryptionPropFile" value="crypto.properties" />
       <parameter name="encryptionUser" value="orakey" />
       <parameter name="encryptionParts" value=

"{Element}{http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-s
ecext-1.0.xsd}
     UsernameToken;{Content}{http://schemas.xmlsoap.org/soap/envelope/}Body" />
       <parameter name="signatureUser" value="orakey" />
       <parameter name="signaturePropFile" value="crypto.properties" />
       <parameter name="signatureKeyIdentifier" value="DirectReference" />
       <parameter name="signatureParts" value=

"{Element}{http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-s
ecext-1.0.xsd}
UsernameToken;{Element}{http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss
-wssecurity-utility-
1.0.xsd}
Timestamp;{Element}{http://schemas.xmlsoap.org/soap/envelope/}Body" />
     </handler>
   </requestFlow>
   <responseFlow>
     <handler type="java:org.apache.ws.axis.security.WSDoAllReceiver">
       <parameter name="passwordCallbackClass"
value="com.oracle.xmlns.ConfigOverride_jws.CO
_SOA.BPELProcess1.PWCallback"/>
       <parameter name="action" value="Timestamp Signature Encrypt" />
       <parameter name="signaturePropFile" value="crypto.properties" />
       <parameter name="decryptionPropFile" value="crypto.properties" />
       <parameter name="enableSignatureConfirmation"  value="false" />
     </handler>
   </responseFlow>
  </globalConfiguration >
</deployment>
```

8.3.2 Configuring an Axis and WSS4J Web Service and an OWSM 12*c* Client

The following instructions tell how to configure an Axis and WSS4J web service and an OWSM 12*c* client service to implement username token with message protection that conforms to the WS-Security 1.0 standard.

To configure the Axis and WSS4J Web Service:

1. Build your web service.
2. Create the password callback class, `PWCallback.java`, and keystore properties file, `crypto.properties`, as described in "Creating Required Files for Interoperability With Axis and WSS4J" on page 8-2.
3. Include the keystore file (for example, `default-keystore.jks`) and `crypto.properties` file directly under the classes folder.

   Ensure that you are using keystore with v3 certificates. By default, the JDK 1.5 keytool generates keystores with v1 certificates.
4. Edit the deployment descriptor, `server_deploy.wsdd`, as shown in the sample at the end of this procedure.

   In the example, the receiver decrypts, verifies, and validates the username token; the sender inserts a username token, timestamp, signs the body, username token, and timestamp, and encrypts the body and username token. As shown in the example, the encryption key transport is overridden to match the OWSM default requirements.

   Note: WSS4J enforces an order to the elements in the header. Ensure action ordering is updated in server_deploy.wsdd as shown in the sample at the end of this procedure.
5. Deploy the web service.

   See the following *server_deploy.wsdd Deployment Descriptor* sample

```
<ns1:service name="HelloWorld" provider="java:RPC" style="wrapped"
use="literal">
<!-- wss10_username_token_with_message_protection -->
<requestFlow>
   <handler type="java:org.apache.ws.axis.security.WSDoAllReceiver">
      <parameter name="passwordCallbackClass" value="PWCallback1"/>
      <parameter name="user" value="wss4j"/>
      <parameter name="action" value="Signature UsernameToken Timestamp
Encrypt"/>
      <parameter name="signaturePropFile" value="crypto.properties" />
      <parameter name="decryptionPropFile" value="crypto.properties" />
   </handler>
</requestFlow>
<responseFlow>
   <handler type="java:org.apache.ws.axis.security.WSDoAllSender" >
       <parameter name="passwordCallbackClass" value="PWCallback1"/>
       <parameter name="user" value="orakey"/>
       <parameter name="action" value="Timestamp Signature Encrypt"/>
       <parameter name="encryptionKeyTransportAlgorithm"
          value="http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p"/>
       <parameter name="signaturePropFile" value="crypto.properties" />
       <parameter name="signatureKeyIdentifier" value="DirectReference" />
       <parameter name="signatureParts"
value="{Element}{http://schemas.xmlsoap.org/soap/envelope/}Body;{Element}
{http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0
.xsd}Timestamp" />
```

