SAML v2.0 Guide

ForgeRock Access Management 5

ForgeRock AS
201 Mission St, Suite 2900
San Francisco, CA 94105, USA
+1 415-599-1100 (US)
www.forgerock.com

Abstract

Guide to working with SAML v2.0. ForgeRock# Access Management provides authentication, authorization, entitlement and federation software.

Table of Contents

Preface iv
1. Introducing SAML v2.0 Support 1
    1.1. Using the Fedlet 2
2. Implementing SAML v2.0 Using the AM Console 3
    2.1. Preparing for Configuring SAML v2.0 3
    2.2. SAML v2.0 Deployment Overview 3
    2.3. Configuring Identity Providers, Service Providers, and Circles of Trust 4
    2.4. Implementing SAML v2.0 Single Sign-On and Single Logout 21
    2.5. Managing Federated Accounts 40
    2.6. SAML v2.0 and Session State 47
3. Implementing SAML v2.0 Service Providers Using the Fedlet 48
    3.1. Using Fedlets in Java Web Applications 48
    3.2. Configuring Java Fedlets By Hand 69
4. Customizing SAML v2.0 Support 91
    4.1. Installing the SAE Samples 92
    4.2. Preparing to Secure SAE Communications 92
    4.3. Securing the Identity Provider Side 93
    4.4. Securing the Service Provider Side 94
    4.5. Trying It Out 95
5. Reference 96
    5.1. SAML v2.0 Standards 96
    5.2. SAML v2.0 Configuration Properties 96
    5.3. SAML v2.0 Global Services Properties 108
A. Getting Support 118
    A.1. Accessing Documentation Online 118
    A.2. Joining the ForgeRock Community 119
    A.3. Getting Support and Contacting ForgeRock 119
Glossary 120

Preface

This guide covers concepts, configuration, and usage procedures for working with the Security Assertion Markup Language (SAML) v2.0 features provided by ForgeRock Access Management.

This guide is written for anyone using ForgeRock Access Management for SAML v2.0 identity and service providers, and for anyone using the Fedlet as a SAML v2.0 service provider.

About ForgeRock Identity Platform™ Software

ForgeRock Identity Platform™ is the only offering for access management, identity management, user-managed access, directory services, and an identity gateway, designed and built as a single, unified platform.

The platform includes the following components that extend what is available in open source projects to provide fully featured, enterprise-ready software:

- ForgeRock Access Management (AM)
- ForgeRock Identity Management (IDM)
- ForgeRock Directory Services (DS)
- ForgeRock Identity Gateway (IG)

Chapter 1

Introducing SAML v2.0 Support

OpenAM servers can function as identity providers and service providers in a SAML v2.0 circle of trust.

SAML v2.0 SSO is part of federated access management. Federation lets access management cross organizational boundaries. Federation helps organizations share identities and services without giving away their identity information, or the services they provide.

To bridge heterogeneous systems, federation requires interoperability, and thus depends on standards for orchestrating interaction and exchanging information between providers. OpenAM federation relies on standards, such as Security Assertion Markup Language (SAML) v2.0. SAML v2.0 describes the messages, how they are relayed, how they are exchanged, and common use cases.

To achieve SAML v2.0 SSO, OpenAM separates *identity providers* from *service providers*, lets you include them in a *circle of trust* and configure how the providers in the circle of trust interact:

- An identity provider stores and serves identity profiles, and handles authentication.
- A service provider offers services that access protected resources and handles authorization.
- A circle of trust groups at least one identity provider and at least one service provider who agree to share authentication information with assertions about authenticated users that let service providers make authorization decisions.

  Providers in a circle of trust share *metadata*, configuration information that federation partners require to access each others' services.
- SAML v2.0 SSO maps attributes from accounts at the identity provider to attributes on accounts at the service provider. The identity provider makes assertions to the service provider, for example, to attest that a user has authenticated with the identity provider. The service provider then consumes assertions from the identity provider to make authorization decisions, for example to let an authenticated user complete a purchase that gets charged to the user's account at the identity provider.

In federation deployments where not all providers support SAML v2.0, OpenAM can act as a multi-protocol hub, translating for providers who rely on other and older standards, such as SAML v1.x, Liberty Alliance Project frameworks, and WS-Federation (for integration with Active Directory Federation Services, for example).

1.1. Using the Fedlet

When your organization acts as the identity provider and you want to enable service providers to federate their services with yours, you can generate configuration files for a *Fedlet*. A Fedlet is a small Java web application that can act as a service provider for a specific identity provider without requiring that you install all of OpenAM.

After receiving the configuration files for the Fedlet, the service provider administrator installs them, and then obtains the Fedlet web application from the OpenAM distribution and installs it in the application web container.

Fedlets support SAML v2.0 features, as shown in the following table:

Table 1.1. Fedlet Support for SAML v2.0 Features

SAML v2.0 Feature	Java Fedlet
IdP and SP-initiated Single Sign-On (HTTP Artifact)	Supported
IdP and SP-initiated Single Sign-On (HTTP POST)	Supported
IdP and SP-initiated Single Logout (HTTP POST)	Supported
IdP and SP-initiated Single Logout (HTTP Redirect)	Supported
Sign Requests and Responses	Supported
Encrypt Assertion, Attribute, and NameID Elements	Supported
Export SP Metadata	Supported
Attribute Queries	Supported
XACML Requests	Supported
Multiple IdPs	Supported
External IdP Discovery Service	Supported
Bundled IdP Reader Service for Discovery	Supported

For more information on installing and using Fedlets, see Chapter 3, "*Implementing SAML v2.0 Service Providers Using the Fedlet*".

Chapter 2

Implementing SAML v2.0 Using the AM Console

This chapter covers implementation of OpenAM's SAML v2.0 component using the AM console.

2.1. Preparing for Configuring SAML v2.0

Before you set up SAML v2.0 SSO in OpenAM, you must:

- Know which providers will participate in circles of trust.
- Know how OpenAM installations act as identity providers or service providers.
- Determine whether your session state configuration limits your usage of certain SAML v2.0 profiles. For more information, see Section 2.6, "SAML v2.0 and Session State".
- Agree with other providers on a synchronized time service.
- Define how to map shared user attributes in identity information exchanged with other participants in a circle of trust. Local user profile attribute names should map to user profile attribute names at other providers.

  For example, if you exchange user identifiers with your partners, and you call it `uid`, whereas another partner calls it `userid`, then you map your `uid` to your partner's `userid`.
- Import the keys used to sign assertions into the keystore in your OpenAM configuration directory. You can use the Java **keytool** command.

  For more information about OpenAM keystores, including location and different types of keystores available and how to change the default keys, see Chapter 5, "*Setting Up Keys and Keystores*" in the *Setup and Maintenance Guide*.

2.2. SAML v2.0 Deployment Overview

Setting up and managing SAML v2.0 for SSO and SLO comprises three processes:

- Configuring identity providers, service providers, and circles of trust.

OpenAM provides wizards that let you configure SAML v2.0 identity providers, service providers, and circles of trust, which define the relationships among providers. You can also configure providers and circles of trust using the AM console and the **ssoadm** command.

See Section 2.3, "Configuring Identity Providers, Service Providers, and Circles of Trust" for procedures to configure identity providers, service providers, and circles of trust.

- Preparing your applications to initiate SSO and SLO.

  After configuring the providers and circles of trust, you can implement OpenAM's support for SSO and SLO in your applications.

  See Section 2.4, "Implementing SAML v2.0 Single Sign-On and Single Logout" for procedures to implement SAML v2.0 SSO and SLO in OpenAM.

- Managing federated accounts.

  After you have implemented SAML v2.0 single sign-on, there are several tasks you might perform when using federated account linking.

  See Section 2.5, "Managing Federated Accounts" for procedures to manage federated accounts.

2.3. Configuring Identity Providers, Service Providers, and Circles of Trust

This section covers configuration tasks you perform before you can implement SAML v2.0 SSO and SLO.

During setup, you must share metadata for providers that you host with other providers in the circle of trust. You must also configure remote providers, connecting to other providers by importing their metadata. In OpenAM terms, a hosted provider is one served by the current OpenAM server; a remote provider is one hosted elsewhere.

This section provides procedures for performing the following tasks:

Table 2.1. Tasks for Configuring Entity Providers and Circles of Trust

Task	See Section(s)
(Required) Creating identity and service providers.	Section 2.3.1, "Creating a Hosted Identity Provider" Section 2.3.2, "Creating a Hosted Service Provider" Section 2.3.3, "Configuring a Remote Identity Provider" Section 2.3.4, "Configuring a Remote Service Provider"

Task	See Section(s)
(Optional) Modifying identity provider, service provider, and circle of trust configurations. You might need to modify these configurations after you have created them using the wizards.	Section 2.3.5, "Modifying Provider and Circle of Trust Configuration"
(Optional) Deploying an identity provider discovery service. When your circle of trust includes multiple identity providers, then service providers must *discover* which identity provider corresponds to a request. You can deploy the identity provider discovery service for this purpose as a separate web application.	Section 2.3.6, "Deploying the Identity Provider Discovery Service"
(Optional) Configuring providers for failover.	Section 2.3.7, "Configuring Providers for Failover"
(Optional) Configuring Google Apps and Salesforce CRM as service providers.	Section 2.3.8, "Configuring Google Apps as a Remote Service Provider" Section 2.3.9, "Configuring Salesforce CRM as a Remote Service Provider"
(Optional) Creating a *Fedlet* for an OpenAM service provider. A Fedlet is an example web application that acts as a lightweight SAML v2.0 service provider.	Chapter 3, "*Implementing SAML v2.0 Service Providers Using the Fedlet*"

2.3.1. Creating a Hosted Identity Provider

The following procedure provides steps for creating a hosted identity provider by using the Create Hosted Identity Provider wizard:

Procedure 2.1. To Create a Hosted Identity Provider

1. Under Realms > *`Realm Name`* > Dashboard > Configure SAMLv2 Providers, click Create Hosted Identity Provider.

2. Unless you already have metadata for the provider, accept the Name for this identity provider in the field provided, or provide your own unique identifier.

   The default name is the URL to the current server which hosts the identity provider.

3. Select the Signing Key alias you imported into the OpenAM keystore as part of your preparation for SAML v2.0 configuration.

4. Either add the provider to the circle of trust you already created, or select the Add to new option and provide a New Circle of Trust name.

5. For the attributes you share, map service provider attribute names (Name in Assertion), to user profile names from your identity repository (Local Attribute Name).

   Use this approach to set up a mapping with all SPs in the circle of trust that do not have their own specific mappings configured.

   The default mapping implementation has additional features beyond simply retrieving string attributes from the user profile.

   - Add an attribute that takes a static value by enclosing the profile attribute name in double quotes (`"`).

     For example, you can add a static SAML attribute called `partnerID` with a value of `staticPartnerIDValue` by adding `partnerID` as the Name in Assertion with `"staticPartnerIDValue"` as the Local Attribute Name.

   - Base64 encode binary attributes when adding them to the SAML attributes by adding `;binary` to the end of the attribute name, as in the following example:

     ```
     objectGUID=objectGUID;binary
     ```

     This maps the local binary attribute `objectGUID` to a SAML attribute called `objectGUID` that is Base64 encoded.

   - Use `NameFormatURI` format as shown in the following example:

     ```
     urn:oasis:names:tc:SAML:2.0:attrname-format:uri|objectGUID=objectGUID;binary
     ```

6. Click Configure to save your configuration.

7. Export the XML-based metadata from your provider to share with other providers in your circle of trust.

   ```
   $ curl \
    --output metadata.xml \
    "http://www.idp.example:8080/openam/saml2/jsp/exportmetadata.jsp?entityid=\
    http://www.idp.example:8080/openam&realm=/realm-name"
   ```

   When you have configured your provider in the Top Level Realm, you can omit the query string from the URL.

   Alternatively, provide the URL to other providers so they can load the metadata.

2.3.2. Creating a Hosted Service Provider

The following procedure provides steps for creating a hosted service provider by using the Create Hosted Service Provider wizard:

Procedure 2.2. To Create a Hosted Service Provider

1. Under Realms > *Realm Name* > Dashboard > Configure SAMLv2 Providers, click Create Hosted Service Provider.
2. Unless you already have metadata for the provider, accept the Name for this service provider in the field provided, or provide your own unique identifier.

   The default name is the URL to the current server which hosts the service provider.
3. Either add the provider to the circle of trust you already created, or select the Add to new option and provide a New Circle of Trust name.
4. If this SP requires more a different attribute mapping configuration than the default IdP attribute mapping, set the mapping in the Attribute Mapping section. Map identity provider attribute names in the Name in Assertion column to user profile names from your identity repository in the Local Attribute Name column.
5. Click Configure to save your configuration.
6. Export the XML-based metadata from your provider to share with other providers in your circle of trust:

   ```
   $ curl \
    --output metadata.xml \
    "http://www.sp.example:8080/openam/saml2/jsp/exportmetadata.jsp?entityid=\
    http://www.sp.example:8080/openam&realm=/realm-name"
   ```

   When you have configured your provider in the Top Level Realm, you can omit the query string from the URL.

   Alternatively, provide the URL to other providers so they can load the metadata.

2.3.3. Configuring a Remote Identity Provider

The following procedure provides steps for configuring a remote identity provider by using the Register Remote Identity Provider wizard:

Procedure 2.3. To Configure a Remote Identity Provider

1. Obtain the identity provider metadata or the URL where you can obtain it.
2. Under Realms > *Realm Name* > Dashboard > Configure SAMLv2 Providers, click Configure Remote Identity Provider.
3. Provide the identity provider metadata or link to obtain metadata.

   The remote identity provider's metadata might contain more than one `KeyDescriptor` elements. If it does, the hosted OpenAM service provider will validate assertions from the identity provider

against certificates with key descriptors with an appropriate `use` attribute. Incoming assertions that cannot be validated against any of the certificates will be rejected by the hosted service provider.

4. Either add the provider to the circle of trust you already created, or select Add to new and provide a New Circle of Trust name.
5. Click Configure to save your configuration.

2.3.4. Configuring a Remote Service Provider

The following procedure provides steps for configuring a remote service provider by using the Register Remote Service Provider wizard:

Procedure 2.4. To Configure a Remote Service Provider

1. Obtain the service provider metadata, or the URL where you can obtain it.
2. Under Realms > *`Realm Name`* > Dashboard > Configure SAMLv2 Providers, click Configure Remote Service Provider.
3. Provide the service provider metadata or link to obtain metadata.

   The remote service provider's metadata might contain more than one `KeyDescriptor` element. In this case, the hosted identity provider should consider any incoming SAML requests from the service provider to be valid as long as it can be validated with any of the certificates.
4. If the identity provider has not already mapped the attributes you share, map identity provider attribute names (Name in Assertion) to user profile names from your identity repository (Local Attribute Name).

   Use this approach to set up a mapping that is specific to this SP. Note that a remote SP-specific attribute mapping overrides the attribute mapping configuration specified in the hosted IdP configuration.

   The default mapping implementation has additional features beyond simply retrieving string attributes from the user profile.

   - Add an attribute that takes a static value by enclosing the profile attribute name in double quotes (`"`).

     For example, you can add a static SAML attribute called `partnerID` with a value of `staticPartnerIDValue` by adding `partnerID` as the Name in Assertion with `"staticPartnerIDValue"` as the Local Attribute Name.
   - Base64 encode binary attributes when adding them to the SAML attributes by adding `;binary` to the end of the attribute name, as in the following example:

     ```
     objectGUID=objectGUID;binary
     ```

This maps the local binary attribute `objectGUID` to a SAML attribute called `objectGUID` that is Base64 encoded.

- Use `NameFormatURI` format as shown in the following example:

```
urn:oasis:names:tc:SAML:2.0:attrname-format:uri|objectGUID=objectGUID;binary
```

5. Either add the provider to the circle of trust you already created, or select Add to new and provide a New Circle of Trust name.
6. Click Configure to save your configuration.

2.3.5. Modifying Provider and Circle of Trust Configuration

After you have set up federation components, you can configure them through the Federation menu in the AM console as follows:

- To configure hosted identity providers, navigate to Realms > *Realm Name* > Applications > SAML > Entity Providers > *Provider Name*. For information about configurable hosted identity provider properties, see Section 5.2.1, "Hosted Identity Provider Configuration Properties".
- To configure hosted service providers, navigate to Realms > *Realm Name* > Applications > SAML > Entity Providers > *Provider Name*. For information about configurable hosted service provider properties, see Section 5.2.2, "Hosted Service Provider Configuration Properties".
- To configure circles of trust, navigate to Realms > *Realm Name* > Applications > SAML > Circle of Trust > *Circle of Trust Name*. For information about circle of trust properties, see Section 5.2.3, "Circle of Trust Configuration Properties".

2.3.6. Deploying the Identity Provider Discovery Service

When your circle of trust includes multiple identity providers, then service providers must discover which identity provider corresponds to a request. You can deploy the identity provider discovery service for this purpose as a separate web application.

Browsers only send cookies for the originating domain. Therefore, when a browser accesses the service provider in the `www.sp.example` domain, the service provider has no way of knowing whether the user has perhaps already authenticated at `www.this-idp.example` or at `www.that-idp.example`. The providers therefore host an identity provider discovery service in a common domain, such as `www.disco.example`, and use that service to discover where the user logged in. The identity provider discovery service essentially writes and reads cookies from the common domain. The providers configure their circle of trust to use the identity provider discovery service as part of SAML v2.0 federation.

Deploying the identity provider discovery service involves the following stages:

1. Deploy the `.war` into your web application container.
2. Configure the discovery service.

3. Add the identity provider discovery service endpoints for writing cookies to and reading cookies from the common domain to the circle of trust configurations for the providers.

4. Share metadata between identity providers and the service provider.

Procedure 2.5. To Deploy the Discovery Service on Tomcat

How you deploy the discovery service `.war` file depends on your web application container. The procedure in this section shows how to deploy on Apache Tomcat.

1. Copy the `IDPDiscovery-14.0.0.war` file to the `webapps/` directory.

```
$ cp ~/Downloads/openam/IDPDiscovery-14.0.0.war \
 /path/to/tomcat/webapps/disco.war
```

2. Access the configuration screen through your browser.

   In this example, Apache Tomcat listens for HTTP requests on `www.disco.example:8080`, and Tomcat has unpacked the application under `/disco`, so the URL is `http://www.disco.example:8080/disco`, which redirects to `Configurator.jsp`.

Procedure 2.6. To Configure the Discovery Service

1. Configure the identity provider discovery service.

Figure 2.1. Completed Discovery Service Configuration Screen

Hints for discovery service configuration parameters follow.

Debug Directory

The discovery service logs to flat files in this directory.

Debug Level

Default is `error`. Other options include `error`, `warning`, `message`, and `off`.

Set this to `message` in order to see the service working when you run your initial tests.

Cookie Type

Set to PERSISTENT if you have configured OpenAM to use persistent cookies, meaning single sign-on cookies that can continue to be valid after the browser is closed.

Cookie Domain

The cookie domain is the common cookie domain used in your circle of trust for identity provider discovery, in this case `.disco.example`.

Secure Cookie

Set this to true if clients should only return cookies when a secure connection is used.

Encode Cookie

Leave this true unless your OpenAM installation requires that you do not encode cookies. Normally, cookies are encoded such that cookies remain valid in HTTP.

HTTP-Only Cookie

Set to true to use HTTPOnly cookies if needed to help prevent third-party programs and scripts from accessing the cookies.

2. Restrict permissions to the discovery service configuration file in `$HOME/libIDPDiscoveryConfig.properties`, where $HOME corresponds to the user who runs the web container where you deployed the service.

Procedure 2.7. To Add the Discovery Service to Your Circles of Trust

Each provider has a circle of trust including itself. You configure each of these circles of trust to use the identity provider discovery service as described in the following steps:

1. On the service provider console, login as OpenAM Administrator.

2. On the service provider console, under Realms > *`Realm Name`* > Applications > SAML > Circle of Trust > *`Circle of Trust Name`*, add SAML2 Writer and Reader Service URLs for the identity provider discovery service endpoints, and save your work.

   In this example, the writer URL is `http://www.disco.example:8080/disco/saml2writer`, and the reader URL is `http://www.disco.example:8080/disco/saml2reader`.

3. On each identity provider console, login as OpenAM Administrator.

4. On the identity provider console, under Realms > *Realm Name* > Applications > SAML > Entity Providers > Circle of Trust > *Circle of Trust Name*, also add SAML2 Writer and Reader Service URLs for the identity provider discovery service endpoints, and save your work.

Procedure 2.8. To Share Identity and Service Provider Metadata

Before performing these steps, install the administration tools for each provider as described in Section 2.3.1, "Setting up Administration Tools" in the *Installation Guide*. The administration tools include the **ssoadm** command that you need to export metadata:

1. On each identity provider console, register the service provider as a remote service provider adding to the circle of trust you configured to use the identity provider discovery service.

   The URL to the service provider metadata is something like `http://www.sp.example:8080/openam/saml2/jsp/exportmetadata.jsp`.

2. Create metadata templates for each identity provider:

```
$ ssh www.this-idp.example
$ cd /path/to/openam-tools/admin/openam/bin
$ ./ssoadm \
 create-metadata-templ \
 --entityid "http://www.this-idp.example:8080/openam" \
 --adminid amadmin \
 --password-file /tmp/pwd.txt \
 --identityprovider /idp \
 --meta-data-file this-standard.xml \
 --extended-data-file this-extended.xml
Hosted entity configuration was written to this-extended.xml.
Hosted entity descriptor was written to this-standard.xml.

$ ssh www.that-idp.example
$ cd /path/to/openam-tools/admin/openam/bin
$ ./ssoadm \
 create-metadata-templ \
 --entityid "http://www.that-idp.example:8080/openam" \
 --adminid amadmin \
 --password-file /tmp/pwd.txt \
 --identityprovider /idp \
 --meta-data-file that-standard.xml \
 --extended-data-file that-extended.xml

Hosted entity configuration was written to that-extended.xml.
Hosted entity descriptor was written to that-standard.xml.
```

3. For each identity provider extended metadata file, change the value of the `hosted` attribute to `0`, meaning the identity provider is remote.

4. On the service provider, add the identity providers to the circle of trust using the identity provider metadata.

```
$ ssh www.sp.example
$ cd /path/to/openam-tools/admin/openam/bin
$ ./ssoadm \
 import-entity \
 --cot discocot \
 --meta-data-file ~/Downloads/this-standard.xml \
 --extended-data-file ~/Downloads/this-extended.xml \
 --adminid amadmin \
 --password-file /tmp/pwd.txt

Import file, /Users/mark/Downloads/this-standard.xml.
Import file, /Users/mark/Downloads/this-extended.xml.
$ ./ssoadm \
 import-entity \
 --cot discocot \
 --meta-data-file ~/Downloads/that-standard.xml \
 --extended-data-file ~/Downloads/that-extended.xml \
 --adminid amadmin \
 --password-file /tmp/pwd.txt

Import file, /Users/mark/Downloads/that-standard.xml.
Import file, /Users/mark/Downloads/that-extended.xml.
```

5. Test your work by using the Federation Connectivity Test that you start from the service provider console under Under Realms > *Realm Name* > Dashboard > Test Federation Connectivity.

   When the test is done, you can see messages from the `CookieWriterServlet` in the `libIDPDiscovery` log file where you set up logging when you configured the identity provider discovery service, such as `/tmp/debug/libIDPDiscovery`. Output generated during a test follows, with some lines folded to fit on the printed page.

```
08/08/2012 11:43:38:341 AM CEST: Thread[http-bio-8080-exec-4,5,main]
CookieUtils.init : idpDiscoveryOnlyWar=true
08/08/2012 11:43:38:341 AM CEST: Thread[http-bio-8080-exec-4,5,main]
CookieWriterServlet Initializing...
08/08/2012 11:43:38:341 AM CEST: Thread[http-bio-8080-exec-4,5,main]
CookieWriterServlet.doGetPost: Preferred Cookie Name is _saml_idp
08/08/2012 11:43:38:341 AM CEST: Thread[http-bio-8080-exec-4,5,main]
CookieWriterServlet.doGetPost: URL Scheme is null, set to https.
08/08/2012 11:43:38:341 AM CEST: Thread[http-bio-8080-exec-4,5,main]
CookieWriterServlet.doGetPost: Preferred IDP Cookie Not found
08/08/2012 11:43:38:342 AM CEST: Thread[http-bio-8080-exec-4,5,main]
CookieWriterServlet.doGetPost: Cookie Type is PERSISTENT
08/08/2012 11:43:38:342 AM CEST: Thread[http-bio-8080-exec-4,5,main]
CookieWriterServlet.doGetPost: Cookie value is
 aHR0cDovL3d3dy50aGF0LWlkcC5jb206ODA4MC9vcGVuYW0=
08/08/2012 11:43:38:342 AM CEST: Thread[http-bio-8080-exec-4,5,main]
CookieWriterServlet.doGetPost: Preferred Cookie Name _saml_idp
08/08/2012 11:43:38:343 AM CEST: Thread[http-bio-8080-exec-4,5,main]
CookieWriterServlet.doGetPost: Redirect to
 http://www.that-idp.example:8080/openam/SSORedirect/metaAlias/idp?resInfoID=
 s28bc4db004f1365d78d07d69846c54a3c850fe801
08/08/2012 11:43:46:957 AM CEST: Thread[http-bio-8080-exec-4,5,main]
CookieWriterServlet.doGetPost: Preferred Cookie Name is _saml_idp
08/08/2012 11:43:46:957 AM CEST: Thread[http-bio-8080-exec-4,5,main]
CookieUtils:cookieValue=aHR0cDovL3d3dy50aGF0LWlkcC5jb206ODA4MC9vcGVuYW0=,
```

```
 result=aHR0cDovL3d3dy50aGF0LWlkcC5jb206ODA4MC9vcGVuYW0=
08/08/2012 11:43:46:957 AM CEST: Thread[http-bio-8080-exec-4,5,main]
CookieWriterServlet.doGetPost: Cookie Type is PERSISTENT
08/08/2012 11:43:46:957 AM CEST: Thread[http-bio-8080-exec-4,5,main]
CookieWriterServlet.doGetPost: Cookie value is
 aHR0cDovL3d3dy50aGF0LWlkcC5jb206ODA4MC9vcGVuYW0=
08/08/2012 11:43:46:957 AM CEST: Thread[http-bio-8080-exec-4,5,main]
CookieWriterServlet.doGetPost: Preferred Cookie Name _saml_idp
08/08/2012 11:43:46:957 AM CEST: Thread[http-bio-8080-exec-4,5,main]
CookieWriterServlet.doGetPost: Redirect to
 http://www.that-idp.example:8080/openam/SSORedirect/metaAlias/idp?resInfoID=
 s2ce9c465cf39c96f31e1dcf009cf9943695d82901
```

2.3.7. Configuring Providers for Failover

OpenAM servers can function in a site configuration behind a load balancer. In addition to configuring the OpenAM site as described in Procedure 2.3, "To Custom Configure an Instance" and Section 2.2, "Installing Multiple Servers" in the *Installation Guide*, update provider metadata to reference the load balancer rather than the server as follows:

1. Before configuring the provider, follow the instructions in the *Installation Guide* mentioned above, and make sure that failover works through the load balancer for normal OpenAM sessions.