```
        <parameter name="encryptionKeyIdentifier" value="DirectReference" />
    </handler>
</responseFlow>
</ns1:service>
```

To configure OWSM 12c client:

1. Attach the following policy to the web service: `oracle/wss10_username_token_with_message_protection_client_policy`.

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

2. For Java SE clients only, configure the web service client properties, as follows:

   Note: This step is not required for Java EE clients.

   ```
   myPort.setProperty(ClientConstants.WSS_KEYSTORE_TYPE,"JKS");
   myPort.setProperty(ClientConstants.WSS_KEYSTORE_LOCATION,
    "/keystore-path/default-keystore.jks");
   myPort.setProperty(ClientConstants.WSS_KEYSTORE_PASSWORD, "welcome1");
   myPort.setProperty(ClientConstants.WSS_RECIPIENT_KEY_ALIAS,"orakey");
   ...
   ```

   Where `setProperty` is defined as follows:

   ```
   public void setProperty(String name, String value) {
      ((Stub) _port)._setProperty(name, value);
   }
   ```

3. Deploy the web service client.

8.4 SAML Token with Message Protection (WS-Security 1.0)

This section describes how to implement SAML token with message protection that conforms to the WS-Security 1.0 standard, in the following interoperability scenarios:

- Configuring an OWSM 12c Web Service and an Axis an WSS4J Client
- Configuring an Axis and WSS4J Web Service and an OWSM 12c Client

8.4.1 Configuring an OWSM 12*c* Web Service and an Axis an WSS4J Client

The following instructions tell how to configure an OWSM 12*c* web service and an Axis and WSS4J client to implement SAML token with message protection that conforms to the WS-Security 1.0 standard.

To configure the OWSM 12c Web Service:

1. Attach the following policy to the web service: `oracle/wss10_saml_token_with_message_protection_service_policy`.

   "Attaching Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*

2. Deploy the web service.

To configure the Axis and WSS4J client:

1. Build your web service client proxy.
2. Create the password callback class, `PWCallback.java`, keystore properties file, `crypto.properties` file, and `saml.properties` file, as described in "Creating

Required Files for Interoperability With Axis and WSS4J" on page 8-2.

3. Include the keystore file (for example, `default-keystore.jks`) and `crypto.properties` file directly under the classes folder.

   Ensure that you are using keystore with v3 certificates. By default, the JDK 1.5 keytool generates keystores with v1 certificates.

4. Edit the deployment descriptor, `client_deploy.wsdd`, similar to the sample at the end of this procedure.

   In the example, the receiver decrypts, verifies, and validates the SAML token; the sender inserts a SAML token, timestamp, signs the body, SAML token, and timestamp, and encrypts the body. As shown in the example, the encryption key transport is overridden to match the OWSM default requirements.

5. Set the following property within the client code to use the deployment descriptor defined in the previous step.

```
System.setProperty("axis.ClientConfigFile", "client_deploy.wsdd");
```

6. Deploy the web service client.

   See the following *client_deploy.wsdd Deployment Descriptor* sample:

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
             xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
 <transport name="http"
  pivot="java:org.apache.axis.transport.http.HTTPSender"/>
  <globalConfiguration >
<!-- wss10_saml_token_with_message_protection -->
    <requestFlow>
      <handler type="java:org.apache.ws.axis.security.WSDoAllSender" >
        <parameter name="passwordCallbackClass"

value="com.oracle.xmlns.ConfigOverride_jws.CO_SOA.BPELProcess1.PWCallback"/>
        <parameter name="passwordType" value="PasswordText"/>
        <parameter name="user" value="weblogic"/>
        <parameter name="action" value="Timestamp Signature SAMLTokenSigned
Encrypt"/>
        <parameter name="samlPropFile" value="saml.properties"/>
        <parameter name="encryptionKeyTransportAlgorithm"
         value="http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p"/>
        <parameter name="encryptionKeyIdentifier" value="DirectReference" />
        <parameter name="encryptionPropFile" value="crypto.properties" />
        <parameter name="encryptionUser" value="orakey" />
        <parameter name="encryptionParts"
         value="{Content}{http://schemas.xmlsoap.org/soap/envelope/}Body" />
        <parameter name="signatureUser" value="orakey" />
        <parameter name="signaturePropFile" value="crypto.properties" />
        <parameter name="signatureKeyIdentifier" value="DirectReference" />
        <parameter name="signatureParts" value="{Element}

{http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0
.xsd}
          Timestamp;{Element}
          {http://schemas.xmlsoap.org/soap/envelope/}Body" />
      </handler>
    </requestFlow>
    <responseFlow>
      <handler type="java:org.apache.ws.axis.security.WSDoAllReceiver">
        <parameter name="passwordCallbackClass"
```

```
value="com.oracle.xmlns.ConfigOverride_jws.CO_SOA.BPELProcess1.PWCallback" />
        <parameter name="action" value="Timestamp Signature Encrypt" />
        <parameter name="signaturePropFile" value="crypto.properties" />
        <parameter name="decryptionPropFile" value="crypto.properties" />
        <parameter name="enableSignatureConfirmation" value="false" />
    </handler>
  </responseFlow>
  </globalConfiguration >
</deployment>
```

8.4.2 Configuring an Axis and WSS4J Web Service and an OWSM 12*c* Client

The following instructions tell how to configure an Axis and WSS4J web service and an OWSM 12*c* client to implement SAML token with message protection that conforms to the WS-Security 1.0 standard.

To configure the Axis and WSS4J Web Service:

1. Build your web service.
2. Create the password callback class, PWCallback.java, keystore properties file, crypto.properties file, and saml.properties file as described in "Creating Required Files for Interoperability With Axis and WSS4J" on page 8-2.
3. Include the keystore file (for example, default-keystore.jks) and crypto.properties file directly under the classes folder.

   Ensure that you are using keystore with v3 certificates. By default, the JDK 1.5 keytool generates keystores with v1 certificates.
4. Edit the deployment descriptor, server_deploy.wsdd, as shown in the sample at the end of this procedure.

   In the example, the receiver decrypts, verifies, and validates the SAML token; the sender inserts a SAML token, timestamp, signs the body, SAML token, and timestamp, and encrypts the body. As shown in the example, the encryption key transport is overridden to match the OWSM default requirements.

   Note: WSS4J enforces an order to the elements in the header. Ensure action ordering is updated in `server_deploy.wsdd` as shown in the sample at the end of this procedure.
5. Deploy the web service.

   See the following *server_deploy.wsdd Deployment Descriptor* sample:

```
<ns1:service name="HelloWorld" provider="java:RPC" style="wrapped"
use="literal">
<!-- wss10_username_token_with_message_protection -->
<requestFlow>
   <handler type="java:org.apache.ws.axis.security.WSDoAllReceiver">
      <parameter name="passwordCallbackClass" value="PWCallback1"/>
      <parameter name="user" value="wss4j"/>
      <parameter name="action" value="Signature SAMLTokenUnsigned Timestamp
Encrypt"/>
      <parameter name="signaturePropFile" value="crypto.properties" />
      <parameter name="decryptionPropFile" value="crypto.properties" />
   </handler>
</requestFlow>
<responseFlow>
   <handler type="java:org.apache.ws.axis.security.WSDoAllSender" >
       <parameter name="passwordCallbackClass" value="PWCallback1"/>
       <parameter name="user" value="orakey"/>
       <parameter name="action" value="Timestamp Signature Encrypt"/>
       <parameter name="encryptionKeyTransportAlgorithm"
          value="http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p"/>
       <parameter name="signaturePropFile" value="crypto.properties" />
       <parameter name="signatureKeyIdentifier" value="DirectReference" />
       <parameter name="signatureParts"
value="{Element}{http://schemas.xmlsoap.org/soap/envelope/}Body;{Element}
{http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0
.xsd}Timestamp" />
       <parameter name="encryptionKeyIdentifier" value="DirectReference" />
   </handler>
</responseFlow>
</ns1:service>
```

To configure the OWSM 12c client:

1. Attach the following policy to the web service: `oracle/wss10_saml_token_with_message_protection_client_policy`.

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

2. For JSE clients only, configure the web service client properties, as follows:

   Note: This step is not required for Java EE clients.

   ```
   myPort.setProperty(ClientConstants.WSS_KEYSTORE_TYPE,"JKS");
   myPort.setProperty(ClientConstants.WSS_KEYSTORE_LOCATION,
    "/keystore-path/default-keystore.jks");
   myPort.setProperty(ClientConstants.WSS_KEYSTORE_PASSWORD, "welcome1");
   myPort.setProperty(ClientConstants.WSS_RECIPIENT_KEY_ALIAS,"orakey");
   ...
   ```

   Where `setProperty` is defined as follows:

   ```
   public void setProperty(String name, String value) {
      ((Stub) _port)._setProperty(name, value);
   }
   ```

3. Deploy the web service client.

8.5 Username Token Over SSL

This section describes how to implement username token over SSL, in the following interoperability scenarios:

- Configuring an OWSM 12c Web Service and an Axis and WSS4J Client
- Configuring an Axis and WSS4J Web Service and an OWSM 12c Client

8.5.1 Configuring an OWSM 12*c* Web Service and an Axis and WSS4J Client

The following instructions tell how to configure an OWSM 12*g* web service and an Axis and WSS4J client to implement username token over SSL.

To configure the OWSM 12c Web Service:

1. Configure the server for SSL.

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

2. Attach the following policy to the web service: `oracle/wss_username_token_over_ssl_service_policy`.

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

3. Deploy the web service.

To configure the Axis and WSS4J client:

1. Build your web service client proxy.
2. Create the password callback class, PWCallback.java, and keystore properties file, crypto.properties, as described in "Creating Required Files for Interoperability With Axis and WSS4J" on page 8-2.
3. Edit the deployment descriptor, client_deploy.wsdd, similar the example below. In the example, the receiver validates the username token and timestamp; the sender inserts a timestamp.