2. Configure the provider on one of the servers using the load balancer URL as the entity ID.

3. Export the metadata and extended metadata for the provider by using the **ssoadm** command. You can export the metadata as shown in the following example for an identity provider, where the entity ID is `http://lb.example.com:80/openam`.

   ```
   $ ssoadm \
    export-entity \
    --entityid "http://lb.example.com:80/openam" \
    --adminid amadmin \
    --password-file /tmp/pwd.txt \
    --meta-data-file idp.xml \
    --extended-data-file idp-extended.xml
   ```

4. Edit both the metadata and the extended metadata, changing all URLs in both files to use the load balancer URL.

5. Delete the provider configuration in the AM console.

6. Import the edited provider configuration in the AM console.

7. Enable SAML v2.0 failover in the AM console.

   Navigate to Configure > Global Services, and then click SAMLv2 Service Configuration.

   Select Enabled next to Enable SAMLv2 failover, and then click Save.

At this point failover is operational for the provider you configured.

2.3.8. Configuring Google Apps as a Remote Service Provider

OpenAM can serve as the identity provider when you use Google Apps as a service provider, allowing users to have single sign-on with their Google Apps account.

In order to use this service, you must have a Google Apps account for at least one of your domains, such as `example.com`.

Procedure 2.9. To Integrate With Google Apps

1. If you have not yet done so, set up OpenAM as described in Procedure 2.1, "To Create a Hosted Identity Provider". As part of the IdP configuration, you specify a signing key alias. In a subsequent step, you will provision Google Apps with this certificate's public key.

   For details about changing the signing certificate, see Procedure 5.3, "To Change Default test Signing Key" in the *Setup and Maintenance Guide*.

2. Under Realms > *`Realm Name`* > Dashboard, click Configure Google Apps.

3. On the first Configure Google Apps for Single Sign-On page, add your domain name(s), such as `example.com` to the list, and then click Create.

4. On the second Configure Google Apps for Single Sign-On page, save the OpenAM verification certificate to a text file, such as `OpenAM.pem`.

5. Follow the instructions under To Enable Access to the Google Apps API before clicking Finish.

   a. Access the Google Apps administration page for the first of your domains in a new browser tab or window.

   b. Login as Google Apps administrator.

   c. Select Enable Single Sign-On.

   d. Copy the URLs from the OpenAM page into the Google Apps setup screen.

   e. Upload the certificate file you saved, such as `OpenAM.pem` as the Google Apps Verification Certificate.

   f. Select Use a domain specific issuer.

   g. Save changes in Google Apps setup.

   h. Repeat the steps above for each domain you have configured.

   i. Click Finish to complete the process.

2.3.9. Configuring Salesforce CRM as a Remote Service Provider

OpenAM can serve as the identity provider when you use Salesforce CRM as a service provider, allowing users to have single sign-on with their Salesforce CRM account.

In order to use this service, you must have Salesforce CRM accounts for your organization or enable Salesforce just-in-time provisioning, which uses content from the SAML assertion created by OpenAM to create regular and portal users in Salesforce the first time they attempt to log in. To enable Salesforce just-in-time provisioning, see Procedure 2.11, "To Enable Salesforce CRM Just-in-Time Provisioning".

Procedure 2.10. To Integrate With Salesforce CRM

1. If you have not yet done so, set up OpenAM as described in Procedure 2.1, "To Create a Hosted Identity Provider", using a signing certificate that is needed by Salesforce CRM.

   For details about changing the signing certificate, see Procedure 5.3, "To Change Default test Signing Key" in the *Setup and Maintenance Guide*.

2. If you do not have an account with administrator credentials on Salesforce CRM, create one. See the Salesforce documentation for information about how to create an account with administrator credentials.

3. In a new browser tab or window, log in to Salesforce CRM with your administrator credentials.

4. If your users go directly to Salesforce to access services, then their single sign-on is SP-initiated from the Salesforce side. Salesforce provides a My Domain feature to facilitate SP-initiated single sign-on for desktop and device users.

   Configure SP-initiated single sign-on in Salesforce as follows:

   a. Select Setup Home > Settings > Company Settings > My Domain.

   b. Select the domain name, and then register the domain.

   c. Wait until the domain is ready for testing to proceed.

   d. After the domain has been created, log out of Salesforce.

   e. Log back in to Salesforce using the domain alias.

   f. Select Setup Home > Settings > Company Settings > My Domain.

   g. Click Deploy to Users.

5. In the AM console, under Realms > *`Realm Name`* > Dashboard, click Configure Salesforce CRM.

   Click Configure Salesforce CRM a second time to start the Configure Salesforce CRM wizard.

   The Configure Salesforce CRM for Single Sign-On page appears.

6. Specify values in the Configure Salesforce CRM for Single Sign-On page as follows:

   a. Specify the Salesforce service provider entity in the "Salesforce Service Provider entityID" field. For example, `https://openam.my.salesforce.com`.

      The entity ID is used as the persistent `EntityDescriptor` metadata element so that users can have multiple service provider instances. It also appears in the Entity Providers list in the Circle of Trust Configuration.

   b. Configure an attribute mapping to associate a Salesforce CRM attribute with the corresponding OpenAM user profile attribute. For example, you might map the Salesforce CRM `IDPEmail` attribute to the OpenAM `mail` attribute.

      The Configure Salesforce CRM wizard requires you to enter at least one attribute mapping.

   c. Click Add to insert the `IDPEmail` to `mail` mapping to the Remote to Local Attribute Mapping Table.

   d. If desired, configure additional attribute mappings.

7. Click Create.

   The following message appears:

   ```
   Metadata now configured successfully.
   Click OK to retrieve the parameters for configuring the Service Provider.
   ```

8. Click OK.

   A second Salesforce CRM Single Sign-On Configuration page appears.

9. Follow the instructions on the second Salesforce CRM Single Sign-On Configuration page:

   a. Specify single sign-on settings for Salesforce as follows:

      i. In Salesforce CRM, navigate to Setup Home > Settings > Identity > Single Sign-On Settings.

      ii. Click Edit.

      iii. Select the SAML Enabled option.

   b. Create a new SAML single sign-on configuration as follows:

      i. For Issuer, copy the issuer name from the Salesforce CRM Single Sign-On Configuration page in the OpenAM Configure Salesforce CRM wizard.

      ii. Set the Name and API Name fields to values of your choosing.

iii. Copy or download the OpenAM verification certificate from the Salesforce CRM Single Sign-On Configuration page in the OpenAM Configure Salesforce CRM wizard. Save the verification certificate to a plain text file.

iv. For Identity Provider Certificate, use the Browse button to locate and upload the file containing the OpenAM verification certificate.

v. For SAML Identity Type, select the "Assertion contains the Federation ID from the User object" option.

vi. For SAML Identity Location, select the "Identity is in an Attribute" option.

vii. Specify the Identity Provider Login URL as the URL for the OpenAM IdP. For example, `https://openam.example.com:8443/openam/SSOPOST/metaalias/idp`.

viii. If you require a specific logout page, enter it in the Identity Provider Logout URL field.

ix. If you have a page to which you would like users redirected when encountering errors, enter the URL of your error page in the Custom Error URL field.

x. Copy the attribute name, such as `IDPEmail`, from the Salesforce CRM Single Sign-On Configuration page in the OpenAM Configure Salesforce CRM wizard to the Attribute Name field.

xi. Select the Entity ID corresponding to the "My Domain" that you set up.

xii. Click Save.

The Salesforce Login URL appears.

c. Perform the final steps required by the OpenAM Configure Salesforce CRM wizard:

i. Copy and paste the Salesforce Login URL to the Salesforce CRM Single Sign-On Configuration page in the OpenAM Configure Salesforce CRM wizard.

ii. Click Finish to conclude operation of the OpenAM Configure Salesforce CRM wizard.

d. Return to the Single Sign-On Settings page in Salesforce.

e. Click Download Metadata to download the Salesforce CRM SP metadata. You will import the metadata into OpenAM in a subsequent step.

f. Configure attribute mapping and name ID format for the OpenAM identity provider:

i. In the AM console, navigate to Realms > *`Realm Name`* > Applications > SAML > Entity Providers > *`Identity Provider Name`* > Assertion Processing.

ii. Review the values in the Attribute Map field, which should be the same values that you configured when you ran the Configure Salesforce CRM wizard. In this example, the values should be `IDPEmail=mail`.

If required, modify the values in the Attribute Map field, and then click Save.

iii. In the AM console, navigate to Realms > *`Realm Name`* > Applications > SAML > Entity Providers > *`Identity Provider Name`* > Assertion Content > NameID Format.

iv. Salesforce requires SAML assertions that specify an `unspecified` name ID format. In this step, configure the OpenAM-hosted IdP to support this requirement.

If a value for an `unspecified` name ID format is already present in the NameID Value Map List, remove it from the list.

v. Add the value `urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified=`*`attribute`* to the NameID Value Map List. For *`attribute`*, specify the attribute that you copied in Step 9.b.x. For example, `urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified=mail`.

vi. Click Save.

g. Add users to Salesforce CRM:

i. In Salesforce CRM, navigate to Setup Home > Administration > Users > Users.

ii. Click Users.

iii. Add users as necessary, making sure the attribute chosen as the Federation ID matches the local attribute you mapped to the remote attribute in OpenAM.

iv. Click Finish.

10. Configure OpenAM as the authentication provider for your Salesforce domain:

a. In Salesforce CRM, navigate to Setup Home > Settings > Company Settings > My Domain.

b. Click Edit in the Authentication Configuration section.

The Authentication Configuration page appears, listing the available identity providers.

c. Select the new Authentication Service.

d. Click Save.

11. Reconfigure the remote service provider definition for Salesforce in OpenAM by deleting the service provider definition created by the Configure Salesforce CRM wizard and then importing service provider metadata that you previously exported from Salesforce CRM:

a. In the AM console, navigate to Realms > *`Realm Name`* > Applications > SAML > Entity Providers.

b. Select the checkbox next to the entity provider definition for the Salesforce CRM service provider, which should be listed as an SP provider with a Remote location.

c. Click Delete to remove the entity provider configuration.

d. Click Import Entity.

The Import Entity Provider page appears.

e. Specify options on the Import Entity Provider page as follows:

- Update the Realm Name if desired.
- Click File as the location of the metadata file.
- Use the Upload button to navigate to the location of the metadata file that you obtained from Salesforce in a previous step.

12. Add the new remote service provider definition for Salesforce CRM to the federation circle of trust in OpenAM:

a. In the AM console, navigate to Realms > *Realm Name* > Applications > SAML > Circle of Trust > *Circle of Trust Name*.

b. Move the Salesforce CRM remote service provider from the Available column to the Selected column.

c. Click Save.

Configuring Salesforce CRM as a remote service provider is now complete. Users navigating to the Salesforce domain should be redirected to OpenAM for authentication. Upon successful authentication, they should be logged in to Salesforce.

Procedure 2.11. To Enable Salesforce CRM Just-in-Time Provisioning

With just-in-time provisioning enabled, Salesforce CRM automatically creates regular and portal users when new users access Salesforce by authenticating to OpenAM.

1. Add mappings to the OpenAM identity provider configuration required by Salesforce just-in-time provisioning:

a. In the AM console, navigate to Realms > *Realm Name* > Applications > SAML > Entity Providers > *Identity Provider Name* > Assertion Processing.

b. Add the following entries to the Attribute Map property:

- `User.Email=mail`
- `User.ProfileID="Standard User"`

- `User.LastName=sn`
- `User.Username=mail`

c. Click Save.

2. Enable user provisioning in Salesforce CRM:

a. Log in to your Salesforce domain.

b. In Salesforce CRM, navigate to Setup Home > Settings > Identity > Single Sign-On Settings.

c. Click Edit.

d. Set options in the Just-in-time User Provisioning section as follows:

- Select the User Provisioning Enabled check box.
- For User Provisioning Type, select Standard.

e. Click Save.

Configuring just-in-time provisioning in Salesforce CRM is now complete. When new users access Salesforce by authenticating to OpenAM, Salesforce automatically creates regular and portal users.

2.4. Implementing SAML v2.0 Single Sign-On and Single Logout

OpenAM provides two options for implementing SAML v2.0 SSO and SLO:

- *Integrated mode*, in which you include a SAML2 authentication module in an OpenAM authentication chain on a service provider (SP), thereby integrating SAML v2.0 authentication into the normal OpenAM authentication process. The authentication module handles the SAML v2.0 protocol details for you.

  Because the authentication chain that includes the SAML2 authentication module resides on the SP, integrated mode supports SP-initiated SSO only. You cannot trigger IdP-initiated SSO from an integrated mode implementation.

  Integrated mode supports both IdP-initiated and SP-initiated SLO.

  See Section 2.4.2, "Implementing SAML v2.0 SSO and SLO in Integrated Mode" for procedures to implement SSO and SLO using integrated mode.

- *Standalone mode*, in which you invoke JSPs to initiate SSO and SLO. When implementing standalone mode, you do not configure an OpenAM authentication chain.

See Section 2.4.3, "Implementing SAML v2.0 SSO and SLO in Standalone Mode" for procedures to implement SSO and SLO using standalone mode.

Integrated mode was introduced in OpenAM 13. All SAML v2.0 deployments prior to OpenAM 13 are standalone mode implementations.

When configuring OpenAM to support SAML v2.0 SSO and SLO, you choose between integrated mode and standalone mode. See Section 2.4.1, "Deciding Between Integrated Mode and Standalone Mode" for details about whether to choose integrated or standalone mode for your deployment.

2.4.1. Deciding Between Integrated Mode and Standalone Mode

You can achieve SAML v2.0 SSO and SLO by using integrated mode, in which you configure a SAML2 authentication module and integrate it into an authentication chain. Or, you can use standalone mode, in which you invoke JSPs to initiate SSO and SLO.

The following table provides information to help you decide whether to implement integrated mode or standalone mode for your OpenAM SAML v2.0 deployment:

Table 2.2. Integrated or Standalone Mode?

Deployment Task or Requirement	Implementation Mode
You are migrating an existing OpenAM SAML v2.0 deployment from OpenAM 12 to AM 5. Note that all OpenAM SAML v2.0 deployments prior to OpenAM 13 are standalone mode deployments.	Do not modify your deployment to integrated mode unless you want to change your authentication scenario to have SAML v2.0 authentication integrated into an OpenAM authentication chain.
You want to deploy SAML v2.0 SSO and SLO using the easiest technique.	Use integrated mode.
You want to integrate SAML v2.0 authentication into an authentication chain, letting you configure an added layer of login security by using additional authentication modules.	Use integrated mode.
You want to trigger SAML v2.0 IdP-initiated SSO.	Use standalone mode.
You want to use the SAML v2.0 Enhanced Client or Proxy (ECP) SSO profile.	Use standalone mode.

2.4.2. Implementing SAML v2.0 SSO and SLO in Integrated Mode

This section covers the following topics:

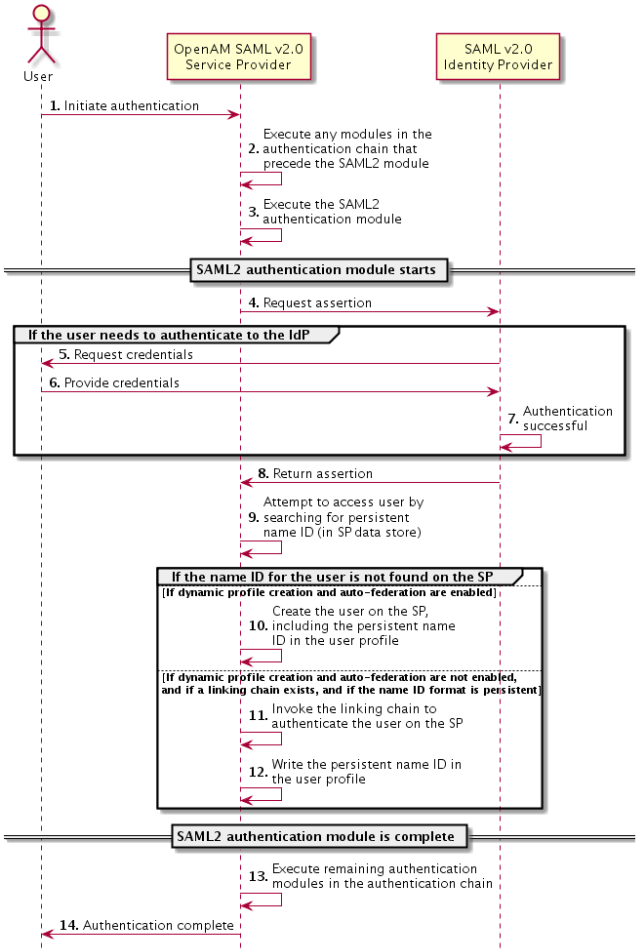
- Section 2.4.2.1, "SAML v2.0 Integrated Mode Authentication Flow"
- Section 2.4.2.2, "SAML v2.0 Integrated Mode Example"
- Section 2.4.2.3, "Implementing SAML v2.0 Single Sign-On in Integrated Mode"

- Section 2.4.2.4, "Configuring Single Logout in an Integrated Mode Implementation"

2.4.2.1. SAML v2.0 Integrated Mode Authentication Flow

The following sequence diagram outlines the flow of SAML v2.0 authentication and persistent federation in an integrated mode implementation:

Figure 2.2. SAML v2.0 Integrated Mode Flow



The following describes the sequence of actions in the diagram:

1. An unauthenticated user initiates authentication to an OpenAM SAML v2.0 service provider. The login URL references an authentication chain that includes a SAML2 authentication module. For example, `http://openam.example.com:8080/openam/XUI/#login/&service=mySAMLChain`.
2. If there are any authentication modules that precede the SAML2 module in the authentication chain, OpenAM executes them.
3. SAML2 authentication module processing begins.
4. The authentication module requests an assertion from the identity provider. The SAML2 module's configuration determines the details of the request.

If the user is currently unauthenticated on the identity provider, the following three steps occur:

5. The identity provider requests credentials from the user.
6. The user provides their credentials.
7. Authentication succeeds (assuming the user provided valid credentials).

Processing continues as follows:

8. The identity provider responds to the service provider with a SAML assertion.
9. If the SAML assertion contains a persistent name ID, OpenAM searches the user datastore, attempting to locate a user with the same name ID.

The flow varies here.

The following event occurs if the name ID for the user is not found in the datastore, if dynamic profile creation is configured in the Core Authentication Service, and if auto-federation is enabled on the service provider:

10. OpenAM adds an entry for the user to the user datastore. Even if a linking authentication chain has been configured, it is not invoked. The user is not prompted to authenticate to the service provider.

The following two events occur if the name ID for the user is not found in the datastore, if a linking authentication chain has been configured in the SAML2 authentication module, if dynamic profile creation is not configured in the Core Authentication Service, and if auto-federation is not enabled on the service provider:

11. The SAML2 authentication module invokes the linking authentication chain, requiring the user to authenticate to the service provider.
12. After successfully completing the linking authentication chain, OpenAM writes the persistent name ID obtained in the SAML assertion sent by the identity provider into the user's profile.

At this point, SAML2 authentication module processing ends. The remaining events comprise completion of the primary authentication chain:

13. If there are any authentication modules remaining in the chain, OpenAM executes them.

14. Authentication is complete.

2.4.2.2. SAML v2.0 Integrated Mode Example

This section describes a SAML v2.0 implementation scenario that provides an example of how you might use integrated mode to satisfy complex authentication requirements.

2.4.2.2.1. Authentication Requirements

The example scenario has the following requirements:

- Users must authenticate with an identity provider using SAML v2.0.
- Users' identities are federated on the identity and service providers.
- Users without federated identities must perform two-step verification at the service provider before their identities can be federated.
- Device fingerprinting for risk-based authentication must be performed for all authenticated users.

2.4.2.2.2. Authentication Chains and Modules

Implementation of the example scenario requires the following authentication chains and authentication modules:

- A primary authentication chain, which implements SAML v2.0 single sign-on and device fingerprinting.

  This chain includes three authentication modules, ordered as follows:

  1. A SAML2 authentication module with the `Required` flag.
  2. A Device ID (Match) authentication module with the `Sufficient` flag.
  3. A Device ID (Save) authentication module with the `Required` flag.

- A linking authentication chain, which identifies the user by user ID and password and requires two-step verification.

  This chain includes two authentication modules, ordered as follows:

  1. A Data Store authentication module with the `Required` flag.
  2. A ForgeRock Authenticator (OATH) authentication module with the `Required` flag.

2.4.2.2.3. How It Works: First Authentication to the Service Provider

This section describes the sequence of events that occurs the first time a user successfully attempts to authenticate to the OpenAM service provider by using the primary authentication chain.

Accessing the service provider. A user authenticates to the OpenAM server acting as a SAML v2.0 service provider, specifying the primary authentication chain in the login URL. For example, `http://www.sp.com:28080/openam/XUI/#login/&service=mySAMLChain`.

Authentication at the identity provider. OpenAM redirects the user to the identity provider. The user authenticates successfully at the identity provider. The identity provider returns a SAML assertion with a persistent name ID to OpenAM.

Service provider attempts to access a federated identity. OpenAM attempts to locate the name ID in its user store. Because this is the first time the user has attempted to authenticate to the OpenAM service provider, the name ID has not yet been associated with any OpenAM user. The search for the name ID fails.

Invocation of the linking chain. Therefore, OpenAM invokes the linking authentication chain. The Data Store authentication module executes first, requiring the user to provide a user ID and password. The ForgeRock Authenticator (OATH) module executes next, requiring the user to provide a one-time password from an authenticator app on the user's mobile device.

Identity federation. OpenAM then writes the name ID into the user's profile in the OpenAM user store. This completes the SAML2 authentication module's processing.

Device fingerprinting (save). Next in sequence is the Device ID (Match) authentication module. Because this is the first time that the user has authenticated to OpenAM, this device profile has not been saved to OpenAM yet and the Device ID (Match) authentication module fails. As a result, control passes to the Device ID (Save) module, which saves the device profile.

2.4.2.2.4. How It Works: Subsequent Authentication to the Service Provider

This section describes the sequence of events that occurs during subsequent successful authentication attempts after the user's identities on the identity and service providers have been federated.

Accessing the service provider. A user authenticates to the OpenAM server acting as a SAML v2.0 service provider specifying the primary authentication chain in the login URL. For example, `http://www.sp.com:28080/openam/XUI/#login/&service=mySAMLChain`.

Authentication at the identity provider. OpenAM redirects the user to the identity provider, and the user authenticates successfully at the identity provider. The identity provider returns a SAML asserting with a persistent name ID to OpenAM.

Service provider attempts to access a federated identity. OpenAM attempts to locate the name ID in its user store. The search for the name ID succeeds. Therefore, OpenAM does not invokes the linking authentication chain.

Device fingerprinting (match). The Device ID (Match) authentication module then executes. Because the user previously authenticated to OpenAM from this device profile, the Device ID (Match) authentication module succeeds, and authentication is complete.

2.4.2.3. Implementing SAML v2.0 Single Sign-On in Integrated Mode

The following list is an overview of the activities you perform when implementing SAML v2.0 SSO in integrated mode:

- Preparing entity providers and a circle of trust.
- Changing several endpoints in the service provider configuration.
- Configuring a SAML2 authentication module and include it in an authentication chain.
- Deciding if and how you want to federate identities during authentication. In integrated mode, you can either create user entries dynamically, or you can configure a linking authentication chain that authenticates users at the service provider after successful authentication at the identity provider, and then federates the identity.

The following procedure provides step-by-step instructions for performing these activities:

Procedure 2.12. To Implement SAML v2.0 SSO in Integrated Mode

1. If you have not already done so, prepare for SAML v2.0 implementation by performing the tasks listed in Section 2.1, "Preparing for Configuring SAML v2.0".
2. Log in to the AM console on the service provider as a top-level administrative user, such as `amadmin`.
3. Create a hosted service provider by following the steps in Procedure 2.2, "To Create a Hosted Service Provider".
4. Configure a remote identity provider by following the steps in Procedure 2.3, "To Configure a Remote Identity Provider". When you specify the circle of trust for the IdP, use the Add to Existing option and specify the circle of trust that you created when you created the hosted service provider.
5. If you want to use dynamic profile creation with auto-federation to federate identities, configure the required options:
   - To configure dynamic profile creation, navigate to Configure > Authentication > Core Attributes.
   - To configure auto-federation, navigate to Realms > *`Realm Name`* > Applications > SAML > Entity Providers > *`Service Provider Name`* > Assertion Processing > Auto Federation.
6. Change the Assertion Consumer Service locations in the service provider configuration. The default locations support standalone mode. Therefore, you must change the locations when implementing integrated mode.

   Change the locations as follows:

   a. In the AM console, navigate to Realms > *`Realm Name`* > Applications > SAML > Entity Providers > *`Service Provider Name`* > Services > Assertion Consumer Service.

b. Change the location of the HTTP-Artifact consumer service to use `AuthConsumer` rather than `Consumer`. For example, if the location is `http://www.sp.com:28080/openam/Consumer/metaAlias/sp`, change it to `http://www.sp.com:28080/openam/AuthConsumer/metaAlias/sp`.

c. Similarly, change the location for the HTTP-POST consumer service to use `AuthConsumer` rather than `Consumer`.

   Note that you do not need to change the location for the PAOS service because integrated mode does not support the PAOS binding.

d. Click Save to save the changes to the endpoints.

7. If the OpenAM server configured as the service provider runs as part of an OpenAM site, enable SAML v2.0 failover. In the AM console, navigate to Configure > Global Services, click SAML v2 Service Configuration, check the Enable SAMLv2 failover checkbox, and then save your changes.

8. Create a SAML2 authentication module:

   a. In the AM console, navigate to Realms > *`Realm Name`* > Authentication > Modules.

   b. Specify a name for the module, and specify the module type as SAML2.

   c. Click Create.

   d. Configure the SAML2 authentication module options. See Section 2.2.23, "SAML2 Authentication Module" in the *Authentication and Single Sign-On Guide* for detailed information about the configuration options.

      If you want to use a linking authentication chain to authenticate users at the service provider and then federate users' identities on the identity and service providers, be sure to specify the name of this chain in the Linking Authentication Chain field.

   e. Save your changes.

9. Create an authentication chain that includes the SAML2 authentication module that you created in the previous step.

10. If you specified a linking authentication chain in the SAML2 module configuration, create the linking chain. A linking chain is an authentication chain that authenticates the user on the service provider, enabling OpenAM to persistently federate a user on the identity and service providers.

11. Test your configuration. First, clear your browser's cache and cookies. Then, attempt to log in to OpenAM using a login URL that references the authentication chain that includes the SAML2 module. For example, `http://www.sp.com:28080/openam/XUI/#login/&service=mySAMLChain`.

    OpenAM should redirect you to the identity provider for authentication. Authenticate to the identity provider.

    If you configured a linking authentication chain, OpenAM should prompt you to authenticate to that chain next. When authentication is complete, try logging out of the service provider,

then navigate to the same login URL that you used earlier. Because you are still logged in at the identity provider, you should not be prompted to reauthenticate to the identity provider. And because your identity at the service provider is now federated with your identity at the identity provider, you should not be prompted to reauthenticate at the service provider either.

2.4.2.4. Configuring Single Logout in an Integrated Mode Implementation

Use the following two options to control single logout in integrated mode:

- The post-authentication processing class for the authentication chain that includes the SAML2 authentication module. You configure post-authentication processing classes by navigating to Realms > *Realm Name* > Authentication > Chains > *Chain Name* > Settings
- The Single Logout Enabled option in the SAML2 authentication module configuration.

Configure these options as follows:

Table 2.3. Configuring Single Logout Options

Requirement	Configuration
Single logout occurs when a user initiates logout at the identity provider or at the service provider.	Set the post-authentication processing class for the authentication chain that contains the SAML2 authentication module to `org.forgerock.openam.authentication.modules.saml2.SAML2PostAuthenticationPlugin`. Set the Single Logout Enabled option to `true` in the SAML2 authentication module configuration.
Single logout occurs only when the user initiates logout at the identity provider.	Set the post-authentication processing class for the authentication chain that contains the SAML2 authentication module to `org.forgerock.openam.authentication.modules.saml2.SAML2PostAuthenticationPlugin`. Set the Single Logout Enabled option to `false` in the SAML2 authentication module configuration.
Single logout occurs only when the user initiates logout at the service provider.	Not available.
Single logout never occurs.	Do not set the post-authentication processing class for the authentication chain that contains the SAML2 authentication module to `org.forgerock.openam.authentication.modules.saml2.SAML2PostAuthenticationPlugin`.

2.4.3. Implementing SAML v2.0 SSO and SLO in Standalone Mode

This section describes how to implement SSO and SLO using standalone mode.

2.4.3.1. Verifying That the Federation Authentication Module Is Present

Standalone mode requires that a Federation authentication module instance is present in the realm in which you define your circle of trust, identity providers, and service providers.

Not only must the module be of type Federation, its *name* must be `Federation` as well.

OpenAM creates a Federation authentication module when you create a new realm, so the required module is already available unless you explicitly deleted it. If you deleted the Federation authentication module and need to restore it to a realm, just create an authentication module named `Federation` of module type Federation. No additional configuration is needed.

Do *not* add the Federation authentication module to an authentication chain. The module is used for internal purposes.

2.4.3.2. JSP Pages for SSO and SLO

With standalone mode, OpenAM SAML v2.0 Federation provides JSPs that let you direct users to do single sign-on (SSO) and single logout (SLO) across providers in a circle of trust. OpenAM has two JSPs for SSO and two JSPs for SLO, allowing you to initiate both processes either from the identity provider side, or from the service provider side.

SSO lets users sign in once and remain authenticated as they access services in the circle of trust.

SLO attempts to log out all session participants:

- For hosted IdPs, SLO attempts to log out of all SPs with which the session established SAML federation.
- For hosted SPs, SLO attempts to log out of the IdP that was source of the assertion for the user's session.

The JSP pages are found under the context root where you deployed OpenAM, in `saml2/jsp/`.

`spSSOInit.jsp`

Used to initiate SSO from the service provider side, so call this on the service provider not the identity provider. This is also mapped to the endpoint `spssoinit` under the context root.

Examples: `http://www.sp.example:8080/openam/saml2/jsp/spSSOInit.jsp`, `http://www.sp.example:8080/openam/spssoinit`

`idpSSOInit.jsp`

Used to initiate SSO from the identity provider side, so call this on the identity provider not the service provider. This is also mapped to the endpoint `idpssoinit` under the context root.

Examples: `http://www.idp.example:8080/openam/saml2/jsp/idpSSOInit.jsp`, `http://www.idp.example:8080/openam/idpssoinit`

`spSingleLogoutInit.jsp`

Used to initiate SLO from the service provider side, so call this on the service provider not the identity provider.

Example: `http://www.sp.example:8080/openam/saml2/jsp/spSingleLogoutInit.jsp`, `http://www.sp.example:8080/openam/SPSloInit`

`idpSingleLogoutInit.jsp`

Used to initiate SLO from the identity provider side, so call this on the identity provider not the service provider.

Example: `http://www.idp.example:8080/openam/saml2/jsp/idpSingleLogoutInit.jsp`, `http://www.idp.example:8080/openam/IDPSloInit`

When you invoke these JSPs, there are several parameters to specify. Which parameters you can use depends on the JSP. When setting parameters in the JSPs, make sure the parameter values are correctly URL-encoded.

idpSSOInit.jsp Parameters

`metaAlias`

(Required) Use this parameter to specify the local alias for the provider, such as `metaAlias=/myRealm/idp`. This parameter takes the format `/realm-name/provider-name` as described in MetaAlias. You do not repeat the slash for the Top Level Realm, for example `metaAlias=/idp`.

`spEntityID`

(Required) Use this parameter to indicate the remote service provider. Make sure you URL-encode the value. For example, specify `spEntityID=http://www.sp.example:8080/openam` as `spEntityID=http%3A%2F%2Fwww.sp.example%3A8080%2Fopenam`.

`affiliationID`

(Optional) Use this parameter to specify a SAML affiliation identifier.

`binding`

(Optional) Use this parameter to indicate what binding to use for the operation. For example, specify `binding=HTTP-POST` to use HTTP POST binding with a self-submitting form. In addition to `binding=HTTP-POST`, you can also use `binding=HTTP-Artifact`.

`NameIDFormat`

(Optional) Use this parameter to specify a SAML Name Identifier format identifier, such as `urn:oasis:names:tc:SAML:2.0:nameid-format:persistent`, or `urn:oasis:names:tc:SAML:2.0:nameid-format:transient`.

`RelayState`

(Optional) Use this parameter to specify where to redirect the user when the process is complete. Make sure you URL-encode the value. For example, `RelayState=http%3A%2F%2Fforgerock.com` takes the user to `http://forgerock.com`.

`RelayStateAlias`

(Optional) Use this parameter to specify the parameter to use as the `RelayState`. For example, if your query string has `target=http%3A%2F%2Fforgerock.com&RelayStateAlias=target`, this is like setting `RelayState=http%3A%2F%2Fforgerock.com`.

spSSOInit.jsp Parameters

`idpEntityID`

(Required) Use this parameter to indicate the remote identity provider. Make sure you URL-encode the value. For example, specify `idpEntityID=http://www.idp.example:8080/openam` as `idpEntityID=http%3A%2F%2Fwww.idp.example%3A8080%2Fopenam`.

`metaAlias`

(Required) Use this parameter to specify the local alias for the provider, such as `metaAlias=/myRealm/sp`. This parameter takes the format */realm-name/provider-name* as described in MetaAlias. You do not repeat the slash for the Top Level Realm, `metaAlias=/sp`.

`affiliationID`

(Optional) Use this parameter to specify a SAML affiliation identifier.

`AllowCreate`

(Optional) Use this parameter to indicate whether the identity provider can create a new identifier for the principal if none exists (`true`) or not (`false`).

`AssertionConsumerServiceIndex`

(Optional) Use this parameter to specify an integer that indicates the location to which the Response message should be returned to the requester.

`AuthComparison`

(Optional) Use this parameter to specify a comparison method to evaluate the requested context classes or statements. OpenAM accepts the following values: `better`, `exact`, `maximum`, and `minimum`.

`AuthnContextClassRef`

(Optional) Use this parameter to specify authentication context class references. Separate multiple values with pipe characters (`|`). When hosted Idp and SP entities are saved in the AM console, any custom authentication contexts are also saved as long as they are included in the extended metadata. You can load custom authentication contexts in the extended metadata using the **ssoadm** command.

`AuthnContextDeclRef`

(Optional) Use this parameter to specify authentication context declaration references. Separate multiple values with pipe characters (`|`).

`AuthLevel`

(Optional) Use this parameter to specify the authentication level of the authentication context that OpenAM should use to authenticate the user.

`binding`

(Optional) Use this parameter to indicate what binding to use for the operation. For example, specify `binding=HTTP-POST` to use HTTP POST binding with a self-submitting form. In addition to `binding=HTTP-POST`, you can also use `binding=HTTP-Artifact`.

`Destination`

(Optional) Use this parameter to specify a URI Reference indicating the address to which the request is sent.

`ForceAuthn`

(Optional) Use this parameter to indicate whether the identity provider should force authentication (`true`) or can reuse existing security contexts (`false`).

`isPassive`

(Optional) Use this parameter to indicate whether the identity provider should authenticate passively (`true`) or not (`false`).

`NameIDFormat`

(Optional) Use this parameter to specify a SAML Name Identifier format identifier, such as `urn:oasis:names:tc:SAML:2.0:nameid-format:persistent`, or `urn:oasis:names:tc:SAML:2.0:nameid-format:transient`.

`RelayState`

(Optional) Use this parameter to specify where to redirect the user when the process is complete. Make sure you URL-encode the value. For example, `RelayState=http%3A%2F%2Fforgerock.com` takes the user to `http://forgerock.com`.

`RelayStateAlias`

(Optional) Use this parameter to specify the parameter to use as the `RelayState`. For example, if your query string has `target=http%3A%2F%2Fforgerock.com&RelayStateAlias=target`, this is like setting `RelayState=http%3A%2F%2Fforgerock.com`.

`reqBinding`

(Optional) Use this parameter to indicate what binding to use for the authentication request. Valid values in include `urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect` (default) and `urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST`.

`sunamcompositeadvice`

(Optional) Use this parameter to specify a URL-encoded XML blob that specifies the authentication level advice. For example, the following XML indicates a requested authentication level of 1. Notice the required `:` before the 1:

```
<Advice>
        <AttributeValuePair>
        <Attribute name="AuthLevelConditionAdvice"/>
        <Value>/:1</Value>
        </AttributeValuePair>
        </Advice>
```

idpSingleLogoutInit.jsp Parameters

`binding`

(Required) Use this parameter to indicate what binding to use for the operation. The full, long name format is required for this parameter to work.

The value must be one of the following:

- `urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect` (default)
- `urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST`
- `urn:oasis:names:tc:SAML:2.0:bindings:SOAP`

`Consent`

(Optional) Use this parameter to specify a URI that is a SAML Consent Identifier.

`Destination`

(Optional) Use this parameter to specify a URI Reference indicating the address to which the request is sent.

`Extension`

(Optional) Use this parameter to specify a list of Extensions as string objects.

`goto`

(Optional) Use this parameter to specify where to redirect the user when the process is complete. `RelayState` takes precedence over this parameter.

`logoutAll`

(Optional) Use this parameter to specify that the identity provider should send single logout requests to service providers without indicating a session index.

`RelayState`

(Optional) Use this parameter to specify where to redirect the user when the process is complete. Make sure you URL-encode the value. For example, `RelayState=http%3A%2F%2Fforgerock.com` takes the user to `http://forgerock.com`.

spSingleLogoutInit.jsp Parameters

`binding`

(Required) Use this parameter to indicate what binding to use for the operation. The full, long name format is required for this parameter to work. For example, specify `binding=urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST` to use HTTP POST binding with a self-submitting form rather than the default HTTP redirect binding. In addition, you can use `binding=urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Artifact`.

`idpEntityID`

(Required for Fedlets) Use this parameter to indicate the remote identity provider. If the `binding` is not set, then OpenAM uses this parameter to find the default binding. Make sure you URL encode the value. For example, specify `idpEntityID=http://www.idp.example:8080/openam` as `idpEntityID=http%3A%2F%2Fwww.idp.example%3A8080%2Fopenam`.

`NameIDValue`

(Required for Fedlets) Use this parameter to indicate the SAML Name Identifier for the user.

`SessionIndex`

(Required for Fedlets) Use this parameter to indicate the `sessionIndex` of the user session to terminate.

`Consent`

(Optional) Use this parameter to specify a URI that is a SAML Consent Identifier.

`Destination`

(Optional) Use this parameter to specify a URI Reference indicating the address to which the request is sent.

`Extension`

(Optional) Use this parameter to specify a list of Extensions as string objects.

`goto`

(Optional) Use this parameter to specify where to redirect the user when the process is complete. `RelayState` takes precedence over this parameter.

`RelayState`

(Optional) Use this parameter to specify where to redirect the user when the process is complete. Make sure you URL-encode the value. For example, `RelayState=http%3A%2F%2Fforgerock.com` takes the user to `http://forgerock.com`.

`spEntityID`

(Optional, for Fedlets) Use this parameter to indicate the Fedlet entity ID. When missing, OpenAM uses the first entity ID in the metadata.

The following URL takes the user from the service provider side to authenticate at the identity provider and then come back to the end user profile page at the service provider after successful SSO. Lines are folded to show you the query string parameters:

```
http://www.sp.example:8080/openam/saml2/jsp/spSSOInit.jsp?metaAlias=/sp
      &idpEntityID=http%3A%2F%2Fwww.idp.example%3A8080%2Fopenam
      &RelayState=http%3A%2F%2Fwww.sp.example%3A8080%2Fopenam%2Fidm%2FEndUser
```

The following URL initiates SLO from the service provider side, leaving the user at `http://forgerock.com`:

```
http://www.sp.example:8080/openam/saml2/jsp/spSingleLogoutInit.jsp?
      &idpEntityID=http%3A%2F%2Fwww.idp.example%3A8080%2Fopenam
      &RelayState=http%3A%2F%2Fforgerock.com
```

Procedure 2.13. To Indicate Progress During SSO

During SSO login, OpenAM presents users with a self-submitting form when access has been validated. This page is otherwise blank. If you want to present users with something to indicate that the operation is in progress, then customize the necessary templates.

1. Modify the templates to add a clue that SSO is in progress, such as an image.

   Edit the source of the OpenAM Java Server Page, `saml2/jsp/autosubmitaccessrights.jsp`, under the file system directory where the OpenAM .war has been unpacked.

   When you add an image or other presentation element, make sure that you retain the form and Java code as is.

2. Unpack OpenAM-14.0.0.war (`OpenAM-14.0.0.war` if you obtained the file by unpacking `OpenAM-14.0.0.zip`), and add your modified template files under `WEB-INF/classes/` where you unpacked the .war.

   Also include any images you reference in the page.

3. Pack up your custom version of OpenAM, and then deploy it in your web container.

2.4.3.3. Using Policy Agents With Standalone Mode

You can use policy agents in a SAML v2.0 Federation deployment.

Procedure 2.14. To Use a Policy Agent with a SAML v2.0 Service Provider

The following procedure applies when OpenAM is configured as an IdP in one domain, and a policy agent protects resources on behalf of a second OpenAM server configured as an SP on a second domain:

1. Install the policy agent.

The basic process for installing policy agents is available in the *Web Policy Agent User's Guide* and the *Java EE Policy Agent User's Guide*.

2. Replace the given OpenAM Login URL and OpenAM Logout URLs with SAML v2.0 URLs described in Section 2.4.3.2, "JSP Pages for SSO and SLO".

   The following steps explain how to do this for web policy agents:

   - If you have configured the Web policy agents to store their properties centralized on an OpenAM server, in the AM console navigate to Realms > *Realm Name* > Applications > Agents > Web > *Agent Name* > OpenAM Services.

     For the Web Agent, under the OpenAM Services tab, in the Agent Logout URL section, set up a list of application logout URLs. In the Logout Redirect URL text box, enter an appropriate URL to redirect the user after logout.

   - Alternatively, if the Web policy agents are set up to store properties on local systems, find the `OpenSSOAgentConfiguration.properties` file in the `/path/to/agent/config/` directory.

     You can specify OpenAM Login and Logout URLs with the `com.sun.identity.agents.config.login.url` and `com.sun.identity.agents.config.logout.url` attributes, respectively.

2.4.3.4. Configuring for the ECP Profile

The SAML v2.0 Enhanced Client or Proxy (ECP) profile is intended for use when accessing services over devices like simple phones, medical devices, and set-top boxes that lack the capabilities needed to use the more widely used SAML v2.0 Web Browser SSO profile.

The ECP knows which identity provider to contact for the user, and is able to use the reverse SOAP (PAOS) SAML v2.0 binding for the authentication request and response. The PAOS binding uses HTTP and SOAP headers to pass information about processing SOAP requests and responses, starting with a PAOS HTTP header that the ECP sends in its initial request to the server. The PAOS messages continue with a SOAP authentication request in the server's HTTP response to the ECP's request for a resource, followed by a SOAP response in an HTTP request from the ECP.

An enhanced client, such as a browser with a plugin or an extension, can handle these communications on its own. An enhanced proxy is an HTTP server, such as a WAP gateway that can support the ECP profile on behalf of client applications.

OpenAM supports the SAML v2.0 ECP profile on the server side for identity providers and service providers. You must build the ECP.

By default, an OpenAM identity provider uses the `com.sun.identity.saml2.plugins.DefaultIDPECPSessionMapper` class to find a user session for requests to the IdP from the ECP. The default session mapper uses OpenAM cookies as it would for any other client application. If for some reason you must change the mapping after writing and installing your own session mapper, you can change the class under Realms > *Realm Name* > Applications > SAML > Entity Providers > *IdP Name* > IDP > Advanced > ECP Configuration.

By default, an OpenAM service provider uses the `com.sun.identity.saml2.plugins.ECPIDPFinder` class to return identity providers from the list under Realms > *Realm Name* > Applications > SAML > Entity Providers > *SP Name* > SP > Advanced > ECP Configuration > Request IDP List. You must populate the list with identity provider entity IDs.

The endpoint for the ECP to contact on the OpenAM service provider is `/SPECP` as in `http://www.sp.example:8080/openam/SPECP`. The ECP provides two query string parameters to identify the service provider and to specify the URL of the resource to access.

`metaAlias`

This specifies the service provider, by default `metaAlias=/realm-name/sp`, as described in MetaAlias.

`RelayState`

This specifies the resource the client aims to access, such as `RelayState=http%3A%2F%2Fforgerock.org%2Findex.html`. Make sure this parameter is correctly URL-encoded.

For example, the URL to access the service provider and finally the resource at `http://forgerock.org/index.html` could be `http://www.sp.example:8080/openam/SPECP?metaAlias=/sp&RelayState=http%3A%2F%2Fforgerock.org%2Findex.html`.

2.4.3.5. Using Transient Federation Identifiers

Identity providers and service providers must be able to communicate about users. Yet, in some cases the identity provider can choose to communicate a minimum of information about an authenticated user, with no user account maintained on the service provider side. In other cases, the identity provider and service provider can choose to link user accounts in a persistent way, in a more permanent way, or even in automatic fashion by using some shared value in the user's profiles, such as an email address or by dynamically creating accounts on the service provider when necessary. OpenAM supports all these alternatives.

OpenAM allows you to link accounts using transient name identifiers, where the identity provider shares a temporary identifier with the service provider for the duration of the user session. Nothing is written to the user profile.

Transient identifiers are useful where the service is anonymous, and all users have similar access on the service provider side.

To use transient name identifiers, specify the name ID format `urn:oasis:names:tc:SAML:2.0:nameid-format:transient` when initiating single sign-on.

The examples below work in an environment where the identity provider is `www.idp.example` and the service provider is `www.sp.example`. Both providers have deployed OpenAM on port 8080 under deployment URI `/openam`.

To initiate single sign-on from the service provider, access the following URL with at least the query parameters shown:

```
http://www.sp.example:8080/openam/saml2/jsp/spSSOInit.jsp?
```

```
     idpEntityID=http%3A%2F%2Fwww.idp.example%3A8080%2Fopenam
     &metaAlias=/sp
     &NameIDFormat=urn:oasis:names:tc:SAML:2.0:nameid-format:transient
```

For a complete list of query parameters, see spSSOInit.jsp Parameters.

To initiate single sign-on from the identity provider, access the following URL with at least the query parameters shown:

```
http://www.idp.example:8080/openam/saml2/jsp/idpSSOInit.jsp?
     spEntityID=http%3A%2F%2Fwww.sp.example%3A8080%2Fopenam
     &metaAlias=/idp
     &NameIDFormat=urn:oasis:names:tc:SAML:2.0:nameid-format:transient
```

For a complete list of query parameters, see idpSSOInit.jsp Parameters.

The accounts are only linked for the duration of the session. Once the user logs out, for example, the accounts are no longer linked.

2.4.3.6. Using Persistent Federation Identifiers

OpenAM lets you use persistent pseudonym identifiers to federate user identities, linking accounts on the identity provider and service provider with a SAML persistent identifier.

Persistent identifiers are useful for establishing links between otherwise unrelated accounts.

The examples below work in an environment where the identity provider is `www.idp.example` and the service provider is `www.sp.example`. Both providers have deployed OpenAM on port 8080 under deployment URI `/openam`.

To initiate single sign-on from the service provider, access the following URL with at least the query parameters shown:

```
http://www.sp.example:8080/openam/saml2/jsp/spSSOInit.jsp?
     idpEntityID=http%3A%2F%2Fwww.idp.example%3A8080%2Fopenam
     &metaAlias=/sp
     &NameIDFormat=urn:oasis:names:tc:SAML:2.0:nameid-format:persistent
```

For a complete list of query parameters, see spSSOInit.jsp Parameters.

To initiate single sign-on from the identity provider, access the following URL with at least the query parameters shown:

```
http://www.idp.example:8080/openam/saml2/jsp/idpSSOInit.jsp?
     spEntityID=http%3A%2F%2Fwww.sp.example%3A8080%2Fopenam
     &metaAlias=/idp
     &NameIDFormat=urn:oasis:names:tc:SAML:2.0:nameid-format:persistent
```

For a complete list of query parameters, see idpSSOInit.jsp Parameters.

On successful login, the accounts are persistently linked, with persistent identifiers stored in the user's accounts on the identity provider and the service provider.

2.5. Managing Federated Accounts

Both integrated and standalone SAML v2.0 implementations allow you to persistently link accounts:

- In integrated mode deployments, you specify the value `urn:oasis:names:tc:SAML:2.0:nameid-format:persistent` in the nameIDFormat field of the SAML2 authentication module.
- In standalone mode, when you initiate single sign-on with either the `spSSOInit.jsp` or `idpSSOInit.jsp` JSP page, you specify the `NameIDFormat=urn:oasis:names:tc:SAML:2.0:nameid-format:persistent` parameter.

This section covers the following topics:

- Section 2.5.1, "Changing Federation of Persistently Linked Accounts"
- Section 2.5.2, "Terminating Federation of Persistently Linked Accounts"
- Section 2.5.3, "Configuring How Remote Accounts Map To Local Accounts"
- Section 2.5.4, "Linking Federated Accounts in Bulk"
- Section 2.5.5, "Authentication and Linked Accounts"

2.5.1. Changing Federation of Persistently Linked Accounts

OpenAM implements the SAML v2.0 Name Identifier Management profile, allowing you to change a persistent identifier that has been set to federate accounts, and also to terminate federation for an account.

When user accounts are stored in an LDAP directory server, name identifier information is stored on the `sun-fm-saml2-nameid-info` and `sun-fm-saml2-nameid-infokey` attributes of a user's entry. [1]

You can retrieve the name identifier value on the IdP side by checking the value of `sun-fm-saml2-nameid-infokey`. For example, if the user's entry in the directory shows `sun-fm-saml2-nameid-infokey: http://www.idp.example:8080/openam|http://www.sp.example:8080/openam| XyfFEsr6Vixbnt0BSqIglLFMGjR2`, then the name identifier on the IdP side is `XyfFEsr6Vixbnt0BSqIglLFMGjR2`.

You can use this identifier to initiate a change request from the service provider as in the following example.

```
http://www.sp.example:8080/openam/saml2/jsp/spMNIRequestInit.jsp?
 idpEntityID=http%3A%2F%2Fwww.idp.example%3A8080%2Fopenam
 &metaAlias=/sp
 &requestType=NewID
 &IDPProvidedID=XyfFEsr6Vixbnt0BSqIglLFMGjR2
```

If desired, you can substitute `openam/SPMniInit` for `openam/saml2/jsp/spMNIRequestInit.jsp`

You can also initiate the change request from the identity provider as in the following example.

[1] To configure these attribute types, in the OpenAM console navigate to Configure > Global Services, and then click SAMLv2 Service Configuration.

```
http://www.idp.example:8080/openam/saml2/jsp/idpMNIRequestInit.jsp?
 spEntityID=http%3A%2F%2Fwww.sp.example%3A8080%2Fopenam
 &metaAlias=/idp
 &requestType=NewID
 &IDPProvidedID=XyfFEsr6Vixbnt0BSqIglLFMGjR2
```

If desired, you can substitute `openam/IDPMniInit` for `openam/saml2/jsp/idpMNIRequestInit.jsp`

You can retrieve the name identifier value on the SP side by checking the value of `sun-fm-saml2-nameid-info`. For example, if the user's entry in the directory shows `sun-fm-saml2-nameid-info: http://www.sp.example:8080/openam| http://www.idp.example:8080/openam| ATo9TSA9Y2Ln7DDrAdO3HFfH5jKD| http://www.idp.example:8080/openam| urn:oasis:names:tc:SAML:2.0:nameid-format:persistent| 9B1OPy3m0ejv3fZYhlqxXmiGD24c| http://www.sp.example:8080/openam| SPRole| false`, then the name identifier on the SP side is `9B1OPy3m0ejv3fZYhlqxXmiGD24c`.

The JSP parameters are listed below. When setting parameters in the JSPs, make sure the parameter values are correctly URL-encoded.

idpMNIRequestInit.jsp Parameters

`spEntityID`

(Required) Use this parameter to indicate the remote service provider. Make sure you URL-encode the value. For example, specify `spEntityID=http://www.sp.example:8080/openam` as `spEntityID=http%3A%2F%2Fwww.sp.example%3A8080%2Fopenam`.

`metaAlias`

(Required) Use this parameter to specify the local alias for the provider, such as `metaAlias=/myRealm/idp`. This parameter takes the format `/realm-name/provider-name` as described in MetaAlias. You do not repeat the slash for the Top Level Realm, for example `metaAlias=/idp`.

`requestType`

(Required) Type of manage name ID request, either `NewID` to change the ID, or `Terminate` to remove the information that links the accounts on the identity provider and service provider.

`SPProvidedID`

(Required if `requestType=NewID`) Name identifier in use as described above.

`affiliationID`

(Optional) Use this parameter to specify a SAML affiliation identifier.

`binding`

(Optional) Use this parameter to indicate what binding to use for the operation. The full, long name format is required for this parameter to work.

The value must be one of the following:

- `urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST`

- `urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect`
- `urn:oasis:names:tc:SAML:2.0:bindings:SOAP`

`relayState`

(Optional) Use this parameter to specify where to redirect the user when the process is complete. Make sure you URL-encode the value. For example, `relayState=http%3A%2F%2Fforgerock.com` takes the user to `http://forgerock.com`.

spMNIRequestInit.jsp Parameters

`idpEntityID`

(Required) Use this parameter to indicate the remote identity provider. Make sure you URL-encode the value. For example, specify `idpEntityID=http://www.idp.example:8080/openam` as `idpEntityID=http%3A%2F%2Fwww.idp.example%3A8080%2Fopenam`.

`metaAlias`

(Required) Use this parameter to specify the local alias for the provider, such as `metaAlias=/myRealm/sp`. This parameter takes the format */realm-name/provider-name* as described in MetaAlias. You do not repeat the slash for the Top Level Realm, `metaAlias=/sp`.

`requestType`

(Required) Type of manage name ID request, either `NewID` to change the ID, or `Terminate` to remove the information that links the accounts on the identity provider and service provider.

`IDPProvidedID`

(Required if `requestType=NewID`) Name identifier in use as described above.

`affiliationID`

(Optional) Use this parameter to specify a SAML affiliation identifier.