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
             xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
 <transport name="http"
  pivot="java:org.apache.axis.transport.http.HTTPSender"/>
<globalConfiguration >
<!-- wss_username_token -->
<requestFlow >
   <handler type="java:org.apache.ws.axis.security.WSDoAllSender" >
      <parameter name="action" value="UsernameToken Timestamp"/>
      <parameter name="user" value="weblogic"/>
      <parameter name="passwordCallbackClass"

value="com.oracle.xmlns.ConfigOverride_jws.CO_SOA.BPELProcess1.PWCallback"/>
      <parameter name="passwordType" value="PasswordText"/>
  </handler>
</requestFlow >
</globalConfiguration >
</deployment>
```

4. Set the following property within the client code to use the deployment descriptor defined in the previous step.

```
System.setProperty("axis.ClientConfigFile", "client_deploy.wsdd");
```

5. Deploy the web service client.

8.5.2 Configuring an Axis and WSS4J Web Service and an OWSM 12*c* Client

The following instructions tell how to configure an Axis and WSS4J web service and an OWSM 12*c* client to implement username token over SSL.

To configure the Axis and WSS4J Web Service:

1. Configure the server for SSL.
2. Build your web service.
3. Create the password callback class, PWCallback.java, and crypto.properties file, as described in "Creating Required Files for Interoperability With Axis and WSS4J" on page 8-2.

4. Edit the deployment descriptor, server_deploy.wsdd, similar to the example below. In the example, the receiver validates the username token and the timestamp; the sender inserts a timestamp.

```
<ns1:service name="HelloWorld" provider="java:RPC" style="wrapped"
 use="literal">
<!-- wss_username_token_over_ssl -->
  <requestFlow>
    <handler type="java:org.apache.ws.axis.security.WSDoAllReceiver">
      <parameter name="passwordCallbackClass" value="PWCallback1"/>
      <parameter name="action" value="Timestamp UsernameToken"/>
    </handler>
  </requestFlow>
  <responseFlow>
     <handler type="java:org.apache.ws.axis.security.WSDoAllSender" >
      <parameter name="action" value="Timestamp"/>
      </handler>
  </responseFlow>
</ns1:service>
```

5. Deploy the web service.

To configure OWSM 12c client:

1. Attach the following policy to the web service client: `wss_username_token_over_ssl_client_policy`.

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

2. For JSE clients only, configure the web service client properties, as shown below. The username and password must be set by the client for generating the username token.

   Note: This step is not required for Java EE clients.

   ```
   myPort.setUsername("wss4j");
   myPort.setPassword("security"););
   ```

3. Deploy the web service client.

   When running the client, include the following client system property, where *default-keystore.jks* specifies the keystore that contains the certificate corresponding to the server certificate.

   ```
   -Djavax.net.ssl.trustStore=default-keystore.jks
   ```

8.6 SAML Token (Sender Vouches) Over SSL

This section describes how to implement SAML token (sender vouches) over SSL, in the following interoperability scenarios:

- Configuring an OWSM 12c Web Service and an Axis and WSS4J Client
- Configuring an Axis and WSS4J Web Service and an OWSM 12c Client

8.6.1 Configuring an OWSM 12*c* Web Service and an Axis and WSS4J Client

The following instructions tell how to configure an OWSM 12*c* web service and an Axis and WSS4J client to implement SAML token (sender vouches) over SSL.

To configure the OWSM 12c Web Service:

1. Configure the server for SSL.

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

2. Attach the following policy to the web service: `wss_saml_token_over_ssl_service_policy`.

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

3. Deploy the web service.

To configure the Axis and WSS4J client:

1. Build your web service client proxy.

2. Create the password callback class, PWCallback.java; keystore properties file, crypto.properties; and SAML properties file, saml.properties, as described in "Creating Required Files for Interoperability With Axis and WSS4J" on page 8-2.

3. Edit the deployment descriptor, client_deploy.wsdd, similar the example below. In the example, the receiver validates the SAML token and timestamp; the sender inserts a timestamp.

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
             xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
<transport name="http"
  pivot="java:org.apache.axis.transport.http.HTTPSender"/>
 <globalConfiguration >
<!-- wss_saml_token -->
<requestFlow >
   <handler type="java:org.apache.ws.axis.security.WSDoAllSender" >
      <parameter name="action" value="SAMLTokenSigned Timestamp"/>
      <parameter name="samlPropFile" value="saml.properties"/>
      <parameter name="user" value="weblogic"/>
      <parameter name="passwordCallbackClass"
 value="com.oracle.xmlns.ConfigOverride_jws.CO_SOA.BPELProcess1.PWCallback"/>
      <parameter name="passwordType" value="PasswordText"/>
      <parameter name="signatureUser" value="orakey" />
      <parameter name="signatureKeyIdentifier" value="DirectReference" />
      <parameter name="signaturePropFile" value="crypto.properties" />
   </handler>
</requestFlow >
</globalConfiguration >
</deployment>
```