`binding`

(Optional) Use this parameter to indicate what binding to use for the operation. The full, long name format is required for this parameter to work.

The value must be one of the following:

- `urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST`
- `urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect`
- `urn:oasis:names:tc:SAML:2.0:bindings:SOAP`

`relayState`

(Optional) Use this parameter to specify where to redirect the user when the process is complete. Make sure you URL-encode the value. For example, `relayState=http%3A%2F%2Fforgerock.com` takes the user to `http://forgerock.com`.

You can terminate federation as described in Section 2.5.2, "Terminating Federation of Persistently Linked Accounts".

2.5.2. Terminating Federation of Persistently Linked Accounts

OpenAM lets you terminate account federation, where the accounts have been linked with a persistent identifier as described in Section 2.4.3.6, "Using Persistent Federation Identifiers".

The examples below work in an environment where the identity provider is `www.idp.example` and the service provider is `www.sp.example`. Both providers have deployed OpenAM on port 8080 under deployment URI `/openam`.

To initiate the process of terminating account federation from the service provider, access the following URL with at least the query parameters shown.

```
http://www.sp.example:8080/openam/saml2/jsp/spMNIRequestInit.jsp?
 idpEntityID=http%3A%2F%2Fwww.idp.example%3A8080%2Fopenam
 &metaAlias=/sp
 &requestType=Terminate
```

To initiate the process of terminating account federation from the identity provider, access the following URL with at least the query parameters shown.

```
http://www.idp.example:8080/openam/saml2/jsp/idpMNIRequestInit.jsp?
 spEntityID=http%3A%2F%2Fwww.sp.example%3A8080%2Fopenam
 &metaAlias=/idp
 &requestType=Terminate
```

2.5.3. Configuring How Remote Accounts Map To Local Accounts

OpenAM lets you configure the service provider to link an account based on an attribute value from the identity provider. When you know the user accounts on both the identity provider and the service provider share a common attribute value, such as an email address or other unique user identifier, you can use this method to link accounts without user interaction. See Procedure 2.15, "To Map Accounts Based on an Attribute Value".

OpenAM also lets you map users on the identity provider temporarily to a single anonymous user account on the service provider, in order to exchange attributes about the user without a user-specific account on the service provider. This approach can be useful when the service provider either needs no user-specific account to provide a service, or when you do not want to retain a user profile on the service provider but instead you make authorization decisions based on attribute values from the identity provider. See Procedure 2.16, "To Map Remote Accounts to a Single-Service Provider Account".

OpenAM further allows you to use attributes from the identity provider to create accounts dynamically on the service provider. When using this method, you should inform the user and obtain consent to create the account if necessary. See Procedure 2.17, "To Map Accounts With Dynamic Service Provider Account Creation".

Procedure 2.15. To Map Accounts Based on an Attribute Value

The following steps demonstrate how to map accounts based on an attribute value that is the same in both accounts.

Perform the following steps on the hosted identity provider(s), and again on the hosted service provider(s):

1. Log in to the AM console as administrator.
2. Navigate to Realms > *`Realm Name`* > Applications > SAML > Entity Providers *`Hosted Provider Name`* > Assertion Processing.
3. If the attribute to use when linking accounts is not yet included in the attribute map, add the attribute mapping, and then save your work.
4. On the hosted service provider, under Auto Federation, select Enabled and enter the local attribute name in the Attribute field, and then save your work.

Procedure 2.16. To Map Remote Accounts to a Single-Service Provider Account

The following steps demonstrate how to auto-federate using a single anonymous user account on the service provider.

Perform the following steps on the hosted identity provider(s), and again on the hosted service provider(s):

1. Log in to the AM console as administrator.
2. Navigate to Realms > *`Realm Name`* > Applications > SAML > Entity Providers > *`Hosted Provider Name`* > Assertion Processing.
3. If you want to get attributes from the identity provider and the attributes are not yet in the attribute map, add the attribute mappings, and then save your work.
4. On the hosted service provider, under Transient User, set the single account to which to map all users, such as `anonymous`, and then save your work.
5. After completing configuration on the providers, use transient identifiers to federate as described in Section 2.4.3.5, "Using Transient Federation Identifiers".

Procedure 2.17. To Map Accounts With Dynamic Service Provider Account Creation

The following steps demonstrate how to map accounts with dynamic creation of missing accounts on the service provider side:

1. Set up a mapping based on an attribute value as described in Procedure 2.15, "To Map Accounts Based on an Attribute Value". The attributes you map from the identity provider are those that the service provider sets on the dynamically created accounts.

2. On the service provider console, navigate to Realms > *Realm Name* > Authentication > Settings > User Profile. For the User Profile field, select Dynamic or Dynamic with User Alias, which are described in Section 11.1.3, "User Profile" in the *Authentication and Single Sign-On Guide*, and then save your work.

3. To test your work, create a user on the identity provider, log out of the AM console, and initiate SSO by logging in as the user you created.

   To initiate SSO, navigate to one of the OpenAM SAML v2.0 JSPs with the appropriate query parameters. The following is an example URL for service provider initiated SSO.

   ```
   http://www.sp.example:8080/openam/saml2/jsp/spSSOInit.jsp?
    idpEntityID=http%3A%2F%2Fwww.idp.example%3A8080%2Fopenam
    &metaAlias=/sp
   ```

   On success, check `http://www.sp.example:8080/openam/idm/EndUser` to see the new user account.

2.5.4. Linking Federated Accounts in Bulk

If you manage both the identity provider and service provider, you can link accounts out-of-band, in bulk. You make permanent connections for a list of identity provider and service provider by using the **ssoadm** bulk federation commands.

Before you can run the bulk federation commands, first establish the relationship between accounts, set up the providers as described in Section 2.3, "Configuring Identity Providers, Service Providers, and Circles of Trust", and install the **ssoadm** command as described in Section 2.3.1, "Setting up Administration Tools" in the *Installation Guide*.

To understand the relationships between accounts, consider an example where the identity provider is at `idp.example.org` and the service provider is at `sp.example.com`. A demo user account has the Universal ID, `id=demo,ou=user,dc=example,dc=org`, on the identity provider. That maps to the Universal ID, `id=demo,ou=user,dc=example,dc=com`, on the service provider.

The **ssoadm** command then needs a file that maps local user IDs to remote user IDs, one per line, separated by the vertical bar character `|`. Each line of the file appears as follows:

```
local-user-ID|remote-user-ID
```

In the example, starting on the service provider side, the line for the demo user reads as follows.

```
id=demo,ou=user,dc=example,dc=com|id=demo,ou=user,dc=example,dc=org
```

All the user accounts mapped in your file must exist at the identity provider and the service provider when you run the commands to link them.

Link the accounts using the **ssoadm** bulk federation commands:

1. Prepare the data with the **ssoadm do-bulk-federation** command.

   The following example starts on the service provider side:

```
$ cat /tmp/user-map.txt
id=demo,ou=user,dc=example,dc=com|id=demo,ou=user,dc=example,dc=org
$ ssoadm  \
 do-bulk-federation \
 --metaalias /sp \
 --remoteentityid http://idp.example.org:8080/openam \
 --useridmapping /tmp/user-map.txt \
 --nameidmapping /tmp/name-map.txt \
 --adminid amadmin  \
 --password-file /tmp/pwd.txt \
 --spec saml2

Bulk Federation for this host was completed. To complete the
federation, name Id mapping file should be loaded to remote
provider.
```

2. Copy the name ID mapping output file to the other provider:

```
$ scp /tmp/name-map.txt openam@idp.example.org:/tmp/name-map.txt
openam@idp.example.org's password:
name-map.txt                    100%  177     0.2KB/s   00:00
```

3. Import the name ID mapping file with the **ssoadm import-bulk-fed-data** command.

   The following example is performed on the identity provider side:

```
$ ssoadm \
 import-bulk-fed-data \
 --adminid amadmin \
 --password-file /tmp/pwd.txt \
 --metaalias /idp \
 --bulk-data-file /tmp/name-map.txt

Bulk Federation for this host was completed.
```

At this point the accounts are linked.

2.5.5. Authentication and Linked Accounts

In a SAML v2.0 federation where accounts are durably linked, authentication is required only on the identity provider side.

Authentication is also required, however, on the service provider side when the OpenAM account mapper on the service provider is not able to map the user identified in the SAML assertion from the identity provider to a local user account.

This can happen, for example, the first time accounts are linked as described in Section 2.4.3.6, "Using Persistent Federation Identifiers", after which the persistent identifier establishes the mapping.

This also happens when transient identifiers are used to map accounts. When accounts are linked as described in Section 2.4.3.5, "Using Transient Federation Identifiers", then the service provider must

locally authenticate the user for every SAML assertion received. This is because the identifier used to link the accounts is transient; it does not provide a durable means to link the accounts.

2.6. SAML v2.0 and Session State

ForgeRock recommends that you configure OpenAM to use stateful sessions when deploying OpenAM as a SAML v2.0 IdP or SP.

The following SAML v2.0 profiles are not supported when you configure OpenAM to use stateless sessions:

- Single logout
- Single sign-on with the HTTP POST binding

The other SAML v2.0 single sign-on profiles are not guaranteed to work with stateless sessions.

For more information about stateful and stateless sessions, see Section 1.8.1, "Session State" in the *Authentication and Single Sign-On Guide*.

Chapter 3

Implementing SAML v2.0 Service Providers Using the Fedlet

This chapter introduces OpenAM Fedlets, and shows how to use the Fedlet as part of your Java application.

An OpenAM *Fedlet* is a small web application that that acts as a lightweight SAML v2.0 service provider.

Fedlets are easy to integrate into Java web applications. The Fedlet does not require an entire OpenAM installation alongside your application, but instead can redirect to OpenAM for single sign-on, and to retrieve SAML assertions.

Review Table 1.1, "Fedlet Support for SAML v2.0 Features" to help you decide if the Fedlet is an appropriate solution for your deployment. The table describes the capabilities available in Java Fedlets.

3.1. Using Fedlets in Java Web Applications

This section introduces OpenAM Fedlets and shows how to use the Fedlet as part of your Java web application.

3.1.1. Creating and Installing a Java Fedlet

The following sections provide procedures for creating, installing, configuring, and testing a Java Fedlet to perform single sign-on and single logout with an identity provider:

- Section 3.1.1.1, "Generating the Fedlet Configuration on the Identity Provider"
- Section 3.1.1.2, "Installing and Configuring the Fedlet on the Service Provider"
- Section 3.1.1.3, "Testing Fedlet Single Sign-on and Single Logout"

You can also use the Fedlet to query attributes of users on identity providers configured with the Attribute Authority (`AttrAuth`) and the XACML Policy Decision Point (`XACML PDP`) types. See the following sections for additional configuration requirements:

- Section 3.1.1.4, "Querying an Attribute Authority"
- Section 3.1.1.5, "Querying an XACML Policy Decision Point"

3.1.1.1. Generating the Fedlet Configuration on the Identity Provider

Perform the following steps on the identity provider to create a `Fedlet.zip` file containing the configuration files for the Java fedlet:

Procedure 3.1. To Create Configuration Files for a Fedlet

1. If you have not already done so, create a hosted identity provider, using the test certificate for the signing key.

   For information about how to create a hosted identity provider, see Section 2.3.1, "Creating a Hosted Identity Provider".

2. Under Realms > *`Realm Name`* > Dashboard, click Create Fedlet Configuration, and then click Create Fedlet Configuration a second time.

   Note that the circle of trust includes your hosted identity provider.

3. Name the Fedlet, and set the Destination URL.

   You can use the deployment URL, such as `http://openam.example.com:8080/fedlet` as both the name and the destination URL.

4. Click Create to generate the `Fedlet.zip` file.

   A message appears indicating Fedlet creation was successful. Note the location of the generated output file in the message. For example, `/path/to/openam/config/myfedlets/httpopenamexamplecom8080fedlet/Fedlet.zip`.

5. Click OK to close the message informing you that the Fedlet was created.

6. Transfer the `Fedlet.zip` file to the service provider administrator.

3.1.1.2. Installing and Configuring the Fedlet on the Service Provider

Having obtained the `Fedlet.zip` file from the identity provider administrator, the service provider adminsitrator unzips and installs the file, and then installs the `fedlet.war` file from the OpenAM distribution:

Procedure 3.2. To Install and Configure the Fedlet as a Demo Application

1. Create the `fedlet` directory in the home directory of the user that runs the OpenAM web container:

   ```
   $ cd $HOME
   $ mkdir fedlet
   ```

2. Copy the `Fedlet.zip` file obtained from the identity provider administrator to the `$HOME/fedlet` directory.

3. Unzip the `Fedlet.zip` file.
4. Move all the files under `$HOME/fedlet/conf` to `$HOME/fedlet`.
5. Obtain the `Fedlet-14.0.0.zip` file from the full OpenAM distribution file, `OpenAM-14.0.0.zip`.
6. Unzip the `Fedlet-14.0.0.zip` file:

```
$ cd /path/to/openam-distribution/openam
$ unzip Fedlet-14.0.0.zip
```

7. Deploy the Fedlet in your web container:

```
$ cp /path/to/openam-distribution/openam/fedlet.war /path/to/tomcat/webapps
```

3.1.1.3. Testing Fedlet Single Sign-on and Single Logout

To test single sign-on and single logout from the Fedlet, browse to the Fedlet URL. For example, `http://openam.example.com:8080/fedlet`.

Try one or more examples from the Fedlet home page:

You can log in to the identity provider with username `demo` and password `changeit`.

3.1.1.4. Querying an Attribute Authority

You can use the Fedlet to query attributes on an identity provider. The identity provider must be configured with the Attribute Authority (`AttrAuth`) type and should sign responses. The Fedlet must be configured to deal with signed responses. Furthermore, map the attributes to request in both the identity provider and the Fedlet configurations.

Perform the following steps:

Procedure 3.3. To Use the Fedlet to Query an Attribute Authority

1. Add the Attribute Authority type to the hosted identity provider.

a. On OpenAM where the identity provider is hosted, log in to the AM console as an administrator, such as `amadmin`.

b. Under Realms > *`Realm Name`* > Applications > SAML > Entity Providers, select the New button even though you plan to change the configuration rather than create a new provider.

c. Select the protocol of the provider: `SAMLv2`.

d. In the Create SAMLv2 Entity Provider page, do the following.

- Set the Realm.
- Set the Entity Identifier to the same entity identifier you selected in the previous screen.
- In the Attribute Authority section, set the Meta Alias for example to `/attra`, and set the Signing certificate alias and Encryption certificate alias values to `test` (to use the test certificate).
- Click Create to save your changes.

  Disregard any error messages stating that the entity already exists.

`AttrAuth` now appears in the list of Types for your identity provider.

2. Under Realms > *`Realm Name`* > Applications > SAML > Entity Providers, select the identity provider's name to open the provider's configuration.

3. Make sure attributes for the query are mapped on the Identity Provider.

   Under IDP > Attribute Mapper, add the following values to the Attribute Map if they are not yet present.

   - `cn=cn`
   - `sn=sn`
   - `uid=uid`

   Note

   Make sure to use thread-safe code if you implement the AttributeAuthorityMapper. You can use the attributes on the HttpRequest instead of synchronizing them. The default AttributeAuthorityMapper uses an attribute on the HttpServletRequest to pass information to the AttributeQueryUtil.

   Click Save to save your changes.

4. Generate the Fedlet configuration files as described in Procedure 3.1, "To Create Configuration Files for a Fedlet", making sure you map the attributes.

- `cn=cn`
- `sn=sn`
- `uid=uid`

This step creates a Fedlet configuration with updated identity provider metadata. If you already created a Fedlet, either use a different name, or delete the existing Fedlet.

5. Deploy the new Fedlet as described in Procedure 3.2, "To Install and Configure the Fedlet as a Demo Application".

6. Edit the new Fedlet configuration to request signing and encryption, and replace the existing configuration in OpenAM with the edited configuration.

   a. Copy the test keystore from OpenAM, and prepare password files.

   ```
   $ scp user@openam:/home/user/openam/keystore.jks ~/fedlet/
   ```

   The Fedlet uses password files when accessing the keystore. These password files contain encoded passwords, where the encoding is specific to the Fedlet.

   To encode the password, use `fedletEncode.jsp`. `fedletEncode.jsp` is in the deployed Fedlet, for example `http://openam.example.com:8080/fedlet/fedletEncode.jsp`. The only password to encode for OpenAM's test keystore is `changeit`, because the keystore and private key passwords are both the same.

   Use the encoded value to create the password files as in the following example.

   ```
   $ echo AQIC5BHNSjLwT303GqndmHbyYvzP9Tz7OAnK > ~/fedlet/.storepass
   $ echo AQIC5BHNSjLwT303GqndmHbyYvzP9Tz7OAnK > ~/fedlet/.keypass
   ```

   b. Edit `~/fedlet/sp.xml`.

   To use the test certificate for the attribute query feature, add a `RoleDescriptor` to the `EntityDescriptor` after the `SSODescriptor`. The `RoleDescriptor` describes the certificates that are used for signing and encryption. The attribute authority encrypts the response with the Fedlet's public key, and the Fedlet decrypts the response with its private key.

   Change the following:

   ```
   <RoleDescriptor xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:query="urn:oasis:names:tc:SAML:metadata:ext:query"
       xsi:type="query:AttributeQueryDescriptorType"
       protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol">
    </RoleDescriptor>
   ```

   To:

   ```
   <RoleDescriptor xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   ```

```
    xmlns:query="urn:oasis:names:tc:SAML:metadata:ext:query"
    xsi:type="query:AttributeQueryDescriptorType"
    protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol">
  <KeyDescriptor use="signing">
   <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <ds:X509Data>
     <ds:X509Certificate>
MIICQDCCAakCBEeNB0swDQYJKoZIhvcNAQEEBQAwZzELMAkGA1UEBhMCVVMxEzARBgNVBAgTCkNh
bGlmb3JuaWExFDASBgNVBAcTC1NhbnRhIENsYXJhMQwwCgYDVQQKEwNTdW4xEDAOBgNVBAsTB09w
ZW5TU08xDTALBgNVBAMTBHRlc3QwHhcNMDgwMTE1MTkxOTM5WhcNMTgwMTEyMTkxOTM5WjBnMQsw
CQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcm5pYTEUMBIGA1UEBxMLU2FudGEgQ2xhcmExDDAK
BgNVBAoTA1N1bjEQMA4GA1UECxMHT3BlblNTTzENMAsGA1UEAxMEdGVzdDCBnzANBgkqhkiG9w0B
AQEFAAOBjQAwgYkCgYEArSQc/U75GB2AtKhbGS5piiLkmJzqEsp64rDxbMJ+xDrye0EN/q1U5Of+
RkDsaN/igkAvV1cuXEgTL6RlafFPcUX7QxDhZBhsYF9pbwtMzi4A4su9hnxIhURebGEmxKW9qJNY
Js0Vo5+IgjxuEWnjnnVgHTs1+mq5QYTA7E6ZyL8CAwEAATANBgkqhkiG9w0BAQQFAAOBgQB3Pw/U
QzPKTPTYi9upbFXlrAKMwtFf2OW4yvGWWvlcwcNSZJmTJ8ARvVYOMEVNbsT4OFcfu2/PeYoAdiDA
cGy/F2Zuj8XJJpuQRSE6PtQqBuDEHjjmOQJ0rV/r8mO1ZCtHRhpZ5zYRjhRC9eCbjx9VrFax0JDC
/FfwWigmrW0Y0Q==
     </ds:X509Certificate>
    </ds:X509Data>
   </ds:KeyInfo>
  </KeyDescriptor>
  <KeyDescriptor use="encryption">
   <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <ds:X509Data>
     <ds:X509Certificate>
MIICQDCCAakCBEeNB0swDQYJKoZIhvcNAQEEBQAwZzELMAkGA1UEBhMCVVMxEzARBgNVBAgTCkNh
bGlmb3JuaWExFDASBgNVBAcTC1NhbnRhIENsYXJhMQwwCgYDVQQKEwNTdW4xEDAOBgNVBAsTB09w
ZW5TU08xDTALBgNVBAMTBHRlc3QwHhcNMDgwMTE1MTkxOTM5WhcNMTgwMTEyMTkxOTM5WjBnMQsw
CQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcm5pYTEUMBIGA1UEBxMLU2FudGEgQ2xhcmExDDAK
BgNVBAoTA1N1bjEQMA4GA1UECxMHT3BlblNTTzENMAsGA1UEAxMEdGVzdDCBnzANBgkqhkiG9w0B
AQEFAAOBjQAwgYkCgYEArSQc/U75GB2AtKhbGS5piiLkmJzqEsp64rDxbMJ+xDrye0EN/q1U5Of+
RkDsaN/igkAvV1cuXEgTL6RlafFPcUX7QxDhZBhsYF9pbwtMzi4A4su9hnxIhURebGEmxKW9qJNY
Js0Vo5+IgjxuEWnjnnVgHTs1+mq5QYTA7E6ZyL8CAwEAATANBgkqhkiG9w0BAQQFAAOBgQB3Pw/U
QzPKTPTYi9upbFXlrAKMwtFf2OW4yvGWWvlcwcNSZJmTJ8ARvVYOMEVNbsT4OFcfu2/PeYoAdiDA
cGy/F2Zuj8XJJpuQRSE6PtQqBuDEHjjmOQJ0rV/r8mO1ZCtHRhpZ5zYRjhRC9eCbjx9VrFax0JDC
/FfwWigmrW0Y0Q==
     </ds:X509Certificate>
    </ds:X509Data>
   </ds:KeyInfo>
   <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc">
    <xenc:KeySize xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
      >128</xenc:KeySize>
   </EncryptionMethod>
  </KeyDescriptor>
 </RoleDescriptor>
```

c. Edit `~/fedlet/sp-extended.xml` to use the test certificate for the attribute query.

Change the following, assuming your circle of trust is called `cot`:

```
<AttributeQueryConfig metaAlias="/attrQuery">
    <Attribute name="signingCertAlias">
        <Value></Value>
    </Attribute>
    <Attribute name="encryptionCertAlias">
        <Value></Value>
    </Attribute>
    <Attribute name="wantNameIDEncrypted">
        <Value></Value>
    </Attribute>
    <Attribute name="cotlist">
        <Value>cot</Value>
    </Attribute>
</AttributeQueryConfig>
```

To:

```
<AttributeQueryConfig metaAlias="/attrQuery">
    <Attribute name="signingCertAlias">
        <Value>test</Value>
    </Attribute>
    <Attribute name="encryptionCertAlias">
        <Value>test</Value>
    </Attribute>
    <Attribute name="wantNameIDEncrypted">
        <Value>true</Value>
    </Attribute>
    <Attribute name="cotlist">
        <Value>cot</Value>
    </Attribute>
</AttributeQueryConfig>
```

d. In the AM console, under Realms > *Realm Name* > Applications > SAML > Entity Providers, delete the configuration for your new Fedlet.

e. Make a copy of `sp-extended.xml` called `sp-extended-copy.xml` and set `hosted="0"` in the root element of the copy.

   Use the copy, `sp-extended-copy.xml`, when importing the Fedlet configuration into OpenAM. OpenAM must register the Fedlet as a *remote* service provider.

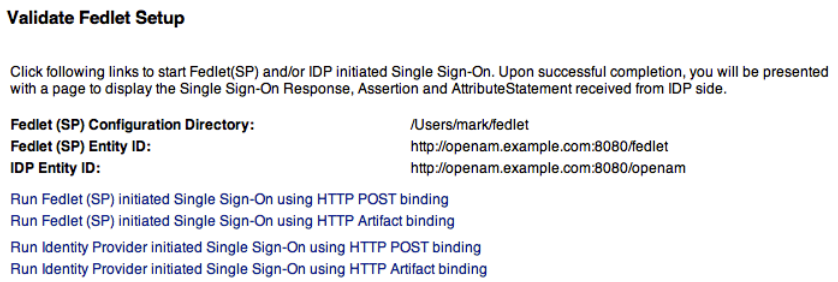
f. Under Realms > *Realm Name* > Applications > SAML > Entity Providers, click the Import Entity button and import your updated Fedlet configuration.

   This ensures OpenAM has the correct service provider configuration for your new Fedlet.

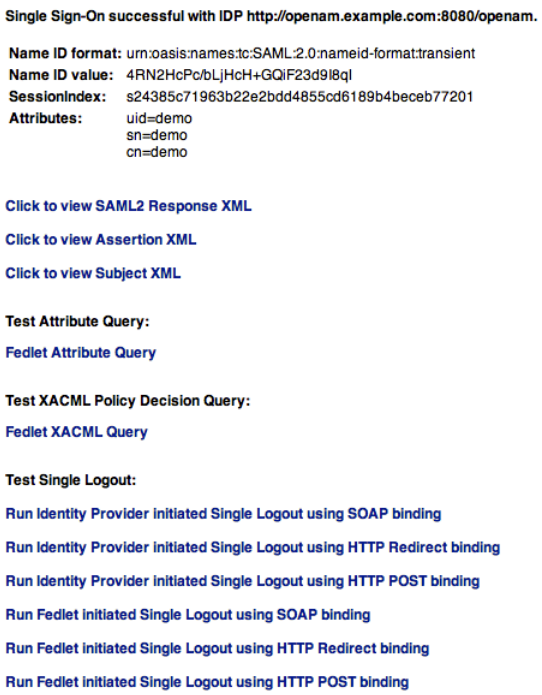
g. Restart the Fedlet or the container where it is deployed.

7. Try the Attribute Query test.

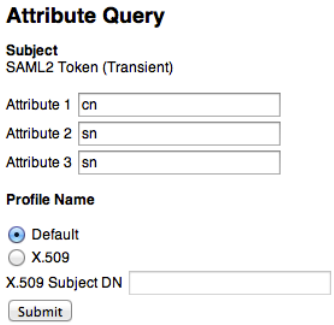
   a. Access the Fedlet.



b. Try SSO with username `demo`, password `changeit`.



c. Click Fedlet Attribute Query, set the attributes in the Attribute Query page to match the mapped attributes, and then click Submit.



d. Check that you see the attribute values in the response.

Fedlet Attribute Query Response

Attribute Query	Attribute Response
uid	demo
sn	demo
cn	demo

3.1.1.5. Querying an XACML Policy Decision Point

You can use the Fedlet to query an XACML policy decision point on an identity provider. The identity provider must have a policy configured, must be configured with the Policy Decision Point (`XACML PDP`) type, and must have a SAML v2.0 SOAP Binding PDP handler configured.

Perform the following steps:

Procedure 3.4. To Use the Fedlet to Query an XACML Policy Decision Point

1. Configure a policy on the hosted identity provider.

   OpenAM uses the policy to make the decision whether to permit or deny access to a resource. For the purpose of the demonstration, configure a simple policy that allows all authenticated users HTTP GET access on `http://www.example.com/`.

   a. Log in to the AM console as an administrator, such as `amadmin`.

   b. Access the policy editor under Realms > *`Realm Name`* > Authorization.

   c. Choose an application that allows the resource pattern `http://www.example.com/*`, and HTTP GET as an action.

      If no application exists in the realm, add a new application for the resource pattern `http://www.example.com/*`.

d. Add a new policy with the following characteristics.

- Resource pattern: `http://www.example.com/*`
- Actions: allow `GET`
- Subject conditions: `Authenticated Users`

2. Add the Policy Decision Point type to the identity provider.

a. Under Realms > *`Realm Name`* > Applications > SAML > Entity Providers, click the New button even though you plan to change the configuration rather than create a new provider.

b. Select the protocol of the provider: `SAMLv2`.

c. In the Create SAMLv2 Entity Provider page, do the following.

- Set the Realm.
- Set the Entity Identifier to the entity identifier for the hosted identity provider.
- In the XACML Policy Decision Point section, set the Meta Alias for example to `/pdp`.
- Click Create to save your changes.

  Disregard any error messages stating that the entity already exists.

`XACML PDP` now appears in the list of Types for your identity provider.

3. Add the PDP handler for the SAML v2.0 SOAP Binding.

a. Navigate to Configure > Global Services, click SAMLv2 SOAP Binding, and then click New.

b. Set the new key to match the meta alias you used when adding the XACML PDP type to the identity provider configuration, for example `/pdp`.

- Key: `/pdp`
- Class: `com.sun.identity.xacml.plugins.XACMLAuthzDecisionQueryHandler`

Click OK. (Your changes are not saved yet.)

c. Click Save to actually save the new Key:Class pair.

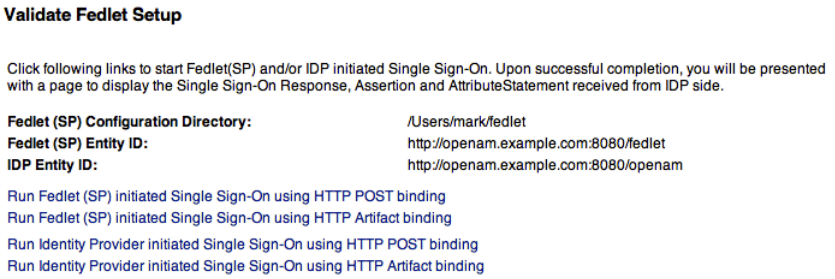
4. Create the Fedlet's configuration files as described in Procedure 3.1, "To Create Configuration Files for a Fedlet".

This step creates Fedlet configuration files with updated identity provider metadata. If you already created a Fedlet, either use a different name, or delete the existing Fedlet.

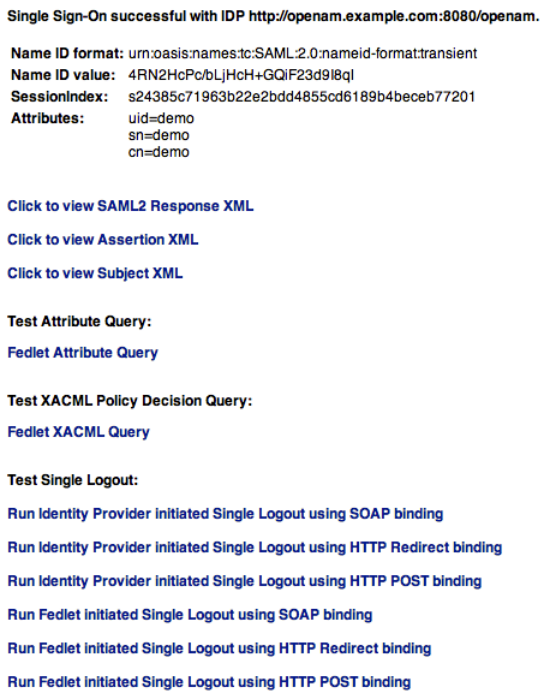
5. Deploy the new Fedlet as described in Procedure 3.2, "To Install and Configure the Fedlet as a Demo Application".

6. Try the XACML Query test.

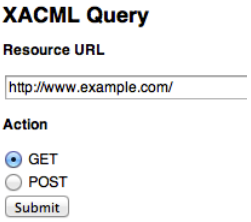
   a. Access the Fedlet.



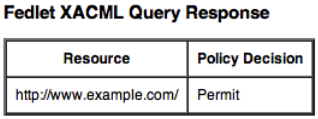
   b. Try SSO with username `demo`, password `changeit`.



   c. Click XACML Attribute Query, set the Resource URL in the XACML Query page to `http://www.example.com/`, and then click Submit.



d. Check that you see the permit decision in the response.



Fedlet XACML Query Response

Resource	Policy Decision
http://www.example.com/	Permit

3.1.2. Enabling Signing and Encryption in a Fedlet

By default when you create the Java Fedlet, signing and encryption are not configured. You can however set up OpenAM and the Fedlet to sign and to verify XML signatures and to encrypt and to decrypt data such as SAML assertions. If you have tried the Attribute Query demonstration, then you have already configured the Fedlet to request signing and encryption using the test keys from the identity provider.

Enabling signing and encryption for the Java Fedlet involves the following high level stages:

- Before you create the Fedlet, configure the IdP to sign and encrypt data. See Realms > *`Realm Name`* > Applications > SAML > Entity Providers > *`IdP Name`* > Signing and Encryption in the AM console.

  For evaluation, you can use the `test` certificate delivered with OpenAM.

- Initially deploy and configure the Fedlet, but do not use the Fedlet until you finish.

- On the Fedlet side set up a JKS keystore used for signing and encryption. For evaluation, you can use copy the `keystore.jks` file delivered with OpenAM. You can find the file under the configuration directory for OpenAM, such as `$HOME/openam/` for a server instance with base URI `openam`. The built-in keystore includes the `test` certificate.

  You must also set up `.storepass` and `.keypass` files using the `fedletEncode.jsp` page, such as `http://openam.example.com:8080/fedlet/fedletEncode.jsp`, to encode passwords on the Fedlet side. The passwords for the test keystore and private key are both `changeit`.

- Configure the Fedlet to perform signing and encryption by ensuring the Fedlet has access to the keystore, and by updating the SP metadata for the Fedlet.

- Import the updated SP metadata into the IdP to replace the default Fedlet configuration.

- Restart the Fedlet or container in which the Fedlet runs for the changes you made on the Fedlet side to take effect.

Procedure 3.5. To Configure the Fedlet For Signing & Encryption

The `FederationConfig.properties` file specifies the paths to the JKS keystore holding the signing or encryption keys for the Fedlet, the keystore password file, and the private key password file.

1. After setting up your keystore and password files as described above, edit the properties file in the configuration directory, such as `$HOME/fedlet/FederationConfig.properties`, to point to the keystore and password files.

2. Export the certificate to use for signing and encryption purposes.

```
$ keytool -export -rfc -keystore keystore.jks -alias test
Enter keystore password:
-----BEGIN CERTIFICATE-----
MIICQDCCAakCBEeNB0swDQYJKoZIhvcNAQEEBQAwZzELMAkGA1UEBhMCVVMxEzARBgNVBAgTCkNh
bGlmb3JuaWExFDASBgNVBAcTC1NhbnRhIENsYXJhMQwwCgYDVQQKEwNTdW4xEDAOBgNVBAsTB09w
ZW5TU08xDTALBgNVBAMTBHRlc3QwHhcNMDgwMTE1MTkxOTM5WhcNMTgwMTEyMTkxOTM5WjBnMQsw
CQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcm5pYTEUMBIGA1UEBxMLU2FudGEgQ2xhcmExDDAK
BgNVBAoTA1N1bjEQMA4GA1UECxMHT3BlblNTTzENMAsGA1UEAxMEdGVzdDCBnzANBgkqhkiG9w0B
AQEFAAOBjQAwgYkCgYEArSQc/U75GB2AtKhbGS5piiLkmJzqEsp64rDxbMJ+xDrye0EN/q1U5Of+
RkDsaN/igkAvV1cuXEgTL6RlafFPcUX7QxDhZBhsYF9pbwtMzi4A4su9hnxIhURebGEmxKW9qJNY
Js0Vo5+IgjxuEWnjnnVgHTs1+mq5QYTA7E6ZyL8CAwEAATANBgkqhkiG9w0BAQQFAAOBgQB3Pw/U
QzPKTPTYi9upbFXlrAKMwtFf2OW4yvGWWvlcwcNSZJmTJ8ARvVYOMEVNbsT4OFcfu2/PeYoAdiDA
cGy/F2Zuj8XJJpuQRSE6PtQqBuDEHjjmOQJ0rV/
r8mO1ZCtHRhpZ5zYRjhRC9eCbjx9VrFax0JDC
/FfwWigmrW0Y0Q==
```

3. Edit the standard metadata file for the Fedlet, such as `$HOME/fedlet/sp.xml`, to include the certificate in KeyDescriptor elements, that are children of the SPSSODescriptor element.

```
<EntityDescriptor
 xmlns="urn:oasis:names:tc:SAML:2.0:metadata"
 entityID="http://www.example.com:8080/fedlet">
 <SPSSODescriptor
  AuthnRequestsSigned="true"
  WantAssertionsSigned="true"
  protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol">
  <KeyDescriptor use="signing">
   <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <ds:X509Data>
     <ds:X509Certificate>
MIICQDCCAakCBEeNB0swDQYJKoZIhvcNAQEEBQAwZzELMAkGA1UEBhMCVVMxEzARBgNVBAgTCkNh
bGlmb3JuaWExFDASBgNVBAcTC1NhbnRhIENsYXJhMQwwCgYDVQQKEwNTdW4xEDAOBgNVBAsTB09w
ZW5TU08xDTALBgNVBAMTBHRlc3QwHhcNMDgwMTE1MTkxOTM5WhcNMTgwMTEyMTkxOTM5WjBnMQsw
CQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcm5pYTEUMBIGA1UEBxMLU2FudGEgQ2xhcmExDDAK
BgNVBAoTA1N1bjEQMA4GA1UECxMHT3BlblNTTzENMAsGA1UEAxMEdGVzdDCBnzANBgkqhkiG9w0B
AQEFAAOBjQAwgYkCgYEArSQc/U75GB2AtKhbGS5piiLkmJzqEsp64rDxbMJ+xDrye0EN/q1U5Of+
RkDsaN/igkAvV1cuXEgTL6RlafFPcUX7QxDhZBhsYF9pbwtMzi4A4su9hnxIhURebGEmxKW9qJNY
Js0Vo5+IgjxuEWnjnnVgHTs1+mq5QYTA7E6ZyL8CAwEAATANBgkqhkiG9w0BAQQFAAOBgQB3Pw/U
QzPKTPTYi9upbFXlrAKMwtFf2OW4yvGWWvlcwcNSZJmTJ8ARvVYOMEVNbsT4OFcfu2/PeYoAdiDA
cGy/F2Zuj8XJJpuQRSE6PtQqBuDEHjjmOQJ0rV/r8mO1ZCtHRhpZ5zYRjhRC9eCbjx9VrFax0JDC
/FfwWigmrW0Y0Q==
     </ds:X509Certificate>
    </ds:X509Data>
   </ds:KeyInfo>
```

```
  </KeyDescriptor>
  <KeyDescriptor use="encryption">
   <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <ds:X509Data>
     <ds:X509Certificate>
MIICQDCCAakCBEeNB0swDQYJKoZIhvcNAQEEBQAwZzELMAkGA1UEBhMCVVMxEzARBgNVBAgTCkNh
bGlmb3JuaWExFDASBgNVBAcTC1NhbnRhIENsYXJhMQwwCgYDVQQKEwNTdW4xEDAOBgNVBAsTB09w
ZW5TU08xDTALBgNVBAMTBHRlc3QwHhcNMDgwMTE1MTkxOTM5WhcNMTgwMTEyMTkxOTM5WjBnMQsw
CQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcm5pYTEUMBIGA1UEBxMLU2FudGEgQ2xhcmExDDAK
BgNVBAoTA1N1bjEQMA4GA1UECxMHT3BlblNTTzENMAsGA1UEAxMEdGVzdDCBnzANBgkqhkiG9w0B
AQEFAAOBjQAwgYkCgYEArSQc/U75GB2AtKhbGS5piiLkmJzqEsp64rDxbMJ+xDrye0EN/q1U5Of+
RkDsaN/igkAvV1cuXEgTL6RlafFPcUX7QxDhZBhsYF9pbwtMzi4A4su9hnxIhURebGEmxKW9qJNY
Js0Vo5+IgjxuEWnjnnVgHTs1+mq5QYTA7E6ZyL8CAwEAATANBgkqhkiG9w0BAQQFAAOBgQB3Pw/U
QzPKTPTYi9upbFXlrAKMwtFf2OW4yvGWWvlcwcNSZJmTJ8ARvVYOMEVNbsT4OFcfu2/PeYoAdiDA
cGy/F2Zuj8XJJpuQRSE6PtQqBuDEHjjmOQJ0rV/r8mO1ZCtHRhpZ5zYRjhRC9eCbjx9VrFax0JDC
/FfwWigmrW0Y0Q==
     </ds:X509Certificate>
    </ds:X509Data>
   </ds:KeyInfo>
   <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc">
    <xenc:KeySize xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
     128
    </xenc:KeySize>
   </EncryptionMethod>
  </KeyDescriptor>
  <SingleLogoutService
   Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect"
   Location="http://www.example.com:8080/fedlet/fedletSloRedirect"
   ResponseLocation="http://www.example.com:8080/fedlet/fedletSloRedirect" />
  <SingleLogoutService
   Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST"
   Location="http://www.example.com:8080/fedlet/fedletSloPOST"
   ResponseLocation="http://www.example.com:8080/fedlet/fedletSloPOST" />
  <SingleLogoutService
   Binding="urn:oasis:names:tc:SAML:2.0:bindings:SOAP"
   Location="http://www.example.com:8080/fedlet/fedletSloSoap" />
  <NameIDFormat>
   urn:oasis:names:tc:SAML:2.0:nameid-format:transient
  </NameIDFormat>
  <AssertionConsumerService
   index="0"
   isDefault="true"
   Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST"
   Location="http://www.example.com:8080/fedlet/fedletapplication" />
  <AssertionConsumerService
   index="1"
   Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Artifact"
   Location="http://www.example.com:8080/fedlet/fedletapplication" />
 </SPSSODescriptor>
 <RoleDescriptor
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:query="urn:oasis:names:tc:SAML:metadata:ext:query"
  xsi:type="query:AttributeQueryDescriptorType"
  protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol">
 </RoleDescriptor>
 <XACMLAuthzDecisionQueryDescriptor
  WantAssertionsSigned="false"
  protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol" />
```

```
</EntityDescriptor>
```

4. Edit the extended metadata file for the Fedlet, such as `$HOME/fedlet/sp-extended.xml`, to set the certificate alias names to the alias for the Fedlet certificate, and the `want*Signed` and `want*Encrypted` values to `true`.

   If you reformat the file, take care not to add white space around string values in elements.

```
<EntityConfig xmlns="urn:sun:fm:SAML:2.0:entityconfig"
 xmlns:fm="urn:sun:fm:SAML:2.0:entityconfig" hosted="1"
 entityID="http://www.example.com:8080/fedlet">
 <SPSSOConfig metaAlias="/sp">
  <Attribute name="description">
   <Value></Value>
  </Attribute>
  <Attribute name="signingCertAlias">
   <Value>test</Value>
  </Attribute>
  <Attribute name="encryptionCertAlias">
   <Value>test</Value>
  </Attribute>
  <Attribute name="basicAuthOn">
   <Value>false</Value>
  </Attribute>
  <Attribute name="basicAuthUser">
   <Value></Value>
  </Attribute>
  <Attribute name="basicAuthPassword">
   <Value></Value>
  </Attribute>
  <Attribute name="autofedEnabled">
   <Value>false</Value>
  </Attribute>
  <Attribute name="autofedAttribute">
   <Value></Value>
  </Attribute>
  <Attribute name="transientUser">
   <Value>anonymous</Value>
  </Attribute>
  <Attribute name="spAdapter">
   <Value></Value>
  </Attribute>
  <Attribute name="spAdapterEnv">
   <Value></Value>
  </Attribute>
  <Attribute name="fedletAdapter">
   <Value>com.sun.identity.saml2.plugins.DefaultFedletAdapter</Value>
  </Attribute>
  <Attribute name="fedletAdapterEnv">
   <Value></Value>
  </Attribute>
  <Attribute name="spAccountMapper">
   <Value>com.sun.identity.saml2.plugins.DefaultLibrarySPAccountMapper</Value>
  </Attribute>
  <Attribute name="useNameIDAsSPUserID">
   <Value>false</Value>
  </Attribute>
```

```
<Attribute name="spAttributeMapper">
 <Value>com.sun.identity.saml2.plugins.DefaultSPAttributeMapper</Value>
</Attribute>
<Attribute name="spAuthncontextMapper">
 <Value>com.sun.identity.saml2.plugins.DefaultSPAuthnContextMapper</Value>
</Attribute>
<Attribute name="spAuthncontextClassrefMapping">
 <Value
 >urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport|0|default</Value>
</Attribute>
<Attribute name="spAuthncontextComparisonType">
 <Value>exact</Value>
</Attribute>
<Attribute name="attributeMap">
 <Value>*=*</Value>
</Attribute>
<Attribute name="saml2AuthModuleName">
 <Value></Value>
</Attribute>
<Attribute name="localAuthURL">
 <Value></Value>
</Attribute>
<Attribute name="intermediateUrl">
 <Value></Value>
</Attribute>
<Attribute name="defaultRelayState">
 <Value></Value>
</Attribute>
<Attribute name="appLogoutUrl">
 <Value>http://www.example.com:8080/fedlet/logout</Value>
</Attribute>
<Attribute name="assertionTimeSkew">
 <Value>300</Value>
</Attribute>
<Attribute name="wantAttributeEncrypted">
 <Value>true</Value>
</Attribute>
<Attribute name="wantAssertionEncrypted">
 <Value>true</Value>
</Attribute>
<Attribute name="wantNameIDEncrypted">
 <Value>true</Value>
</Attribute>
<Attribute name="wantPOSTResponseSigned">
 <Value></Value>
</Attribute>
<Attribute name="wantArtifactResponseSigned">
 <Value></Value>
</Attribute>
<Attribute name="wantLogoutRequestSigned">
 <Value></Value>
</Attribute>
<Attribute name="wantLogoutResponseSigned">
 <Value></Value>
</Attribute>
<Attribute name="wantMNIRequestSigned">
 <Value></Value>
</Attribute>
<Attribute name="wantMNIResponseSigned">
```

```
    <Value></Value>
   </Attribute>
   <Attribute name="responseArtifactMessageEncoding">
    <Value>URI</Value>
   </Attribute>
   <Attribute name="cotlist">
    <Value>fedlet-cot</Value>
   </Attribute>
   <Attribute name="saeAppSecretList">
   </Attribute>
   <Attribute name="saeSPUrl">
    <Value></Value>
   </Attribute>
   <Attribute name="saeSPLogoutUrl">
   </Attribute>
   <Attribute name="ECPRequestIDPListFinderImpl">
    <Value>com.sun.identity.saml2.plugins.ECPIDPFinder</Value>
   </Attribute>
   <Attribute name="ECPRequestIDPList">
    <Value></Value>
   </Attribute>
   <Attribute name="ECPRequestIDPListGetComplete">
    <Value></Value>
   </Attribute>
   <Attribute name="enableIDPProxy">
    <Value>false</Value>
   </Attribute>
   <Attribute name="idpProxyList">
    <Value></Value>
   </Attribute>
   <Attribute name="idpProxyCount">
    <Value>0</Value>
   </Attribute>
   <Attribute name="useIntroductionForIDPProxy">
    <Value>false</Value>
   </Attribute>
   <Attribute name="spSessionSyncEnabled">
    <Value>false</Value>
   </Attribute>
   <Attribute name="relayStateUrlList">
   </Attribute>
  </SPSSOConfig>
  <AttributeQueryConfig metaAlias="/attrQuery">
   <Attribute name="signingCertAlias">
    <Value>test</Value>
   </Attribute>
   <Attribute name="encryptionCertAlias">
    <Value>test</Value>
   </Attribute>
   <Attribute name="wantNameIDEncrypted">
    <Value>true</Value>
   </Attribute>
   <Attribute name="cotlist">
    <Value>fedlet-cot</Value>
   </Attribute>
  </AttributeQueryConfig>
  <XACMLAuthzDecisionQueryConfig metaAlias="/pep">
   <Attribute name="signingCertAlias">
    <Value>test</Value>
```

```
  </Attribute>
  <Attribute name="encryptionCertAlias">
   <Value>test</Value>
  </Attribute>
  <Attribute name="basicAuthOn">
   <Value>false</Value>
  </Attribute>
  <Attribute name="basicAuthUser">
   <Value></Value>
  </Attribute>
  <Attribute name="basicAuthPassword">
   <Value></Value>
  </Attribute>
  <Attribute name="wantXACMLAuthzDecisionResponseSigned">
   <Value>false</Value>
  </Attribute>
  <Attribute name="wantAssertionEncrypted">
   <Value>true</Value>
  </Attribute>
  <Attribute name="cotlist">
   <Value>fedlet-cot</Value>
  </Attribute>
 </XACMLAuthzDecisionQueryConfig>
</EntityConfig>
```

5. Make a copy of `sp-extended.xml` called `sp-extended-copy.xml` and set `hosted="0"` in the root element of the copy.

   Use the copy, `sp-extended-copy.xml`, when importing the Fedlet configuration into OpenAM. OpenAM must register the Fedlet as a *remote* service provider.

6. In the AM console delete the original SP entity configuration for the Fedlet, and then import the updated metadata for the new configuration into OpenAM on the IdP side.

7. Restart the Fedlet or the container in which it runs in order for the Fedlet to pick up the changes to the configuration properties and the metadata.

3.1.3. Customizing a Java Fedlet

You can customize the Java Fedlet to perform many of the SAML v2.0 service provider operations. The Java Fedlet has the SAML v2.0 capabilities identified in Table 1.1, "Fedlet Support for SAML v2.0 Features".

Procedure 3.6. To Add Your Application

The Fedlet includes the following files that you use when building your own service provider application based on the demo web application, including a set of JavaServer Pages (JSP) examples.

`conf/`

Configuration files copied to `$HOME/fedlet` when you first deploy and configure the Fedlet. When deploying your application, you can move these to an alternate location passed to the Java virtual

machine for the web application container at startup. For example, if you store the configuration under `/export/fedlet/`, then you could pass the following property to the JVM.

```
-Dcom.sun.identity.fedlet.home=/export/fedlet/conf
```

You do not need to include these files in your application.

`fedletAttrQuery.jsp`
`fedletAttrResp.jsp`

Sample SAML attribute query and response handlers.

`fedletEncode.jsp`

Utility JSP to encode a password, such as the password used to protect a Java keystore.

`fedletSampleApp.jsp`
`index.jsp`

Demo application. You can remove these before deployment to replace them with your application.

`fedletXACMLQuery.jsp`
`fedletXACMLResp.jsp`

Sample SAML XACML query and response handlers.

`logout.jsp`

Utility page to perform single log out.

`saml2/jsp/`

JSPs to initiate single sign-on and single logout, and to handle errors, and also a JSP for obtaining Fedlet metadata, `saml2/jsp/exportmetadata.jsp`.

`WEB-INF/classes/`

Localized Java properties files for strings used in the Fedlet user interface.

`WEB-INF/lib/`

Fedlet libraries required by your application.

`WEB-INF/web.xml`

Fedlet web application configuration, showing how JSPs map to URLs used in the Fedlet. Add mappings for your application before deployment.

In the `web.xml` mappings, your application must be mapped to `/fedletapplication`, as this is the assertion consumer URL set in the Fedlet metadata.

```
<servlet>
    <servlet-name>yourApp</servlet-name>
    <jsp-file>/fedletSampleApp.jsp</jsp-file>
</servlet>
<servlet-mapping>
    <servlet-name>yourApp</servlet-name>
    <url-pattern>/fedletapplication</url-pattern>
</servlet-mapping>
```

Follow these steps for a very simple demonstration of how to customize the Fedlet:

1. Backup `fedletSampleApp.jsp`.

```
$ cd /path/to/tomcat/webapps/fedlet/
$ cp fedletSampleApp.jsp fedletSampleApp.jsp.orig
```

2. Edit `fedletSampleApp.jsp` to reduce it to a single redirection to `myapp.jsp`. An implementation of the <html> element of the file follows below.