4. Set the following property within the client code to use the deployment descriptor defined in the previous step.

   ```
   System.setProperty("axis.ClientConfigFile", "client_deploy.wsdd");
   ```

5. Deploy the web service client.

8.6.2 Configuring an Axis and WSS4J Web Service and an OWSM 12*c* Client

The following instructions tell how to configure an Axis and WSS4J web service and an OWSM 12*c* client to implement SAML token (sender vouches) over SSL.

To configure the Axis and WSS4J Web Service:

1. Configure the server for SSL.

2. Build your web service.

3. Create the password callback class, PWCallback.java, and crypto.properties file, as described in "Creating Required Files for Interoperability With Axis and WSS4J" on page 8-2.

4. Edit the deployment descriptor, server_deploy.wsdd, similar to the example below.

   In the example, the receiver validates the SAML token and the timestamp; the sender inserts a timestamp.

   ```
   <ns1:service name="HelloWorld" provider="java:RPC" style="wrapped"
    use="literal">
   <!-- wss_saml_token_over_ssl -->
   <requestFlow>
      <handler type="java:org.apache.ws.axis.security.WSDoAllReceiver">
         <parameter name="passwordCallbackClass" value="PWCallback1"/>
         <parameter name="action" value="Timestamp SAMLTokenUnsigned"/>
      </handler>
   </requestFlow>
   <responseFlow>
      <handler type="java:org.apache.ws.axis.security.WSDoAllSender" >
         <parameter name="action" value="Timestamp"/>
      </handler>
   </responseFlow>
   </ns1:service>
   ```

5. Deploy the web service.

To configure the OWSM 12c client:

1. Attach the following policy to the web service client: `wss_saml_token_over_ssl_client_policy`.

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

2. For JSE clients, configure the web service client properties, as shown below. The username must be set by the client for generating the SAML assertion.

   ```
   myPort.setUsername("wss4j");
   ```

   Note: This step is not required for Java EE clients.

3. Deploy the web service client.

   When running the client, include the following client system property, where *default-keystore.jks* specifies the keystore that contains the certificate corresponding to the server certificate.

   ```
   -Djavax.net.ssl.trustStore=default-keystore.jks
   ```

9

Interoperability with Oracle GlassFish Server Release 3.0.1

This chapter describes interoperability of Oracle Web Services Manager (OWSM) with Oracle Glassfish Server Release 3.0.1.

This chapter includes the following sections:

- Overview of Interoperability With Oracle GlassFish Security Environments
- Username Token with Message Protection (WS-Security 1.1)
- SAML Token (Sender Vouches) with Message Protection (WS-Security 1.1)

9.1 Overview of Interoperability With Oracle GlassFish Security Environments

Oracle GlassFish Server Release 3.0.1 is an open source application server for the Java EE platform. Metro is an open-source web service stack that is a part of Oracle GlassFish Server.

With OWSM 12*c*, you attach *policies* to web service endpoints. Each policy consists of one or more *assertions*, defined at the domain-level, that define the security requirements. A set of predefined policies and assertions are provided out-of-the-box.

For more information about:

- OWSM predefined policies, see "Predefined Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.
- Configuring and attaching OWSM 12*c* policies, see "Securing Web Services" and "Attaching Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.
- Configuring Oracle GlassFish, see `http://download.oracle.com/docs/cd/E18930_01/index.html`.
- Configuring Metro web services, see `http://metro.java.net/guide/`

Table 9–1 and Table 9–2 summarize the most common GlassFish Server interoperability scenarios based on the following security requirements: authentication, message protection, and transport.

Table 9–1 OWSM 11g Service Policy and GlassFish Client Interoperability

Identity Token	WS-Security Version	Message Protection	Transport Security	Service Policy	Client Policy
SAML	1.1	Yes	No	`oracle/wss11_saml_token_with_message_protection_service_policy`	See "Configuring an OWSM 12c Web Service and a GlassFish Client" on page 9-5

Table 9–2 GlassFish Service and OWSM 11g Client Policy Interoperability

Identity Token	WS-Security Version	Message Protection	Transport Security	Service Policy	Client Policy
SAML	1.1	Yes	No	See "Configuring a GlassFish Web Service and an OWSM 12c Client" on page 9-6	`oracle/wss11_saml_token_with_message_protection_client_policy`

9.2 Username Token with Message Protection (WS-Security 1.1)

This section describes how to implement username token with message protection that conforms to the WS-Security 1.1 standard, in the following interoperability scenarios:

- Configuring an OWSM 12c Web Service and a GlassFish Client
- Configuring a GlassFish Web Service and an OWSM 12c Client

9.2.1 Configuring an OWSM 12*c* Web Service and a GlassFish Client

The following instructions tell how to configure an OWSM 12*c* web service and a GlassFish client to implement username token with message protection that conforms to the WS-Security 1.1 standard.

To Configure Prerequisites for Interoperability:

1. Create a `default-keystore.jks` file with the following command:

   ```
   $JAVA_HOME/bin/keytool -genkeypair -alias orakey -keypass welcome -keyalg RSA
    -dname "CN=orakey, O=oracle C=us" -keystore default-keystore.jks -storepass
    welcome
   ```
2. Copy `default-keystore.jks` to the domain's `fmwconfig` directory.
3. Create a file user in GlassFish with the following command:

   ```
   $<GLASSFISHV3_HOME>/glassfish/bin/asadmin create-file-user
   ```
   For more information, see

   `http://download.oracle.com/docs/cd/E18930_01/html/821-2433/create-file-user-1.html`
4. Import `orakey` from `default-keystore.jks` into GlassFish keystore and truststore. These are located in the directory `<domain-dir>/config`

   ```
   $JAVA_HOME/bin/keytool -importkeystore -srckeystore
    <path-to>/default-keystore.jks -destkeystore
    <path-to-gf-domain>/config/cacerts.jks -srcalias  orakey -destalias orakey
    -srckeypass welcome -destkeypass changeit
   ```
5. Copy `jps-config.xml` and `default-keystore.jks` from the domain's `fmwconfig` directory into a local folder.

To Configure OWSM 12c Web Service:

1. Create a Web service.

2. Attach the following policy to the Web service: `oracle/wss11_username_token_with_message_protection_service_policy`.

   For more information, see “Attaching Policies” in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

To Configure GlassFish/Metro Client:

1. Using NetBeans, create a Metro client by selecting **New Project > Java > Java Application**. Provide a project name and location and select Finish.
2. Right-click on the project. Select **New > Web service Client**. Follow the wizard and provide WSDL URL for service deployed in WebLogic.
3. Select **Edit Web Services Attributes**.
4. Check **Use Development Defaults** to include Metro libraries into the project.
5. Uncheck **Use Development Defaults**. Provide username subject and password.
6. For a Metro SE client:
   a. Edit the truststore configuration. Select the same `default-keystore.jks` created in Table 9–3, " Configuration Prerequisites for Interoperability".
   b. Drag and drop the Web service operation into main class, main method.
   c. Right click on the project and choose run to execute the project.
7. For a Metro Java EE client:
   a. Drag and drop the Web service operation into EJB or Servlet to invoke.
   b. Deploy the application into GlassFish and invoke the Web service.

9.2.2 Configuring a GlassFish Web Service and an OWSM 12*c* Client

The following instructions tell how to configure a GlassFish web service and an OWSM 12*c* client to implement username token with message protection that conforms to the WS-Security 1.1 standard.

To Configure Prerequisites for Interoperability

1. Create a `default-keystore.jks` file with the following command:

```
$JAVA_HOME/bin/keytool -genkeypair -alias orakey -keypass welcome -keyalg RSA
 -dname "CN=orakey, O=oracle C=us" -keystore default-keystore.jks -storepass
 welcome
```

2. Copy `default-keystore.jks` to the domain's `fmwconfig` directory.
3. Save the credentials in credential store using WLST commands. For example:

```
$<ORACLE_HOME>/common/bin/wlst.sh
> connect()
> createCred(map="oracle.wsm.security", key="keystore-csf-key",
 user="keystore", password="welcome")
> createCred(map="oracle.wsm.security", key="sign-csf-key", user="orakey",
 password="welcome")
> createCred(map="oracle.wsm.security", key="enc-csf-key", user="orakey",
 password="welcome")
>createCred(map="oracle.wsm.security", key="glassfish.credentials" ,
 user="wlsUser" , password="welcome1" , description="Glassfish user
 credentials");
```

   A file `cwallet.sso` is created in the directory `DOMAIN_HOME/config/fmwconfig`

4. Create a file user in GlassFish with the following command:

```
$<GLASSFISHV3_HOME>/glassfish/bin/asadmin create-file-user
```

For more information, see http://download.oracle.com/docs/cd/E18930_01/html/821-2433/create-file-user-1.html

5. Import `orakey` from `default-keystore.jks` into GlassFish keystore and truststore. These are located in the directory `<domain-dir>/config`.