```
<html>
<head>
    <title>Fedlet Sample Application</title>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
</head>

<body>
<%
    // BEGIN : following code is a must for Fedlet (SP) side application
    Map map;
    try {
        // invoke the Fedlet processing logic. this will do all the
        // necessary processing conforming to SAML v2.0 specifications,
        // such as XML signature validation, Audience and Recipient
        // validation etc.
        map = SPACSUtils.processResponseForFedlet(request, response,
            new PrintWriter(out, true));
        response.sendRedirect("myapp.jsp");
    } catch (SAML2Exception sme) {
        SAMLUtils.sendError(request, response,
            response.SC_INTERNAL_SERVER_ERROR, "failedToProcessSSOResponse",
            sme.getMessage());
        return;
    } catch (IOException ioe) {
        SAMLUtils.sendError(request, response,
            response.SC_INTERNAL_SERVER_ERROR, "failedToProcessSSOResponse",
            ioe.getMessage());
        return;
    } catch (SessionException se) {
        SAMLUtils.sendError(request, response,
            response.SC_INTERNAL_SERVER_ERROR, "failedToProcessSSOResponse",
            se.getMessage());
        return;
    } catch (ServletException se) {
        SAMLUtils.sendError(request, response,
            response.SC_BAD_REQUEST, "failedToProcessSSOResponse",
            se.getMessage());
        return;
```

```
    }
    // END : code is a must for Fedlet (SP) side application
%>
</body>
</html>
```

3. Add a `myapp.jsp` page to the Fedlet, such as the following:

```
<html>
<head>
<title>My Application</title>
<meta http-equiv="Content-Type" content="text/html" />
</head>

<body>

<h1>My Application</h1>

<p>After you change the <code>fedletSampleApp.jsp</code>,
   all it does is redirect to this home page after
   successful login.</p>

</body>
</html>
```

4. Browse to the Fedlet URL, such as `http://openam.example.com:8080/fedlet/`, and try one of the login methods.

   After login you are redirected to `myapp.jsp`.

3.1.3.1. Performing Single Sign-On

The Java Fedlet includes a JSP file, `saml2/jsp/fedletSSOInit.jsp`, that you can call to initiate single sign-on from the Fedlet (SP) side. The Fedlet home page, `index.jsp`, calls this page when the user does Fedlet-initiated single sign-on.

When calling this JSP, the parameters to use are those also used by `saml2/jsp/spSSOInit.jsp` in OpenAM. Those parameters are described in spSSOInit.jsp Parameters.

For IdP-initiated single sign-on, call the appropriate page on the identity provider. OpenAM's page is described in idpSSOInit.jsp Parameters.

After single sign-on, the user-agent is directed by default to the assertion consumer URI set in the Fedlet metadata, which by default is `/fedletapplication`. Also by default that URI points to the JSP, `fedletSampleApp.jsp`

3.1.3.2. Performing Single Logout

The Java Fedlet includes a JSP file, `saml2/jsp/spSingleLogoutInit.jsp`, that you can call to initiate single logout from the Fedlet (SP) side. The Fedlet assertion consumer page, `fedletSampleApp.jsp`, calls this when the user does Fedlet-initiated single logout.

When calling this JSP, the parameters to use are those also used by `saml2/jsp/spSingleLogoutInit.jsp` in OpenAM. Those parameters are described in spSingleLogoutInit.jsp Parameters.

For IdP-initiated single logout, call the appropriate page on the identity provider. OpenAM's page is described in idpSingleLogoutInit.jsp Parameters.

Set the `RelayState` parameter when initiating logout to redirect the user-agent appropriately when the process is complete.

3.1.3.3. Performing Attribute Queries

As seen in Procedure 3.3, "To Use the Fedlet to Query an Attribute Authority", an attribute query allows the Fedlet to get profile information about a subject from the attribute authority. The Fedlet must be configured to deal with responses from the attribute authority, including configuration for signing and encryption. Also, an identity provider and attribute authority is likely to share only those attributes that the Fedlet absolutely requires to provide service, such as, for example, a name to customize a page. The attributes must then be mapped in the attribute authority and Fedlet metadata.

The Java Fedlet includes a JSP file, `fedletAttrQuery.jsp`, which is used in the procedure described above to prepare an attribute query using the transient subject identifier obtained during single sign-on. The `fedletAttrQuery.jsp` also supports using the Subject name from an X.509 identity certificate.

Another JSP file, `fedletAttrResp.jsp`, sends the query to the attribute authority using `com.sun.identity.saml2.profile.AttributeQueryUtil.html.getAttributesForFedlet()`, and if successful processes the result, which is a `java.util.Map` of the attribute types and their values.

3.1.3.4. Performing XACML Queries

As seen in Procedure 3.4, "To Use the Fedlet to Query an XACML Policy Decision Point", a XACML query allows the Fedlet to request a policy decision from a XACML PDP. You can configure OpenAM to respond to such queries as described in that procedure.

The Java Fedlet includes a JSP file, `fedletXACMLQuery.jsp`, which is used in the procedure described above to prepare a XACML query, identifying a resource URL and a type of HTTP operation to perform, and specifying the subject identifier obtained during single sign-on.

Another JSP file, `fedletXACMLResp.jsp`, sends the query to the XACML PDP using `com.sun.identity.saml2.profile.XACMLQueryUtil.getPolicyDecisionForFedlet()`, and if successful processes the result, which is a `java.lang.String` representing the decision, such as `Permit` if the decision is to allow access, or `Deny` if the decision is to deny access.

3.2. Configuring Java Fedlets By Hand

An OpenAM Fedlet is a small web application that makes it easy to add SAML v2.0 service provider (SP) capabilities to your Java web application. The AM console offers a wizard for configuring a Java

Fedlet as a SAML v2.0 service provider with OpenAM as the identity provider (IdP). If that fits your purposes, then read the chapter Section 3.1, "Using Fedlets in Java Web Applications" instead.

The full distribution file, `OpenAM-14.0.0.zip`, also includes an Java Fedlet, `Fedlet-14.0.0.zip`, that you can configure by hand. This chapter covers how to configure a Java Fedlet using that distribution, by manually editing the Circle of Trust, Java properties, and IdP and SP XML configuration templates.

Seen from a high level, what you must do is this:

- Determine the roles that the IdP(s) and Fedlet play in SAML v2.0 Circles of Trust.
- Unpack the unconfigured Fedlet from the full OpenAM distribution to access the Fedlet war and template configuration files.
- Begin preparing the Fedlet configuration, including setting up a configuration directory and keystore if needed.
- Obtain SAML v2.0 metadata configuration files from the IdP(s), and add them to the Fedlet configuration.

  The IdP must provide at least the standard SAML v2.0 metadata.
- Finish preparing the Fedlet configuration by editing the remaining Fedlet template configuration files.
- Share the Fedlet SAML v2.0 configuration files at least for the standard SAML v2.0 metadata with the IdP(s).

  An IdP relies on the standard SAML v2.0 metadata to communicate with the Fedlet.
- Deploy and test the Fedlet with each IdP.

3.2.1. Java Fedlet Layout

Unpack the Java Fedlet distribution into a working directory.

```
$ mkdir fedlet && cd fedlet
$ unzip ../Fedlet-14.0.0.zip
```

When you unpack the `Fedlet-14.0.0.zip` file, you find the following files.

`Fedlet-14.0.0.war`

This file contains a Java Fedlet web application that serves as an example, and that you can embed in your applications.

`README`

This file succinctly describes how to configure some Fedlet features.

`conf/`

This folder contains the Fedlet configuration templates that you edit as appropriate for your deployment.

When editing the templates, place copies of the files in the Fedlet home directory on the system where you deploy the Fedlet. By default the Fedlet home directory is *`user.home/uri`*, where *`user.home`* is the value of the Java system property `user.home` for the user running the web container where you deploy the Fedlet, and *`uri`* is the path of the URI where you deploy the Fedlet, such as `/fedlet`.

For example, if *`user.home`* is `/home/user`, that user could have a `/home/user/fedlet` folder for Fedlet configuration files.

```
$ mkdir ~/fedlet
```

To change the location, set the system property `com.sun.identity.fedlet.home` when starting the container where the Fedlet runs.

```
$ java -Dcom.sun.identity.fedlet.home=/path/to/fedlet/conf ...
```

`conf/FederationConfig.properties`

This file defines settings for the Fedlet as a web application. It does not address the SAML v2.0 configuration.

For more about this file, see Section 3.2.2, "Configuring Java Fedlet Properties".

`conf/fedlet.cot-template`

This template defines settings for a SAML v2.0 Circle of Trust to which the Fedlet belongs.

For more about this file, see Section 3.2.3, "Configuring Circles of Trust".

`conf/idp.xml` **(not provided)**

The `idp.xml` file is standard SAML v2.0 metadata that describes the IdP configuration.

Templates for other SAML v2.0 configuration files are provided, but no `idp.xml` template file is provided.

Instead you must obtain the SAML v2.0 metadata from the IdP, and add it as `idp.xml` here, alongside the other SAML v2.0 configuration files. How you obtain this file from the IdP depends on the IdP implementation.

`conf/idp-extended.xml-template`

This template holds extended SAML v2.0 IdP settings that OpenAM uses.

For more about this file, see Section 3.2.4, "Configuring the Identity Providers".

`conf/sp.xml-template`

This template describes standard SAML v2.0 SP settings.

For more about this file, see Section 3.2.5, "Configuring the Service Providers".

`conf/sp-extended.xml-template`

This template describes extended SAML v2.0 SP settings that the Fedlet uses.

For more about this file, see Section 3.2.5, "Configuring the Service Providers".

3.2.2. Configuring Java Fedlet Properties

The Java Fedlet to configure by hand includes a `FederationConfig.properties` file that defines settings for the Fedlet as a web application. The configuration for a single Java Fedlet includes only one `FederationConfig.properties` file, regardless of how many IdP and SP configurations are involved. This file does not address the SAML v2.0 configuration.

When configured this file contains sensitive properties such as the value of `am.encryption.pwd`. Make sure it is readable only by the user running the Fedlet application.

This section categorizes the settings as follows:

- Deployment URL Settings
- Log and Statistics Settings
- Public and Private Key Settings
- Alternative Implementation Settings

Deployment URL Settings

The following settings define the Fedlet deployment URL.

`com.iplanet.am.server.protocol`

Set this to the protocol portion of the URL, such as HTTP or HTTPS.

`com.iplanet.am.server.host`

Set this to the host portion of the URL, such as `sp.example.com`.

`com.iplanet.am.server.port`

Set this to the port portion of the URL, such as 80, 443, 8080, or 8443.

`com.iplanet.am.services.deploymentDescriptor`

Set this to path portion of the URL, starting with a `/`, such as `/fedlet`.

Log and Statistics Settings

The following settings define the Fedlet configuration for logging and monitoring statistics.

`com.iplanet.am.logstatus`

This sets whether the Fedlet actively writes debug log files.

Default: `ACTIVE`

`com.iplanet.services.debug.level`

This sets the debug log level.

The following settings are available, in order of increasing verbosity:

`off`
`error`
`warning`
`message`

Default: `message`

`com.iplanet.services.debug.directory`

This sets the location of the debug log folder.

Trailing spaces in the file names are significant. Even on Windows systems, use slashes to separate directories.

Examples: `/home/user/fedlet/debug`, `C:/fedlet/debug`

`com.iplanet.am.stats.interval`

This sets the interval at which statistics are written, in seconds.

The shortest interval supported is 5 seconds. Settings less than 5 (seconds) are taken as 5 seconds.

Default: 60

`com.iplanet.services.stats.state`

This sets how the Fedlet writes monitoring statistics.

The following settings are available:

`off`
`console` (write to the container logs)
`file` (write to Fedlet stats logs)

Default: `file`

`com.iplanet.services.stats.directory`

This sets the location of the stats file folder.

Trailing spaces in the file names are significant. Even on Windows systems, use slashes to separate directories.

Examples: `/home/user/fedlet/stats`, `C:/fedlet/stats`

Public and Private Key Settings

The following settings define settings for access to certificates and private keys used in signing and encryption.

Other sections in this guide explain how to configure a Fedlet for signing and encryption including how to work with the keystores that these settings reference, and how to specify public key certificates in standard SAML v2.0 metadata. When working with a Java Fedlet, see the section on Section 3.1.2, "Enabling Signing and Encryption in a Fedlet".

`com.sun.identity.saml.xmlsig.keystore`

This sets the path to the keystore file that holds public key certificates of IdPs and key pairs for the Fedlet.

For hints on generating a keystore file with a key pair, see Procedure 5.3, "To Change Default test Signing Key" in the *Setup and Maintenance Guide*.

Example: `@FEDLET_HOME@/keystore.jks`

`com.sun.identity.saml.xmlsig.storepass`

This sets the path to the file that contains the keystore password encoded by using the symmetric key set as the value of `am.encryption.pwd`.

When creating the file, encode the cleartext password by using your own test copy (not a production version) of OpenAM.

- Log in to the AM console as administrator `amadmin`.
- Under Deployment > Servers > *`Server Name`* > Security > Encryption, set the Password Encryption Key to your symmetric key, and save your work.

  Do not do this in a production system where the existing symmetric key is already in use!
- Switch to the `encode.jsp` page, such as `http://openam.example.com:8080/openam/encode.jsp`, enter the cleartext password to encode with your symmetric key, and click Encode.
- Copy the encoded password to your file.

Example: `@FEDLET_HOME@/.storepass`

`com.sun.identity.saml.xmlsig.keypass`

This sets the path to the file that contains the private key password encoded by using the symmetric key set as the value of `am.encryption.pwd`.

To encode the cleartext password, follow the same steps for the password used when setting `com.sun.identity.saml.xmlsig.storepass`.

Example: `@FEDLET_HOME@/.keypass`

`com.sun.identity.saml.xmlsig.certalias`

This sets the alias of the Fedlet's public key certificate.

Example: `fedlet-cert`

`com.sun.identity.saml.xmlsig.storetype`

The sets the type of keystore.

Default: `JKS`

`am.encryption.pwd`

This sets the symmetric key that used to encrypt and decrypt passwords.

Example: `uu4dHvBkJJpIjPQWM74pxH3brZJ5gJje`

Alternative Implementation Settings

The Java Fedlet properties file includes settings that let you plug in alternative implementations of Fedlet capabilities. You can safely use the default settings, as specified in the following list. The list uses the same order for the keys you find in the file.

`com.sun.identity.plugin.configuration.class`

Default: `com.sun.identity.plugin.configuration.impl.FedletConfigurationImpl`

`com.sun.identity.plugin.datastore.class.default`

Default: `com.sun.identity.plugin.datastore.impl.FedletDataStoreProvider`

`com.sun.identity.plugin.log.class`

Default: `com.sun.identity.plugin.log.impl.FedletLogger`

`com.sun.identity.plugin.session.class`

Default: `com.sun.identity.plugin.session.impl.FedletSessionProvider`

`com.sun.identity.plugin.monitoring.agent.class`

Default: `com.sun.identity.plugin.monitoring.impl.FedletAgentProvider`

`com.sun.identity.plugin.monitoring.saml1.class`

Default: `com.sun.identity.plugin.monitoring.impl.FedletMonSAML1SvcProvider`

`com.sun.identity.plugin.monitoring.saml2.class`

Default: `com.sun.identity.plugin.monitoring.impl.FedletMonSAML2SvcProvider`

`com.sun.identity.plugin.monitoring.idff.class`

Default: `com.sun.identity.plugin.monitoring.impl.FedletMonIDFFSvcProvider`

`com.sun.identity.saml.xmlsig.keyprovider.class`

Default: `com.sun.identity.saml.xmlsig.JKSKeyProvider`

`com.sun.identity.saml.xmlsig.signatureprovider.class`

Default: `com.sun.identity.saml.xmlsig.AMSignatureProvider`

`com.sun.identity.common.serverMode`

Default: `false`

`com.sun.identity.webcontainer`

Default: `WEB_CONTAINER`

`com.sun.identity.saml.xmlsig.passwordDecoder`

Default: `com.sun.identity.fedlet.FedletEncodeDecode`

`com.iplanet.services.comm.server.pllrequest.maxContentLength`

Default: 16384

`com.iplanet.security.SecureRandomFactoryImpl`

Default: `com.iplanet.am.util.SecureRandomFactoryImpl`

`com.iplanet.security.SSLSocketFactoryImpl`

Default: `com.sun.identity.shared.ldap.factory.JSSESocketFactory`

`com.iplanet.security.encryptor`

Default: `com.iplanet.services.util.JCEEncryption`

`com.sun.identity.jss.donotInstallAtHighestPriority`

Default: `true`

`com.iplanet.services.configpath`

Default: `@BASE_DIR@`

3.2.3. Configuring Circles of Trust

As described in Section 3.2.1, "Java Fedlet Layout", this template defines settings for a SAML v2.0 Circle of Trust. The Fedlet belongs to at least one Circle of Trust.

This section includes the following procedures:

- Procedure 3.7, "To Configure a Circle of Trust With a Single IdP"
- Procedure 3.8, "To Configure Multiple Circles of Trust"
- Procedure 3.9, "To Configure a Circle of Trust With Multiple IdPs"

Procedure 3.7. To Configure a Circle of Trust With a Single IdP

When the Fedlet is involved in only a single Circle of Trust with one IdP and the Fedlet as an SP, the only settings to change are `cot-name` and `sun-fm-trusted-providers`.

1. Save a copy of the template as `fedlet.cot` in the configuration folder, as in the following example.

   ```
   $ cp ~/Downloads/fedlet/conf/fedlet.cot-template ~/fedlet/fedlet.cot
   ```

2. Set `cot-name` to the name of the Circle of Trust.

3. Set `sun-fm-trusted-providers` to a comma-separated list of the entity names for the IdP and SP.

   For example, if the IdP is OpenAM with entity ID `https://openam.example.com:8443/openam` and the SP is the Fedlet with entity ID `https://sp.example.net:8443/fedlet`, then set the property as follows.

   ```
   sun-fm-trusted-providers=https://openam.example.com:8443/openam,\
    https://sp.example.net:8443/fedlet
   ```

Procedure 3.8. To Configure Multiple Circles of Trust

This procedure concerns deployments where the Fedlet participates as SP in multiple Circles of Trust, each involving their own IdP.

1. For each Circle of Trust, save a copy of the template in the configuration folder.

   The following example involves two Circles of Trust.

   ```
   $ cp ~/Downloads/fedlet/conf/fedlet.cot-template ~/fedlet/fedlet.cot
   $ cp ~/Downloads/fedlet/conf/fedlet.cot-template ~/fedlet/fedlet2.cot
   ```

2. Set up IdP XML files for each IdP as described in Section 3.2.4, "Configuring the Identity Providers".

3. For each Circle of Trust, set up the cot file as described in Procedure 3.7, "To Configure a Circle of Trust With a Single IdP".

4. In the extended SP XML file described in Section 3.2.4, "Configuring the Identity Providers", set the Attribute element with name `cotlist` to include values for all Circles of Trust. The values are taken from the `cot-name` settings in the cot files.

The following example works with two Circles of Trust, `cot` and `cot2`.

```
<Attribute name="cotlist">
    <Value>cot</Value>
    <Value>cot2</Value>
</Attribute>
```

The same Attribute element is also available in extended IdP XML files for cases where an IdP belongs to multiple Circles of Trust.

Procedure 3.9. To Configure a Circle of Trust With Multiple IdPs

When the Circle of Trust involves multiple IdPs, use the Fedlet in combination with the OpenAM IdP Discovery service.

Note

For this to work, the IdPs must be configured to use IdP discovery, and users must have preferred IdPs.

1. Set up the OpenAM IdP Discovery service.

   For details see Section 2.3.6, "Deploying the Identity Provider Discovery Service".

2. Configure the Circle of Trust as described in Procedure 3.7, "To Configure a Circle of Trust With a Single IdP", but specifying multiple IdPs, including the IdP that provides the IdP Discovery service.

3. Set the `sun-fm-saml2-readerservice-url` and the `sun-fm-saml2-writerservice-url` properties as defined for the IdP Discovery service.

3.2.4. Configuring the Identity Providers

As described in Section 3.2.1, "Java Fedlet Layout", the IdP provides its standard SAML v2.0 metadata as XML, which you save in the configuration folder as `idp.xml`. If the IdP uses OpenAM, the IdP can also provide extended SAML v2.0 metadata as XML, which you save in the configuration folder as `idp-extended.xml`, rather than using the template for extended information.

If you have multiple identity providers, then number the configuration files, as in `idp.xml`, `idp2.xml`, `idp3.xml`, and also `idp-extended.xml`, `idp2-extended.xml`, `idp3-extended.xml` and so on.

3.2.4.1. Identity Provider Standard XML

This section covers the configuration in `idp.xml`. The `idp.xml` file contains standard SAML v2.0 metadata for an IdP in a Circle of Trust that includes the Fedlet as SP. The IdP provides you the content of this file.

If the IdP uses OpenAM then the administrator can export the metadata by using either the **ssoadm create-metadata-templ** command or the `/saml2/jsp/exportmetadata.jsp` endpoint under the OpenAM deployment URL.

If the IdP uses an implementation different from OpenAM, see the documentation for details on obtaining the standard metadata. The standard, product-independent metadata are covered in *Metadata for the OASIS Security Assertion Markup Language (SAML) V2.0*. The standard XML namespace describing the XML document has identifier `urn:oasis:names:tc:SAML:2.0:metadata`. An XML schema description for this namespace is found online at http://docs.oasis-open.org/security/saml/v2.0/saml-schema-metadata-2.0.xsd.

3.2.4.2. Identity Provider Extended XML

This section covers the configuration in `idp-extended.xml`. Most extended metadata are specific to the OpenAM implementation of SAML v2.0. If the IdP runs OpenAM, have the IdP provide the extended metadata exported by using the **ssoadm create-metadata-templ** command. This section covers only the basic settings relative to all IdPs.

The extended metadata file describes an `EntityConfig` element, defined by the namespace with the identifier `urn:sun:fm:SAML:2.0:entityconfig`. The XML schema definition is described in `entity-config-schema.xsd`, available online as part of the OpenAM source code, though not included in the OpenAM war file.

The unconfigured Fedlet includes a template file, `conf/idp-extended.xml-template`. This extended metadata template for the IdP requires that you edit at least the `IDP_ENTITY_ID` and `fedletcot` values to reflect the IdP entity ID used in the standard metadata and the Circle of Trust name defined in `fedlet.cot`, respectively. The `hosted` attribute on the `EntityConfig` element must remain set to `hosted="0"`, meaning that the IdP is remote. The IdP is likely to play at least the role of SSO identity provider, though the namespace defines elements for the attribute authority and policy decision point roles shown in the template, as well as the others defined in the standard governing SAML v2.0 metadata.

The extended metadata file is essentially a series of XML maps of key-value pairs specifying IdP configuration for each role. All role-level elements can take a `metaAlias` attribute that the Fedlet uses when communicating with the IdP. Each child element of a role element defines an `Attribute` whose `name` is the key. Each `Attribute` element can contain multiple `Value` elements. The `Value` elements' contents comprise the values for the key. All values are strings, sometimes with a format that is meaningful to OpenAM. The basic example in the IdP template shows the minimal configuration for the SSO IdP role.

In the following example, the `description` is empty and the name of the Circle of Trust is `fedletcot`.

```
<IDPSSOConfig>
<Attribute name="description">
    <Value/>
</Attribute>
<Attribute name="cotlist">
    <Value>fedletcot</Value>
</Attribute>
```

When functioning as IdP, OpenAM can take many other `Attribute` values. These are implementation dependent. You can obtain the extended metadata from OpenAM either as part of the pre-packaged Java Fedlet that you create by using the AM console wizard as described in Procedure 3.1, "To Create Configuration Files for a Fedlet", or by using the **ssoadm create-metadata-templ** subcommand.

Note

Custom authentication contexts can be loaded and saved when they are loaded via ssoadm as part of the hosted IdP/SP extended metadata and the saves are made in the AM console. Any custom contexts loaded via ssoadm are also visible in the AM console.

For example, you can specify custom entries in the `idpAuthncontextClassrefMapping` element of the extended metadata for a hosted IdP as follows:

```
<Attribute name="idpAuthncontextClassrefMapping">
 <Value>urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport
  |1||default</Value>
 <Value>http://idmanagement.gov/ns/assurance/loa/4|4||</Value>
 <Value>http://idmanagement.gov/ns/assurance/loa/3|3||</Value>
 <Value>http://idmanagement.gov/ns/assurance/loa/2|2||</Value>
 <Value>http://idmanagement.gov/ns/assurance/loa/1|1||</Value>
</Attribute>
```

3.2.5. Configuring the Service Providers

As mentioned in Section 3.2.1, "Java Fedlet Layout", the Fedlet SAML v2.0 configuration is defined in two XML files, the standard metadata in `sp.xml` and the extended metadata in `sp-extended.xml`.

If the Fedlet has multiple service provider personalities, then number the configuration files, as in `sp.xml`, `sp2.xml`, `sp3.xml`, and also `sp-extended.xml`, `sp2-extended.xml`, `sp3-extended.xml` and so on.

3.2.5.1. Service Provider Standard XML

This section covers the configuration in `sp.xml`. The `sp.xml` file contains standard SAML v2.0 metadata for the Fedlet as SP. If you edit the standard metadata, make sure that you provide the new version to your IdP, as the IdP software relies on the metadata to get the Fedlet's configuration.

The standard metadata are covered in *Metadata for the OASIS Security Assertion Markup Language (SAML) V2.0*. The standard XML namespace describing the XML document has identifier `urn:oasis:names:tc:SAML:2.0:metadata`. An XML schema description for this namespace is found online at http://docs.oasis-open.org/security/saml/v2.0/saml-schema-metadata-2.0.xsd.

A standard metadata file describes the SAML v2.0 roles that the Fedlet plays. The default base element of the file is an `EntityDescriptor`, which is a container for role descriptor elements. The `EntityDescriptor` element can therefore contain multiple role descriptor elements. The namespace for the standard metadata document is `urn:oasis:names:tc:SAML:2.0:metadata`. You can get the corresponding XML schema description online at http://docs.oasis-open.org/security/saml/v2.0/saml-schema-

metadata-2.0.xsd. In general, you can find standard SAML v2.0-related XML schema definitions at http://docs.oasis-open.org/security/saml/v2.0/.

Fedlets do not support all arbitrary SP configurations. As lightweight service provider components, Fedlets are built to play the SP role in web single sign-on and single logout, to perform attribute queries and XACML policy decision requests, and to work with multiple IdPs including Circles of Trust with an IdP discovery service. For a list of what Fedlets support, see the table Table 1.1, "Fedlet Support for SAML v2.0 Features".

When preparing a standard SP metadata file, follow these suggestions.

- Start either with an existing example or with the template, `conf/sp.xml-template`.
- When using the template, replace the following placeholders.

  `FEDLET_ENTITY_ID`

  The Fedlet entity ID used when communicating with the IdP.

  OpenAM often uses the deployment URL as the entity ID, though that is a convention rather than a requirement.

  `FEDLET_PROTOCOL`

  The Fedlet deployment protocol (`http`, `https`)

  `FEDLET_HOST`

  The Fedlet deployment host name

  `FEDLET_PORT`

  The Fedlet deployment port number

  `FEDLET_DEPLOY_URI`

  The Fedlet application deployment path

- Add and edit role elements as children depending on the roles the Fedlet plays as described in the following sections.

3.2.5.1.1. Single Sign-On and Logout: SPSSODescriptor Element

Add an `SPSSODescriptor` element to play the SP role in web single sign-on and logout. An `SPSSODescriptor` element has attributes specifying whether requests and assertion responses should be digitally signed.

- The `AuthnRequestsSigned` attribute indicates whether the Fedlet signs authentication requests.

  If you set the `AuthnRequestsSigned` attribute to true, then you must also configure the `SPSSODescriptor` element to allow the Fedlet to sign requests. For details see the section on Section 3.1.2, "Enabling Signing and Encryption in a Fedlet".

- The `WantAssertionsSigned` attribute indicates whether the Fedlet requests signed assertion responses from the IdP.

An `SPSSODescriptor` element's children indicate what name ID formats the Fedlet supports, and where the IdP can call the following services on the Fedlet.

- The `AssertionConsumerService` elements specify endpoints that support the SAML Authentication Request protocols.

  You must specify at least one of these. The template specifies two, with the endpoint supporting the HTTP POST binding as the default.

- The optional `SingleLogoutService` elements specify endpoints that support the SAML Single Logout protocols.

3.2.5.1.2. Attribute Queries: RoleDescriptor Element

Add a `RoleDescriptor` element with `type="query:AttributeQueryDescriptorType"` to perform attribute queries.

Attribute queries require the IdP to act as Attribute Authority and call for signing and encryption to be configured for the Fedlet. For details see the example in the procedure Procedure 3.3, "To Use the Fedlet to Query an Attribute Authority". For example, you can set the attribute mapping on the Fedlet by editing the extended metadata attribute `attributeMap` in the `SPSSOConfig` element as described in Section 3.2.5.2.1, "Service Provider Extended XML: SPSSOConfig Settings".

3.2.5.1.3. XACML Requests: XACMLAuthzDecisionQueryDescriptor Element

Add an `XACMLAuthzDecisionQueryDescriptor` element to perform XACML policy decision queries.

Attribute queries require the IdP to act as XACML PDP. For details see the example in the procedure Procedure 3.4, "To Use the Fedlet to Query an XACML Policy Decision Point".

3.2.5.2. Service Provider Extended XML

This section covers the configuration in the `sp-extended.xml` file. The extended metadata are specific to the OpenAM implementation of SAML v2.0.

The extended metadata file describes an `EntityConfig` element, defined by the namespace with the identifier `urn:sun:fm:SAML:2.0:entityconfig`. The XML schema definition is described in `entity-config-schema.xsd`, available online as part of the OpenAM source code, though not included with the unconfigured Fedlet.

The unconfigured Fedlet does include a template file, `conf/sp-extended.xml-template`. This extended metadata template for the IdP requires that you edit at least the `FEDLET_ENTITY_ID` placeholder value, the `appLogoutUrl` attribute value in the `SPSSOConfig` element, and the `fedletcot` values. The `FEDLET_ENTITY_ID` value must reflect the SP entity ID used in the standard metadata. For the single

logout profile, the `appLogoutUrl` attribute value must match the Fedlet URL based on the values used in the `FederationConfig.properties` file. The `fedletcot` values must correspond to the Circle of Trust name defined in `fedlet.cot`.

The `hosted` attribute on the `EntityConfig` element must remain set to `hosted="1"`, meaning that the SP is hosted (local to the Fedlet). If you provide a copy of the file to your IdP running OpenAM, however, then set `hosted="0"` for the IdP, as the Fedlet is remote to the IdP.

The extended metadata file is essentially a series of XML maps of key-value pairs specifying IdP configuration for each role. All role-level elements can take a `metaAlias` attribute that the Fedlet uses when communicating with the IdP. Each child element of a role element defines an `Attribute` whose `name` is the key. Each `Attribute` element can contain multiple `Value` elements. The `Value` elements' contents comprise the values for the key. All values are strings, sometimes with a format that is meaningful to the Fedlet. The basic example in the SP template shows the configuration options, documented in the following lists.

3.2.5.2.1. Service Provider Extended XML: SPSSOConfig Settings

This section covers elements for the SP SSO role, arranged in the order they appear in the template.

`description`

Human-readable description of the Fedlet in the SP SSO role

`signingCertAlias`

Alias of the public key certificate for the key pair used when signing messages to the IdP

The key pair is found in the Fedlet's keystore, and the certificate is included in the standard metadata. See Public and Private Key Settings for details on how to specify access to the keystore, and Section 3.2.5.1, "Service Provider Standard XML" for details on how to set up standard metadata.

`encryptionCertAlias`

Alias of the public key certificate for the key pair used when encrypting messages to the IdP

The key pair is found in the Fedlet's keystore, and the certificate is included in the standard metadata. See Public and Private Key Settings for details on how to specify access to the keystore, and Section 3.2.5.1, "Service Provider Standard XML" for details on how to set up standard metadata.

`basicAuthOn`

Set this to true to use HTTP Basic authorization with the IdP.

Default: false

`basicAuthUser`

When using HTTP Basic authorization with the IdP, this value is the user name.

`basicAuthPassword`

When using HTTP Basic authorization with the IdP, this value is the password.

Encrypt the password using the `encode.jsp` page of your test copy of OpenAM that you might also have used to encode keystore passwords as described in Public and Private Key Settings.

`autofedEnabled`

Set this to true to enable automatic federation with OpenAM based on the value of a profile attribute that is common to user profiles both in OpenAM and in the Fedlet's context.

Default: false

`autofedAttribute`

When automatic federation is enabled, set this to the name of the user profile attribute used for automatic federation.

`transientUser`

Use this effective identity for users with transient identifiers.

Default: anonymous

spAdapter

Class name for a plugin service provider adapter

This class must extend `com.sun.identity.saml2.plugins.SAML2ServiceProviderAdapter`.

`spAdapterEnv`

When using a plugin service provider adapter, this attribute's values optionally take a map of settings *`key=value`* used to initialize the plugin.

`fedletAdapter`

Class name for an alternate fedlet adapter

Default: `com.sun.identity.saml2.plugins.DefaultFedletAdapter`

`fedletAdapterEnv`

When using an alternate fedlet adapter, this attribute's values optionally take a map of settings *`key=value`* used to initialize the plugin.

`spAccountMapper`

Class name for an implementation mapping SAML protocol objects to local user profiles

Default: `com.sun.identity.saml2.plugins.DefaultLibrarySPAccountMapper`

`spAttributeMapper`

Class name for an implementation mapping SAML assertion attributes to local user profile attributes

Default: `com.sun.identity.saml2.plugins.DefaultSPAttributeMapper`

`spAuthncontextMapper`

Class name for an implementation determining the authentication context to set in an authentication request, and mapping the authentication context to an authentication level

Default: `com.sun.identity.saml2.plugins.DefaultSPAuthnContextMapper`

`spAuthncontextClassrefMapping`

String defining how the SAML authentication context classes map to authentication levels and indicate the default context class

Format: *`authnContextClass`*`|`*`authLevel`*`[|default]`

Default: `urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport|0|default`

`spAuthncontextComparisonType`

How to evaluate authentication context class identifiers.

`exact`

Assertion context must exactly match a context in the list

`minimum`

Assertion context must be at least as strong as a context in the list

`maximum`

Assertion context must be no stronger than a context in the list

`better`

Assertion context must be stronger than all contexts in the list

Default: `exact`

`attributeMap`

Map of SAML assertion attributes to local user profile attributes

Default: `*=*`

`saml2AuthModuleName`

Name of an alternative SAML v2.0 authentication module

`localAuthURL`

URL to a login page on the Fedlet side

Use this to override the Assertion Consumer Service URL from the standard metadata when consuming assertions.

`intermediateUrl`

URL to an intermediate page returned before the user accesses the final protected resource

`defaultRelayState`

If no RelayState is specified in a SAML request, redirect to this URL after successful single sign-on.

URL-encode the `defaultRelayState` value.

`appLogoutUrl`

One or more Fedlet URLs that initiate single logout

Replace the placeholders in the default with the values for your Fedlet.

Default: `FEDLET_PROTOCOL://FEDLET_HOST:FEDLET_PORT/FEDLET_DEPLOY_URI/logout`

`assertionTimeSkew`

Tolerate clock skew between the Fedlet and the IdP of at most this number of seconds

Default: 300

`wantAttributeEncrypted`

Set to true to request that the IdP encrypt attributes in the response

`wantAssertionEncrypted`

Set to true to request that the IdP encrypt the SAML assertion in the response

`wantNameIDEncrypted`

Set to true to request that the IdP encrypt the name ID in the response

`wantPOSTResponseSigned`

Set to true to request that the IdP sign the response when using HTTP POST

`wantArtifactResponseSigned`

Set to true to request that the IdP sign the response when using HTTP Artifact

`wantLogoutRequestSigned`

Set to true to request that the IdP sign single logout requests

`wantLogoutResponseSigned`

Set to true to request that the IdP sign single logout responses

`wantMNIRequestSigned`

Set to true to request that the IdP manage name ID requests

`wantMNIResponseSigned`

Set to true to request that the IdP manage name ID responses

`cotlist`

Set this to the Circle of Trust name used in Section 3.2.3, "Configuring Circles of Trust".

Default: `fedletcot`

`saeAppSecretList`

When using Secure Attribute Exchange with OpenAM this represents the Application Security Configuration settings.

Values take the format `url=`*`FedletURL`*`|type=`*`symmetric`*`|secret=`*`EncodedSharedSecret`*`[|encryptionalgorithm=`*`EncAlg`*`|encryptionkeystrength=`*`EncStrength`*`]` or `url=`*`FedletURL`*`|type=`*`asymmetric`*`|privatekeyalias=`*`FedletSigningCertAlias`*`[|encryptionalgorithm=`*`EncAlg`*`|encryptionkeystrength=`*`EncStrength`*`|pubkeyalias=`*`FedletPublicKeyAlias`*`]`

You can omit the `privatekeyalias` setting if the signing certifcate is specified in the standard metadata.

`saeSPUrl`

When using Secure Attribute Exchange (SAE) with OpenAM this is the Fedlet URL that handles SAE requests. If this is omitted, then SAE is not enabled.

`saeSPLogoutUrl`

When using Secure Attribute Exchange with OpenAM this is the Fedlet URL that handles SAE global logout requests.

`ECPRequestIDPListFinderImpl`

When using the Enhanced Client and Proxy profile this is the class name for the implementation that returns a list of preferred IdPs trusted by the ECP.

Default: `com.sun.identity.saml2.plugins.ECPIDPFinder`

`ECPRequestIDPList`

When using the Enhanced Client and Proxy profile this is the list of IdPs for the ECP to contact.

When not specified the list finder implementation is used.

`enableIDPProxy`

Set this to true to enable IdP proxy functionality.

Default: false

`idpProxyList`

A list of preferred IdPs that the Fedlet can proxy to

idpProxyCount

Number of IdP proxies that the Fedlet can have

Default: 0

useIntroductionForIDPProxy

Set this to true to pick a preferred IdP based on a SAML v2.0 introduction cookie.

Default: false

3.2.5.2.2. Service Provider Extended XML: AttributeQueryConfig Settings

This section covers elements for the Attribute Requester role, arranged in the order they appear in the template.

signingCertAlias

Alias of the public key certificate for the key pair used when signing messages to the IdP

The key pair is found in the Fedlet's keystore, and the certificate is included in the standard metadata. See Public and Private Key Settings for details on how to specify access to the keystore, and Section 3.2.5.1, "Service Provider Standard XML" for details on how to set up standard metadata.

encryptionCertAlias

Alias of the public key certificate for the key pair used when encrypting messages to the IdP

The key pair is found in the Fedlet's keystore, and the certificate is included in the standard metadata. See Public and Private Key Settings for details on how to specify access to the keystore, and Section 3.2.5.1, "Service Provider Standard XML" for details on how to set up standard metadata.

wantNameIDEncrypted

Set to true to request that the IdP encrypt the name ID

cotlist

Set this to the Circle of Trust name used in Section 3.2.3, "Configuring Circles of Trust".

Default: `fedletcot`

3.2.5.2.3. Service Provider Extended XML: XACMLAuthzDecisionQueryConfig Settings

This section covers elements for the XACML decision requester role, enabling the Fedlet to act as a Policy Enforcement Point, arranged in the order they appear in the template.

signingCertAlias

Alias of the public key certificate for the key pair used when signing messages to the IdP

The key pair is found in the Fedlet's keystore, and the certificate is included in the standard metadata. See Public and Private Key Settings for details on how to specify access to the keystore, and Section 3.2.5.1, "Service Provider Standard XML" for details on how to set up standard metadata.

`encryptionCertAlias`

Alias of the public key certificate for the key pair used when encrypting messages to the IdP

The key pair is found in the Fedlet's keystore, and the certificate is included in the standard metadata. See Public and Private Key Settings for details on how to specify access to the keystore, and Section 3.2.5.1, "Service Provider Standard XML" for details on how to set up standard metadata.

`basicAuthOn`

Set to true to use HTTP Basic authorization when contacting the Policy Decision Provider

Default: false

`basicAuthUser`

When using Basic authorization to contact the Policy Decision Provider, use this value as the user name

`basicAuthPassword`

When using Basic authorization to contact the Policy Decision Provider, use this value as the password

Encrypt the password using the `encode.jsp` page of your test copy of OpenAM that you might also have used to encode keystore passwords as described in Public and Private Key Settings.

`wantXACMLAuthzDecisionResponseSigned`

Set this to true to request that the Policy Decision Provider sign the XACML response

`wantAssertionEncrypted`

Set this to true to request that the Policy Decision Provider encrypt the SAML assertion response

`cotlist`

Set this to the Circle of Trust name used in Section 3.2.3, "Configuring Circles of Trust".

Default: `fedletcot`

3.2.6. Embedding the Java Fedlet in a Web Application

The Fedlet war file, `Fedlet-14.0.0.war`, serves both as an example and also to provide the code needed to embed the Fedlet in your web application.

The basic steps for using the Fedlet in your application are as follows.

- Unpack the Fedlet war file to a working directory, remove any files you do not want to keep such as `index.jsp` or `fedletEncode.jsp`, and overlay the Fedlet files with those of your web application.
- To integrate single sign-on into your application, modify the functionality in `fedletSampleApp.jsp` or add it to your application's logic.

  If you add it to your application's logic, then you must also edit your application's deployment descriptor file, `web.xml`, to set the assertion consumer URI, which by default is `/fedletapplication` in the basic SP XML for the Fedlet. Add `servlet` and `servlet-mapping` elements as shown in the following example.

  ```
  <servlet>
      <servlet-name>yourapplication</servlet-name>
      <jsp-file>/your-application.jsp</jsp-file>
  </servlet>
  <servlet-mapping>
      <servlet-name>yourapplication</servlet-name>
      <url-pattern>/fedletapplication</url-pattern>
  </servlet-mapping>
  ```

- Build a war file from your web application with embedded Fedlet files.

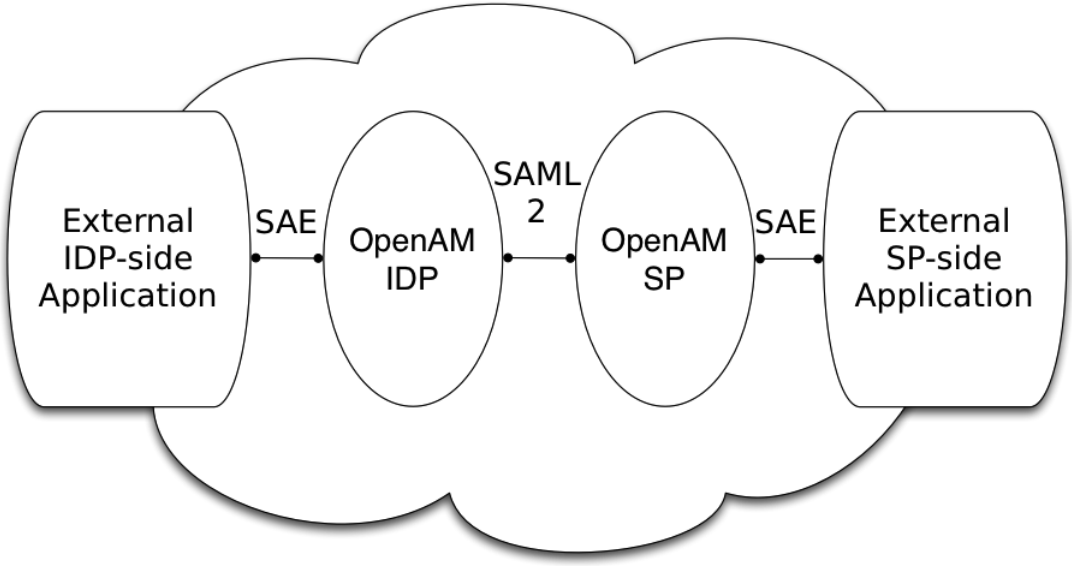
  This is the version of the application to deploy.
- When you deploy your war file, also provide the Fedlet configuration as described in this section.

Chapter 4

Customizing SAML v2.0 Support

Most deployments can rely on OpenAM to handle authentication and provide identity assertions. OpenAM supports a wide variety of authentication scenarios out of the box, but OpenAM also makes it possible to add custom authentication modules. Furthermore, OpenIG lets you integrate legacy systems into your access management deployment.

In a deployment where you need OpenAM to act as a SAML v2.0 gateway to a legacy application that serves as an identity provider, you can use OpenAM Secure Attribute Exchange (SAE). On the identity provider side, SAE lets OpenAM retrieve the information needed to create assertions from an external authentication service, bypassing OpenAM authentication and trusting the external service as the authoritative source of authentication. On the service provider side, SAE lets OpenAM securely provide attributes to an application that makes its own policy decision based on the attributes rather than rely on OpenAM for the policy decision.



When you use SAE on the identity provider side, an external application acts as the authoritative source of authentication. After a user authenticates successfully, the application tells OpenAM to create a session by sending a secure HTTP GET or POST to OpenAM that asserts the identity of the user. OpenAM processes the assertion to create a session for the user. If the user is already

authenticated and comes back to access the application, the application sends a secure HTTP POST to OpenAM to assert both the user's identity and also any necessary attributes related to the user. OpenAM processes the assertion to create the session for the user and populate the attributes in the user's session. When the user logs out, the external authentication application can initiate single logout from the identity provider OpenAM server by sending the `sun.cmd=logout` attribute to OpenAM using SAE.

On the service provider side, OpenAM communicates using SAML v2.0 with OpenAM on the identity provider side. OpenAM can use SAE to transmit attributes to an application through a secure HTTP POST.

SAE relies either on shared keys and symmetric encryption, or on public and private keys and asymmetric encryption to protect attributes communicated between OpenAM and external applications.

OpenAM ships with sample JSPs that demonstrate secure attribute exchange. To try the sample, you must set up an OpenAM Circle of Trust to include an identity provider and a service provider, install the SDK sample web application on each provider, and then configure the providers appropriately as described in this chapter to secure communications with the sample SAE applications on both the identity provider and service provider sides.

4.1. Installing the SAE Samples

Set up an OpenAM server as an identity provider, and another as a service provider, connecting the two in a circle of trust called `samplesaml2cot`. Configure both the hosted providers and also the remote providers as described in Section 2.3, "Configuring Identity Providers, Service Providers, and Circles of Trust". This chapter assumes you set up the hosted identity provider at `http://idp.example.com:8080/openam` and the hosted service provider at `http://sp.example.com:8080/openam`. Use Realms > *`Realm Name`* > Test Federation Connectivity in the AM console to make sure Federation is working before you add secure attribute exchange applications that rely on functioning SAML v2.0 communications between the providers.

Set up the sample web application as described in Section 3.1, "Installing Client SDK Samples" in the *Development Guide*, both on the identity provider side and also on the service provider side. The SAE samples are found under `/saml2/sae` where you installed the samples. `saeIDPApp.jsp` is the identity provider side external application. `saeSPApp.jsp` is the service provider side external application.

4.2. Preparing to Secure SAE Communications

In order for SAE to be secure, you must both set up a trust relationship between the application on the identity provider side and the OpenAM server acting as identity provider, and sets up a trust relationship between the application on the service provider side and the OpenAM server acting as the service provider. These trust relationships can be based on a shared secret and symmetric encryption, or on public and private key pairs and asymmetric encryption. The trust relationships on

either side are independent. For example, you can use a shared secret on the identity provider side and certificates on the service provider side if you chose.

When using symmetric encryption, you must define a shared secret string used both for the application and the provider. The sample uses `secret12` as the shared secret. To simplify configuration, the sample uses the same shared secret, and thus symmetric encryption, for both trust relationships.

When using symmetric encryption, you must also use the encoded version of your shared secret. To get the encoded version of a shared secret string, use the `encode.jsp` page on the provider, as in `http://idp.example.com:8080/openam/encode.jsp` and `http://sp.example.com:8080/openam/encode.jsp`. An encoded version of `secret12` looks something like `AQICEcFhDWmb6sVmMuCJuVh43306HVacDte9`.

When using asymmetric encryption, you must obtain a public-private key pair for the application, and store the keys in a keystore on the application side. Also store the public key from OpenAM which is acting as the provider in the application's keystore. Make note of the certificate aliases for your application's private key, and for OpenAM's public key. Also note the path to the keystore for your application, the keystore password, and the private key password.

4.3. Securing the Identity Provider Side

This configuration uses the default sample settings with a shared secret of `secret12`, without encryption of the attributes:

1. Log in as `amadmin` to the OpenAM server console where you set up the hosted identity provider (IdP).

2. The sample includes a `branch` attribute not found in user profiles by default. Therefore, under Realms > *`Realm Name`* > Authentication > Settings > User Profile, set User Profile to Ignored, and then save your work.

3. Under Realms > *`Realm Name`* > Applications > SAML > Entity Providers, click the name of the hosted IdP to access the IdP configuration:

   - Under Assertion Processing > Attribute Mapper, add both `mail=mail` and `branch=branch` to the attribute map, and then save your work.

   - Under Advanced > SAE Configuration, make sure the IdP URL reflects an endpoint on the IdP such as `http://idp.example.com:8080/openam/idpsaehandler/metaAlias/idp`, and then save your work.

   - Also under Advanced > SAE Configuration > Application Security Configuration, add the URL value for the kind of encryption you are using, and then save your work.

     When using the defaults, the value is something like `url=http://idp.example.com:8080/samples/saml2/sae/saeIDPApp.jsp|type=symmetric|secret=`*`encoded-secret`*, where the OpenAM SDK sample web application is deployed on the IdP side with context root `/samples` and the *`encoded-secret`* is something like `AQICEcFhDWmb6sVmMuCJuVh43306HVacDte9`.

If you use a different mechanism to secure the communications between the SAE application and the provider, read the online help in the console to see how to construct your URL value.

4. Under Realms > *Realm Name* > Applications > SAML > Entity Providers, click the name of the remote SP to access the SP configuration on the IdP side:

   - Under Assertion Processing > Attribute Mapper, add both `mail=mail` and `branch=branch` to the attribute map, and then save your work.
   - Under Advanced > SAE Configuration, make sure the SP URL reflects an endpoint on the SP, such as `http://sp.example.com:8080/openam/spsaehandler/metaAlias/sp`, and then save your work.
   - Also under Advanced > SAE Configuration, add the URL to the sample SAE application as the SP Logout URL, such as `http://sp.example.com:8080/samples/saml2/sae/saeSPApp.jsp`, and then save your work.

4.4. Securing the Service Provider Side

This configuration uses the default sample setting of symmetric encryption, with a shared secret of `secret12`.

Login as `amadmin` to the OpenAM server console where you set up the hosted service provider (SP):

1. The sample includes a `branch` attribute not found in user profiles by default. Therefore, under Realms > *Realm Name* > Authentication > Settings > User Profile, set User Profile to Ignored, and then save your work.

2. Under Realms > *Realm Name* > Applications > SAML > Entity Providers, click the name of the hosted SP to access the SP configuration:

   - Under Assertion Processing > Attribute Mapper, add both `mail=mail` and `branch=branch` to the attribute map, and then save your work.
   - Also under Assertion Processing > Attribute Mapper > Auto Federation, select Enabled, set the Attribute to `mail`, and then save your work.
   - Under Advanced > SAE Configuration, make sure the SP URL reflects an endpoint on the SP such as `http://sp.example.com:8080/openam/spsaehandler/metaAlias/sp`, and then save your work.
   - Furthermore, under Advanced > SAE Configuration, add the URL to the sample SAE application as the SP Logout URL such as `http://sp.example.com:8080/samples/saml2/sae/saeSPApp.jsp`, and then save your work.
   - Also under Advanced > SAE Configuration > Application Security Configuration, add the URL value for the kind of encryption you are using, and then save your work.

When using the defaults, the value is something like `url=http://sp.example.com:8080/samples/saml2/sae/saeSPApp.jsp|type=symmetric|secret=encoded-secret`, where the OpenAM SDK sample web application is deployed on the IdP side with context root `/samples` and the *encoded-secret* is something like `AQICkX24RbZboAVgr2FG1kWoqRv1zM2a6KEH`.

If you use a different mechanism to secure the communications between the SAE application and the provider, read the online help in the console to see how to construct your URL value.

4.5. Trying It Out

After completing the setup described above, navigate to the IdP side SAE application, for example at `http://idp.example.com:8080/samples/saml2/sae/saeIDPApp.jsp`.

Make sure you set at least the "SP App URL" and "SAE URL on IDP end" to fit your configuration. For example if you used the settings above then use the following values:

SP App URL

`http://sp.example.com:8080/samples/saml2/sae/saeSPApp.jsp`

SAE URL on IDP end

`http://idp.example.com:8080/openam/idpsaehandler/metaAlias/idp`

Check the settings, and then click Generate URL to open the Secure Attributes Exchange IDP APP SAMPLE page.

Click the `ssourl` link in the page to start the exchange.

The resulting web page shows the attributes exchanged, including the mail and branch values used. The text of that page is something like the following:

```
SAE SP APP SAMPLE

Secure Attrs :
mail            testuser@foo.com
sun.idpentityid http://idp.example.com:8080/openam
sun.spentityid  http://sp.example.com:8080/openam
branch          mainbranch
sun.authlevel   0
```

Chapter 5
Reference

This reference section covers configuration settings for OpenAM's SAML v2.0 support.

5.1. SAML v2.0 Standards

OpenAM implements the following RFCs, Internet-Drafts, and standards relating to SAML v2.0:

Security Assertion Markup Language (SAML)

Standard, XML-based framework for creating and exchanging security information between online partners. OpenAM supports multiple versions of SAML including 2.0, 1.1, and 1.0.

Specifications are available from the OASIS standards page.

5.2. SAML v2.0 Configuration Properties

This section covers service provider, identity provider, and circle of trust configuration properties.

5.2.1. Hosted Identity Provider Configuration Properties

Once you have set up an identity provider, you can configure it through the AM console under Realms > *Realm Name* > Applications > SAML > Entity Providers > *Provider Name*.

5.2.1.1. Assertion Content

The following properties appear under the Assertion Content tab:

Signing and Encryption

Request/Response Signing

Specifies what parts of messages the identity provider requires the service provider to sign digitally.

Encryption

When selected, the service provider must encrypt NameID elements.

Certificate Aliases

Specifies aliases for certificates in the OpenAM keystore that are used to handle digital signatures, and to handle encrypted messages.

Specify a Key Pass if the private key password is different from the keystore password, which is stored encrypted in the `.keypass` file for the server. For instructions on working with key pairs, also see Chapter 5, "*Setting Up Keys and Keystores*" in the *Setup and Maintenance Guide*.

You can specify lists of aliases for signing and encryption:

- If you specify multiple aliases in the Signing property, OpenAM uses the first key alias from the list to sign SAML assertions.
- If you specify multiple aliases in the Encryption property, OpenAM will attempt to decrypt incoming protocol messages with all matching certificates in the list until decryption is successful.

When a certificate is about to expire, add a new alias to either field to enable OpenAM to maintain the trust relationship between entities for a longer period of time. Make sure that the remote providers also update their copy of the OpenAM provider's metadata to ensure the key rollover process is seamless.

NameID Format

NameID Format List

Specifies the supported name identifiers for users that are shared between providers for single sign-on. If no name identifier is specified when initiating single sign-on, then the identity provider uses the first one that is supported by both providers.

NameID Value Map

Maps name identifier formats to user profile attributes. The `persistent` and `transient` name identifiers need not be mapped.

NameID mapping supports Base64-encoded binary values by adding a `;binary` flag to the mapping. With this flag set, OpenAM Base64-encodes the profile attribute when adding it to the assertion. The mapping may resemble the following:

```
urn:oasis:names:tc:SAML:2.0:nameid-format:persistent=objectGUID;binary
```

Authentication Context

Mapper

Specifies a class that implements the `IDPAuthnContextMapper` interface and sets up the authentication context.

Default Authentication Context

Specifies the authentication context used if no authentication context specified in the request.

Authentication Context Items

Specifies the supported authentication contexts, where the Key and Value can specify a corresponding OpenAM authentication method, and the Level corresponds to an authentication module authentication level.

Assertion Time

Not-Before Time Skew

Grace period in seconds for the `NotBefore` time in assertions.

Effective Time

Validity in seconds of an assertion.

Basic Authentication

Enabled, User Name, Password

When enabled, authenticate with the specified user name and password at SOAP end points.

Assertion Cache

Enabled

When enabled, cache assertions.

5.2.1.2. Assertion Processing

The following properties appear under the Assertion Processing tab:

Attribute Mapper

Attribute Mapper

Specifies a class that implements the attribute mapping.

The default implementation attempts to retrieve the mapped attribute values from the user profile first. If the attribute values are not present in the user's profile, then it attempts to retrieve them from the user's session.

Default: `com.sun.identity.saml2.plugins.DefaultIDPAttributeMapper`

Attribute Map

Maps SAML attributes to user profile attributes.

The user profile attributes used here must both be allowed in user profiles, and also be specified for the identity repository. See Section 3.3.1, "Customizing Profile Attributes" in the *Setup and Maintenance Guide*, for instructions on allowing additional attributes in user profiles.

To specify the list of profile attributes for an LDAP identity repository, login to the AM console as administrator and browse to Realms > *Realm Name* > Data Stores, and click the data store name to open the configuration page. Scroll down to User Configuration, and edit the LDAP User Attributes list, and then click Save to keep your work.

The default IdP mapping implementation allows you to add static values in addition to values taken from the user profile. You add a static value by enclosing the profile attribute name in double quotes (`"`), as in the following examples.

To add a static SAML attribute called `nameID` with a value of `staticNameIDValue` with a name format of `urn:oasis:names:tc:SAML:2.0:attrname-format:uri`, add the following mapping.