   ```
   $JAVA_HOME/bin/keytool -importkeystore -srckeystore
    <path-to>/default-keystore.jks -destkeystore
    <path-to-gf-domain>/config/keystore.jks -srcalias  orakey -destalias orakey
    -srckeypass welcome -destkeypass changeit
   ```

6. Copy `cwallet.sso`, `jps-config.xml` and `default-keystore.jks` from the domain's `fmwconfig` directory into a local folder.

To Configure the GlassFish/Metro Web Service:

1. Create a Metro Web service. For more information, see http://metro.java.net/guide/ch02.html#using_metro-developing_with_nb.
2. Configure the appropriate security mechanism. for more information, see `http://metro.java.net/guide/ch12.html#ahicu`.

To Configure the OWSM 11g Client:

1. Using JDeveloper, create a Web service proxy for the GlassFish service. Select the policy `oracle/wss11_username_token_with_message_protection_client_policy` in the wizard.
2. Set the `csf-key` to `glassfish.credentials` in the Override Properties option for the Web service proxy.
3. In the Web service proxy main class, set the system property of `oracle.security.jps.config` to `jps-config.xml` from Step 6 of Table 9–5, " Configuration Prerequisites for Interoperability".

> **Note:** If you are using:
>
> - Oracle Service Bus business service, set the property overrides to glassfish.credentials in the Security page. For more information, see "Policy Overrides" in *Oracle Fusion Middleware Developer's Guide for Oracle Service Bus* at `http://docs.oracle.com/docs/html/E15866_01/owsm.htm`.
> - SOA Web service reference, set the property overrides to glassfish.credentials in the Security page. For more information, see Section 46.2.2 "How to Override Policy Configuration Property Values" in Developer's Guide for SOA Suite at `http://docs.oracle.com/middleware/1213/soasuite/develop-soa/soa-security-policies-jdev.htm#SOASE85427`.

9.3 SAML Token (Sender Vouches) with Message Protection (WS-Security 1.1)

This section tells how to implement SAML token (sender vouches) with message protection that conforms to the WS-Security 1.1 standard, in the following interoperability scenarios:

- "Configuring an OWSM 12c Web Service and a GlassFish Client" on page 9-5
- "Configuring a GlassFish Web Service and an OWSM 12c Client" on page 9-6

9.3.1 Configuring an OWSM 12*c* Web Service and a GlassFish Client

The following instructions tell how to configure an OWSM 12*c* web service and a GlassFish client to implement SAML token (sender vouches) with message protection that conforms to the WS-Security 1.1 standard.

To Configure Prerequisites for Interoperability:

1. Create a `default-keystore.jks` file with the following command:

   ```
   $JAVA_HOME/bin/keytool -genkeypair -alias orakey -keypass welcome -keyalg RSA
    -dname "CN=orakey, O=oracle C=us" -keystore default-keystore.jks -storepass
    welcome
   ```
2. Copy `default-keystore.jks` to the domain's `fmwconfig` directory.
3. Create a file user in GlassFish with the following command:

   ```
   $<GLASSFISHV3_HOME>/glassfish/bin/asadmin create-file-user
   ```
   For more information, see http://download.oracle.com/docs/cd/E18930_01/html/821-2433/create-file-user-1.html
4. Add the user. For more information, see "Create users" in *Oracle WebLogic Server Administration Console Online Help*.
5. Import `orakey` from `default-keystore.jks` into GlassFish keystore and truststore. These are located in the directory `<domain-dir>/config`

   ```
   $JAVA_HOME/bin/keytool -importkeystore -srckeystore
    <path-to>/default-keystore.jks -destkeystore
    <path-to-gf-domain>/config/cacerts.jks -srcalias  orakey -destalias orakey
    -srckeypass welcome -destkeypass changeit
   ```
6. Copy `jps-config.xml` and `default-keystore.jks` from the domain's `fmwconfig` directory into a local folder.

To Configure the OWSM 11g Web Service:

1. Create a web service.
2. Attach the following policy to the web service: `oracle/wss11_saml_token_with_message_protection_service_policy`. For more information, see "Attaching Policies" in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

To Configure the GlassFish/Metro Client:

1. Using NetBeans, create a Metro client by selecting **New Project > Java > Java Application**. Provide a project name and location. Select the server to deploy and select **Finish**.
2. Right-click the project. Select **New > Web Service Client**. Follow the wizard and provide WSDL URL for service deployed in WebLogic.
3. Create a SAML CallbackHandler that can be used with WSIT SAML Security Mechanisms supported by NetBeans.

   ```
   Is the link to the sample callbackhandler file an external link that is
   available to customers? ---- This is a comment. Please check with the SME and
   update this section with relevant conent.
   ```
   a. Place the file in the source folder of the project.

   b. Ensure issuer variable value is the same as in the `jps-config.xml` file created in Step 5 of Table 9–8, " Configuration Prerequisites for Interoperability".

   c. Set the urn reference to `urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified`.

   d. Set the user created in Step 3 and Step 4 of Table 9–11, " Configuration Prerequisites for Interoperability". For example, to set the user to wlsuser, modify the file as follows: `CN=wlsuser,OU=SU,O=wlsuser,L=Los Angeles,ST=CA,C=US`.

4. To configure the JVM, log on to the GlassFish Administration Console.

   a. In the left pane, expand **Configuration** and click **JVM Setting**.

   b. In the right pane, click JVM Option tab.

   c. Click Add JVM Option. A new text field is displayed. Enter -DWSIT_HOME=${com.sun.aas.installRoot}.

   d. Click Enterprise Server in left pane.

   e. Click Restart in the right pane to restart the server.

   For more information, see *Oracle GlassFish Server 3.1 Administration Guide* at: `http://download.oracle.com/docs/cd/E18930_01/html/821-2416/gepzd.html`.

5. Expand Web Services Reference node. Using NetBeans, right click S**ervice Reference** and select **Edit Web Services** Attributes.

6. For SAML Callback Handler option, click Browse and select the file from Step 3.

7. Set the alias in Keystore and Truststore.

8. Open index.jsp file. Right click and select **Web Service Client Reference**. Select Operation in **Select Operation to Invoke** dialog box and click ok.

9. Run the project.

9.3.2 Configuring a GlassFish Web Service and an OWSM 12*c* Client

The following instructions tell how to configure an GlassFish web service and a OWSM 12*c* client to implement SAML token (sender vouches) with message protection that conforms to the WS-Security 1.1 standard.

To Configure Prerequisites for Interoperability:

1. Create a default-keystore.jks file with the following command:

```
$JAVA_HOME/bin/keytool -genkeypair -alias orakey -keypass welcome -keyalg RSA
 -dname "CN=orakey, O=oracle C=us" -keystore default-keystore.jks -storepass
 welcome
```

2. Copy default-keystore.jks to the domain's fmwconfig directory.

3. Save the credentials in credential store using WLST commands.

   For example:

```
$<ORACLE_HOME>/common/bin/wlst.sh
> connect()
> createCred(map="oracle.wsm.security", key="keystore-csf-key",
 user="keystore", password="welcome")
> createCred(map="oracle.wsm.security", key="sign-csf-key", user="orakey",
 password="welcome")
> createCred(map="oracle.wsm.security", key="enc-csf-key", user="orakey",
 password="welcome")
>createCred(map="oracle.wsm.security", key="glassfish.credentials" ,
 user="wlsUser" , password="welcome1" , description="Glassfish user
 credentials");
```

   A file cwallet.sso is created in the directory DOMAIN_HOME/config/fmwconfig.

4. Create a file user in GlassFish with the following command:

   ```
   $<GLASSFISHV3_HOME>/glassfish/bin/asadmin create-file-user
   ```

   For more information, see http://download.oracle.com/docs/cd/E18930_01/html/821-2433/create-file-user-1.html

5. Import orakey from default-keystore.jks into GlassFish keystore and truststore. These are located in the directory <domain-dir>/config.

   ```
   $JAVA_HOME/bin/keytool -importkeystore -srckeystore
    <path-to>/default-keystore.jks -destkeystore
    <path-to-gf-domain>/config/keystore.jks -srcalias  orakey -destalias orakey
    -srckeypass welcome -destkeypass changeit
   ```

6. Copy cwallet.sso, jps-config.xml and default-keystore.jks from the domain's fmwconfig directory into a local folder.

To Configure GlassFish/Metro Web Service:

1. Create a Metro web service.

   For more information, see http://metro.java.net/guide/ch02.html#using_metro-developing_with_nb

2. Configure the appropriate security mechanism. For more information, see `http://metro.java.net/guide/ch12.html#ahicu`

To Configure the OWSM 11g Client:

1. Using JDeveloper, create a web service proxy for the GlassFish service. Select the policy `oracle/wss11_saml_token_with_message_protection_client_policy` in the wizard.

   For more information, see " Developing and Securing Web Services " in *Developing Applications with Oracle JDeveloper*.

2. Set the path to `jps-config.xml` created in Step 6 of Table 9–11, " Configuration Prerequisites for Interoperability".

3. Set the USERNAME_PROPERTY as follows: `((BindingProvider) sAMLTokenEchoService).getRequestContext().put(BindingProvider.USERNAME_PROPERTY, "wlsUser");`

4. Invoke the web service.