```
urn:oasis:names:tc:SAML:2.0:attrname-format:uri|nameID="staticNameIDValue"
```

Account Mapper

Account Mapper

Specifies a class that implements `AccountMapper` to map remote users to local user profiles.

Local Configuration

Auth URL

URL where users are redirected to authenticate.

Reverse Proxy URL

When a reverse proxy is used for SAML endpoints, it is specified here.

External Application Logout URL

URL to which to send an HTTP POST including all cookies when receiving a logout request. To add a user session property as a POST parameter, include it in the URL query string as a `appsessionproperty` parameter.

5.2.1.3. Services

The following properties appear under the Services tab:

MetaAlias

MetaAlias

Used to locate the provider's entity identifier, specified as `[/realm-name]*/provider-name`, where *`provider-name`* cannot contain slash characters (/). For example: `/myRealm/mySubrealm/idp`.

IdP Service Attributes

Artifact Resolution Service

Specifies the end point to handle artifact resolution. The Index is a unique number identifier for the end point.

Single Logout Service

Specifies the end points to handle single logout, depending on the SAML binding selected.

Manage NameID Service

Specifies the end points to handle name identifiers, depending on the SAML binding selected.

Single SignOn Service

Specifies the end points to handle single sign-on.

NameID Mapping

URL

Specifies the end point to handle name identifier mapping.

5.2.1.4. Advanced Settings

The following properties appear under the Advanced tab:

SAE Configuration

IDP URL

Specifies the end point to handle Secure Attribute Exchange requests.

Application Security Configuration

Specifies how to handle encryption for Secure Attribute Exchange operations.

ECP Configuration

IDP Session Mapper

Specifies the class that finds a valid session from an HTTP servlet request to an identity provider with a SAML Enhanced Client or Proxy profile.

Session Synchronization

Enabled

When enabled, the identity provider sends a SOAP logout request over the back channel to all service providers when a session times out. A session may time out when the maximum idle time or maximum session time is reached, for example.

IDP Finder Implementation

IDP Finder Implementation Class

Specifies a class that finds the preferred identity provider to handle a proxied authentication request.

IDP Finder JSP

Specifies a JSP that presents the list of identity providers to the user.

Enable Proxy IDP Finder For All SPs

When enabled, apply the finder for all remote service providers.

Relay State URL List

Relay State URL List

List of URLs permitted for the `RelayState` parameter. OpenAM validates the redirection URL in the `RelayState` parameter against this list. If the `RelayState` parameter's value is in the list, OpenAM allows redirection to the `RelayState` URL. If it is not in the list, a browser error occurs.

Use the pattern matching rules described in Section 8.1.3, "Constraining Post-Login Redirects" in the *Authentication and Single Sign-On Guide* to specify URLs in the list.

If you do not specify any URLs in this property, OpenAM does not validate the `RelayState` parameter.

IDP Adapter

IDP Adapter Class

Specifies a class to invoke immediately before sending a SAML v2.0 response.

5.2.2. Hosted Service Provider Configuration Properties

Once you have set up a service provider, you can configure it through the AM console by navigating to > *`Realm Name`* > Applications > SAML > Entity Providers > *`Provider Name`*.

5.2.2.1. Assertion Content

The following properties appear under the Assertion Content tab:

Signing and Encryption

Request/Response Signing

Specifies what parts of messages the service provider requires the identity provider to sign digitally.

Encryption

The identity provider must encrypt selected elements.

Certificate Aliases

Specifies aliases for certificates in the OpenAM keystore that are used to handle digital signatures, and to handle encrypted messages.

You can specify lists of aliases for signing and encryption:

- If you specify multiple aliases in the Signing property, OpenAM uses the first key alias from the list to sign SAML assertions.
- If you configure multiple aliases in the Encryption property, OpenAM will use all private keys associated with the aliases until decryption is successful.

When a certificate is about to expire, add a new alias to either field to enable OpenAM to maintain the trust relationship between entities for a longer period of time. Make sure that the remote providers also update their copy of the OpenAM provider's metadata to ensure the key rollover process is seamless.

NameID Format

NameID Format List

Specifies the supported name identifiers for users that are shared between providers for single sign-on. If no name identifier is specified when initiating single sign-on, then the service provider uses the first one in the list supported by the identity provider.

Disable Federation persistence if NameID Format is unspecified

When enabled, the NameID Format in the authentication response is `urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified`, and the Account Mapper has identified the local user, the service provider does not persist federation information in the user profile.

Authentication Context

Mapper

Specifies a class that implements the `SPAuthnContextMapper` interface and sets up the authentication context.

Default Authentication Context

Specifies the authentication context used if no authentication context specified in the request.

Authentication Context Items

Specifies the supported authentication contexts. The Level corresponds to an authentication module authentication level.

Comparison Type

How the authentication context in the assertion response must compare to the supported contexts.

Assertion Time

Assertion Time Skew

Grace period in seconds for the `NotBefore` time in assertions.

Basic Authentication

Enabled, User Name, Password

When enabled, authenticate with the specified user name and password at SOAP end points.

5.2.2.2. Assertion Processing

The following properties appear under the Assertion Processing tab:

Attribute Mapper

Attribute Mapper

Specifies a class that implements the attribute mapping.

Attribute Map

Maps SAML attributes to user profile attributes.

Auto Federation

Enabled

When enabled, automatically federate user's accounts at different providers based on the specified SAML attribute.

Attribute

Specifies the SAML attribute to match accounts at different providers.

Account Mapper

Account Mapper

Specifies a class that implements `AccountMapper` to map remote users to local user profiles.

Use Name ID as User ID

When selected, fall back to using the name identifier from the assertion to find the user.

Artifact Message Encoding

Encoding

Specifies the message encoding format for artifacts.

Transient User

Transient User

Specifies the user profile to map all identity provider users when sending transient name identifiers.

URL

Local Authentication URL

Specifies the local login URL.

Intermediate URL

Specifies a URL to which the user is redirected after authentication but before the original URL requested.

External Application Logout URL

Specifies the URL to which to send an HTTP POST including all cookies when receiving a logout request. To add a user session property as a POST parameter, include it in the URL query string as a `appsessionproperty` parameter.

Default Relay State URL

Default Relay State URL

Specifies the URL to which to redirect users after the request has been handled. Used if not specified in the response.

Adapter

Adapter

Specifies a class that implements the `FederationSPAdapter` interface and performs application specific processing during the federation process.

Adapter Environment

Specifies environment variables passed to the adapter class.

5.2.2.3. Services

The following properties appear under the Services tab:

MetaAlias

MetaAlias

Used to locate the hosted provider's entity identifier, specified as `[/realm-name]*/provider-name`, where `provider-name` can not contain slash characters (/). For example: `/myRealm/mySubrealm/sp`.

SP Service Attributes

Single Logout Service

Specifies the end points to handle single logout, depending on the SAML binding selected.

Manage NameID Service

Specifies the end points to handle name identifiers, depending on the SAML binding selected.

Assertion Consumer Service

Specifies the end points to consume assertions, with Index corresponding to the index of the URL in the standard metadata.

5.2.2.4. Advanced Settings

The following properties appear under the Advanced tab:

SAE Configuration

SP URL

Specifies the end point to handle Secure Attribute Exchange requests.

SP Logout URL

Specifies the end point of the service provider that can handle global logout requests.

Application Security Configuration

Specifies how to handle encryption for Secure Attribute Exchange operations.

ECP Configuration

Request IDP List Finder Implementation

Specifies a class that returns a list of preferred identity providers trusted by the SAML Enhanced Client or Proxy profile.

Request IDP List Get Complete

Specifies a URI reference used to retrieve the complete identity provider list if the `IDPList` element is not complete.

Request IDP List

Specifies a list of identity providers for the SAML Enhanced Client or Proxy to contact, used by the default implementation of the IDP Finder.

IDP Proxy

IDP Proxy

When enabled, allow proxied authentication for this service provider.

Introduction

When enabled, use introductions to find the proxy identity provider.

Proxy Count

Specifies the maximum number of proxy identity providers.

IDP Proxy List

Specifies a list of URIs identifying preferred proxy identity providers.

Session Synchronization

Enabled

When enabled, the service provider sends a SOAP logout request over the back channel to all identity providers when a session times out. A session may time out when the maximum idle time or maximum session time is reached, for example.

Relay State URL List

Relay State URL List

List of URLs permitted for the `RelayState` parameter. OpenAM validates the redirection URL in the `RelayState` parameter against this list. If the `RelayState` parameter's value is in the list, OpenAM allows redirection to the `RelayState` URL. If it is not in the list, a browser error occurs.

Use the pattern matching rules described in Section 8.1.3, "Constraining Post-Login Redirects" in the *Authentication and Single Sign-On Guide* to specify URLs in the list.

If you do not specify any URLs in this property, OpenAM does not validate the `RelayState` parameter.

5.2.3. Circle of Trust Configuration Properties

Once you have set up a circle of trust, you can configure it through the AM console under Realms > *`Realm Name`* > Applications > SAML > Circle of Trust > *`Circle of Trust Name`*.

Name

String to refer to the circle of trust.

Description

Short description of the circle of trust.

IDFF Writer Service URL

Liberty Identity Federation Framework service that writes identity provider entity identifiers to Common Domain cookies after successful authentication, used in identity provider discovery. Example: `http://www.disco.example:8080/openam/idffwriter`.

IDFF Reader Service URL

Liberty Identity Federation Framework service that reads identity provider entity identifiers from Common Domain cookies, used in identity provider discovery. Example: `http://www.disco.example:8080/openam/transfer`.

SAML2 Writer Service URL

SAML v2.0 service that writes identity provider entity identifiers to Common Domain cookies after successful authentication, used in identity provider discovery. Example: `http://www.disco.example:8080/openam/saml2writer`.

SAML2 Reader Service URL

SAML v2.0 service that reads identity provider entity identifiers from Common Domain cookies, used in identity provider discovery. Example: `http://www.disco.example:8080/openam/saml2reader`.

Status

Whether this circle of trust is operational.

Realm

Name of the realm participating in this circle of trust.

Entity Providers

Known hosted and remote identity and service providers participating in this circle of trust.

5.3. SAML v2.0 Global Services Properties

This section covers services that include global federation properties for OpenAM. Navigate to Configure > Global Services to configure these properties.

5.3.1. SAML v2.0 Service Configuration

ssoadm service name: `saml2`

5.3.1.1. Global Attributes

The following settings appear on the **Global Attributes** tab:

Cache cleanup interval (in seconds)

Time between cache cleanup operations, in seconds.

Default value: `600`

ssoadm attribute: `cacheCleanupInterval`

Attribute name for Name ID information

User entry attribute to store name identifier information.

Default value: `sun-fm-saml2-nameid-info`

ssoadm attribute: `nameIDInfoAttribute`

Attribute name for Name ID information key

User entry attribute to store the name identifier key.

Default value: `sun-fm-saml2-nameid-infokey`

ssoadm attribute: `nameIDInfoKeyAttribute`

Cookie domain for IdP Discovery Service

Specifies the cookie domain for the IDP discovery service.

Default value: `openam.example.com`

ssoadm attribute: `idpDiscoveryCookieDomain`

Cookie type for IdP Discovery Service

Specifies the cookie type to use.

The possible values for this property are:

```
PERSISTENT
SESSION
```

Default value: `PERSISTENT`

ssoadm attribute: `idpDiscoveryCookieType`

URL scheme for IdP Discovery Service

Specifies the URL scheme to use.

The possible values for this property are:

```
http
https
```

Default value: `https`

ssoadm attribute: `idpDiscoveryUrlSchema`

XML Encryption SPI implementation class

Used by the SAML2 engine to *encrypt* and *decrypt* documents.

Default value: `com.sun.identity.saml2.xmlenc.FMEncProvider`

ssoadm attribute: `xmlEncryptionClass`

Include xenc:EncryptedKey inside ds:KeyInfo Element

Specify whether to include the `xenc:EncryptedKey` property inside the `ds:KeyInfo` element.

Default value: `true`

ssoadm attribute: `encryptedKeyInKeyInfo`

XML Signing SPI implementation class

Used by the SAML2 engine to *sign* documents.

Default value: `com.sun.identity.saml2.xmlsig.FMSigProvider`

ssoadm attribute: `xmlSigningClass`

XML Signing Certificate Validation

If enabled, then validate certificates used to sign documents.

Default value: `false`

ssoadm attribute: `signingCertValidation`

CA Certificate Validation

If enabled, then validate CA certificates.

Default value: `false`

ssoadm attribute: `caCertValidation`

Enable SAML v2.0 failover

If enabled, OpenAM can failover SAML v2.0 requests to another instance.

Default value: `false`

ssoadm attribute: `failOverEnabled`

Buffer length (in bytes) to decompress request

Specify the size of the buffer used for decompressing requests, in bytes.

Default value: `2048`

ssoadm attribute: `bufferLength`

5.3.1.2. Realm Defaults

The following settings appear on the *Realm Defaults* tab:

Metadata signing key alias

Specify the private key alias to be used to sign the given entity's metadata when requesting signed metadata, either by using exportmetadata.jsp or the `ssoadm` command.

Default value: `test`

ssoadm attribute: `metadataSigningKey`

Metadata signing key password

Specify the password used to retrieve the signing key from the keystore.

ssoadm attribute: `metadataSigningKeyPass`

5.3.2. SAML v2.0 SOAP Binding

ssoadm service name: `federation/saml2soapbinding`

The following settings are available in this service:

Request Handler List

List of handlers to deal with SAML v2.0 requests bound to SOAP.

The required format is: `key=Meta Alias|class=Handler Class`

Set the *key* property for a request handler to the meta alias, and the *class* property to the name of the class that implements the handler.

For example: `key=/pdp|class=com.sun.identity.xacml.plugins.XACMLAuthzDecisionQueryHandler`

ssoadm attribute: `requestHandlers`

5.3.3. Multi-Federation Protocol

ssoadm service name: `federation/multi`

The following settings are available in this service:

Single Logout Handler List

List of Logout handlers for each supported federation protocol

The multi-federation protocol engine supports Single Logout. Each federation protocol requires a different single logout handler. Logout handler must implement the `com.sun.identity.multiprotocol.SingleLogoutHandler` interface.

Default value:

```
key=IDFF|class=com.sun.identity.multiprotocol.IDFFSingleLogoutHandler
key=WSFED|class=com.sun.identity.multiprotocol.WSFederationSingleLogoutHandler
key=SAML2|class=com.sun.identity.multiprotocol.SAML2SingleLogoutHandler
```

ssoadm attribute: `singleLogoutHandlerList`

5.3.4. Common Federation Configuration

ssoadm service name: `federation/common`

5.3.4.1. General Configuration

The following settings appear on the **General Configuration** tab:

Maximum allowed content length

The maximum content length allowed in federation communications, in bytes.

Default value: `20480`

ssoadm attribute: `maxContentLength`

Check presence of certificates

Enable checking of certificates against local copy

Whether to verify that the partner's signing certificate included in the Federation XML document is the same as the one stored in the said partner's meta data.

The possible values for this property are:

```
off
on
```

Default value: `on`

ssoadm attribute: `certificateChecking`

SAML Error Page URL

OpenAM redirects users here when an error occurs in the SAML2 engine.

Both relative and absolute URLs are supported. Users are redirected to an absolute URL using the configured HTTP Binding whereas relative URLs are displayed within the request.

Default value: `/saml2/jsp/saml2error.jsp`

ssoadm attribute: `samlErrorPageUrl`

SAML Error Page HTTP Binding

The possible values are HTTP-Redirect or HTTP-POST.

Default value: `HTTP-POST`

ssoadm attribute: `samlErrorPageHttpBinding`

5.3.4.2. Implementation Classes

The following settings appear on the **Implementation Classes** tab:

Datastore SPI implementation class

The Federation system uses this class to get/set user profile attributes.

The default implementation uses the Identity repository APIs to access user profile attributes. A custom implementation must implement the `com.sun.identity.plugin.datastore.DataStoreProvider` interface.

Default value: `com.sun.identity.plugin.datastore.impl.IdRepoDataStoreProvider`

ssoadm attribute: `datastoreClass`

ConfigurationInstance SPI implementation class

The Federation system uses this class to fetch service configuration.

The default implementation uses the SMS APIs to access service configuration. A custom implementation must implement the `com.sun.identity.plugin.configuration.ConfigurationInstance` interface.

Default value: `com.sun.identity.plugin.configuration.impl.ConfigurationInstanceImpl`

ssoadm attribute: `configurationClass`

Logger SPI implementation class

The Federation system uses this class to record log entries.

The default implementation uses the Logging APIs to record log entries. A custom implementation must implement the `com.sun.identity.plugin.log.Logger` interface.

Default value: `com.sun.identity.plugin.log.impl.LogProvider`

ssoadm attribute: `loggerClass`

SessionProvider SPI implementation class

The Federation system uses this class to interface with the session service.

The default implementation uses the standard authentication and SSO APIs to access the session service. A custom implementation must implement the `com.sun.identity.plugin.session.SessionProvider` interface.

Default value: `com.sun.identity.plugin.session.impl.FMSessionProvider`

ssoadm attribute: `sessionProviderClass`

PasswordDecoder SPI implementation class

The Federation system uses this class to decode password encoded by OpenAM.

The default implementation uses the internal OpenAM decryption API to decode passwords. A custom implementation must implement the `com.sun.identity.saml.xmlsig.PasswordDecoder` interface.

Default value: `com.sun.identity.saml.xmlsig.FMPasswordDecoder`

ssoadm attribute: `passwordDecoderClass`

SignatureProvider SPI implementation class

The Federation system uses this class to digitally sign SAML documents.

The default implementation uses the XERCES APIs to sign the documents. A custom implementation must implement the `com.sun.identity.saml.xmlsig.SignatureProvider` interface.

Default value: `com.sun.identity.saml.xmlsig.AMSignatureProvider`

ssoadm attribute: `signatureProviderClass`

KeyProvider SPI implementation class

The Federation system uses this class to provide access to the underlying Java keystore.

The default implementation uses the Java Cryptographic Engine to provide access to the Java keystore. A custom implementation must implement the `com.sun.identity.saml.xmlsig.KeyProvider` interface.

Default value: `com.sun.identity.saml.xmlsig.JKSKeyProvider`

ssoadm attribute: `keyProviderClass`

5.3.4.3. Algorithms

The following settings appear on the **Algorithms** tab:

XML canonicalization algorithm

The algorithm used to canonicalize XML documents.

The possible values for this property are:

```
http://www.w3.org/2001/10/xml-exc-c14n#
http://www.w3.org/2001/10/xml-exc-c14n#WithComments
http://www.w3.org/TR/2001/REC-xml-c14n-20010315
http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments
```

Default value: `http://www.w3.org/2001/10/xml-exc-c14n#`

ssoadm attribute: `canonicalizationAlgorithm`

XML signature algorithm

The algorithm used to sign XML documents.

The possible values for this property are:

```
http://www.w3.org/2000/09/xmldsig#rsa-sha1
http://www.w3.org/2000/09/xmldsig#hmac-sha1
http://www.w3.org/2000/09/xmldsig#dsa-sha1
http://www.w3.org/2001/04/xmldsig-more#rsa-md5
```

```
http://www.w3.org/2001/04/xmldsig-more#rsa-ripemd160
http://www.w3.org/2001/04/xmldsig-more#rsa-sha256
http://www.w3.org/2001/04/xmldsig-more#rsa-sha384
http://www.w3.org/2001/04/xmldsig-more#rsa-sha512
http://www.w3.org/2001/04/xmldsig-more#hmac-md5
http://www.w3.org/2001/04/xmldsig-more#hmac-ripemd160
http://www.w3.org/2001/04/xmldsig-more#hmac-sha256
http://www.w3.org/2001/04/xmldsig-more#hmac-sha384
http://www.w3.org/2001/04/xmldsig-more#hmac-sha512
```

Default value: `http://www.w3.org/2000/09/xmldsig#rsa-sha1`

ssoadm attribute: `signatureAlgorithm`

XML digest algorithm

The default digest algorithm to use in signing XML.

The possible values for this property are:

```
http://www.w3.org/2000/09/xmldsig#sha1
http://www.w3.org/2001/04/xmlenc#sha256
http://www.w3.org/2001/04/xmlenc#sha512
http://www.w3.org/2001/04/xmldsig-more#sha384
```

Default value: `http://www.w3.org/2000/09/xmldsig#sha1`

ssoadm attribute: `DigestAlgorithm`

Query String signature algorithm (RSA)

The default signature algorithm to use in case of RSA keys.

The possible values for this property are:

```
http://www.w3.org/2000/09/xmldsig#rsa-sha1
http://www.w3.org/2001/04/xmldsig-more#rsa-sha256
http://www.w3.org/2001/04/xmldsig-more#rsa-sha384
http://www.w3.org/2001/04/xmldsig-more#rsa-sha512
```

Default value: `http://www.w3.org/2000/09/xmldsig#rsa-sha1`

ssoadm attribute: `QuerySignatureAlgorithmRSA`

Query String signature algorithm (DSA)

The default signature algorithm to use in case of DSA keys.

The possible values for this property are:

```
http://www.w3.org/2000/09/xmldsig#dsa-sha1
```

Default value: `http://www.w3.org/2000/09/xmldsig#dsa-sha1`

ssoadm attribute: `QuerySignatureAlgorithmDSA`

Query String signature algorithm (EC)

The default signature algorithm to use in case of EC keys.

The possible values for this property are:

```
http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha1
http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha256
http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha384
http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha512
```

Default value: `http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha512`

ssoadm attribute: `QuerySignatureAlgorithmEC`

XML transformation algorithm

The algorithm used to transform XML documents.

The possible values for this property are:

```
http://www.w3.org/2001/10/xml-exc-c14n#
http://www.w3.org/2001/10/xml-exc-c14n#WithComments
http://www.w3.org/TR/2001/REC-xml-c14n-20010315
http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments
http://www.w3.org/TR/1999/REC-xslt-19991116
http://www.w3.org/2000/09/xmldsig#base64
http://www.w3.org/TR/1999/REC-xpath-19991116
http://www.w3.org/2000/09/xmldsig#enveloped-signature
http://www.w3.org/TR/2001/WD-xptr-20010108
http://www.w3.org/2002/04/xmldsig-filter2
http://www.w3.org/2002/06/xmldsig-filter2
http://www.nue.et-inf.uni-siegen.de/~geuer-pollmann/#xpathFilter
```

Default value: `http://www.w3.org/2001/10/xml-exc-c14n#`

ssoadm attribute: `transformationAlgorithm`

5.3.4.4. Monitoring

The following settings appear on the **Monitoring** tab:

Monitoring Agent Provider Class

The Federation system uses this class to gain access to the monitoring system.

The default implementation uses the built-in OpenAM monitoring system. A custom implementation must implement the `com.sun.identity.plugin.monitoring.FedMonAgent` interface.

Default value: `com.sun.identity.plugin.monitoring.impl.AgentProvider`

ssoadm attribute: `monitoringAgentClass`

Monitoring Provider Class for SAML1

The SAMLv1 engine uses this class to gain access to the monitoring system

The default implementation uses the built-in OpenAM monitoring system. A custom implementation must implement the `com.sun.identity.plugin.monitoring.FedMonSAML1Svc` interface.

Default value: `com.sun.identity.plugin.monitoring.impl.FedMonSAML1SvcProvider`

ssoadm attribute: `monitoringSaml1Class`

Monitoring Provider Class for SAML2

The SAML2 engine uses this class to gain access to the monitoring system.

The default implementation uses the built-in OpenAM monitoring system. A custom implementation must implement the `com.sun.identity.plugin.monitoring.FedMonSAML2Svc` interface.

Default value: `com.sun.identity.plugin.monitoring.impl.FedMonSAML2SvcProvider`

ssoadm attribute: `monitoringSaml2Class`

Monitoring Provider Class for ID-FF

The ID-FF engine uses this class to gain access to the monitoring system.

The default implementation uses the built-in OpenAM monitoring system. A custom implementation must implement the `com.sun.identity.plugin.monitoring.FedMonIDFFSvc` interface.

Default value: `com.sun.identity.plugin.monitoring.impl.FedMonIDFFSvcProvider`

ssoadm attribute: `monitoringIdffClass`

Appendix A. Getting Support

For more information or resources about OpenAM and ForgeRock Support, see the following sections:

A.1. Accessing Documentation Online

ForgeRock publishes comprehensive documentation online:

- The ForgeRock Knowledge Base offers a large and increasing number of up-to-date, practical articles that help you deploy and manage ForgeRock software.
- ForgeRock core documentation, such as this document, aims to be technically accurate and complete with respect to the software documented. It is visible to everyone and covers all product features and examples of how to use them.

Core documentation therefore follows a three-phase review process designed to eliminate errors:

- Product managers and software architects review project documentation design with respect to the readers' software lifecycle needs.
- Subject matter experts review proposed documentation changes for technical accuracy and completeness with respect to the corresponding software.
- Quality experts validate implemented documentation changes for technical accuracy, completeness in scope, and usability for the readership.

The review process helps to ensure that documentation published for a ForgeRock release is technically accurate and complete.

Fully reviewed, published core documentation is available at http://backstage.forgerock.com/. Use this documentation when working with a ForgeRock Identity Platform release.

A.2. Joining the ForgeRock Community

Visit the Community resource center where you can find information about each project, download trial builds, browse the resource catalog, ask and answer questions on the forums, find community events near you, and find the source code for open source software.

A.3. Getting Support and Contacting ForgeRock

ForgeRock provides support services, professional services, classes through ForgeRock University, and partner services to assist you in setting up and maintaining your deployments. For a general overview of these services, see https://www.forgerock.com.

ForgeRock has staff members around the globe who support our international customers and partners. For details, visit https://www.forgerock.com, or send an email to ForgeRock at info@forgerock.com.

Glossary

Access control	Control to grant or to deny access to a resource.
Account lockout	The act of making an account temporarily or permanently inactive after successive authentication failures.
Actions	Defined as part of policies, these verbs indicate what authorized subjects can do to resources.
Advice	In the context of a policy decision denying access, a hint to the policy enforcement point about remedial action to take that could result in a decision allowing access.
Agent administrator	User having privileges only to read and write policy agent profile configuration information, typically created to delegate policy agent profile creation to the user installing a policy agent.
Agent authenticator	Entity with read-only access to multiple agent profiles defined in the same realm; allows an agent to read web service profiles.
Application	In general terms, a service exposing protected resources. In the context of OpenAM policies, the application is a template that constrains the policies that govern access to protected resources. An application can have zero or more policies.
Application type	Application types act as templates for creating policy applications. Application types define a preset list of actions and functional logic, such as policy lookup and resource comparator logic.

	Application types also define the internal normalization, indexing logic, and comparator logic for applications.
Attribute-based access control (ABAC)	Access control that is based on attributes of a user, such as how old a user is or whether the user is a paying customer.
Authentication	The act of confirming the identity of a principal.
Authentication chaining	A series of authentication modules configured together which a principal must negotiate as configured in order to authenticate successfully.
Authentication level	Positive integer associated with an authentication module, usually used to require success with more stringent authentication measures when requesting resources requiring special protection.
Authentication module	OpenAM authentication unit that handles one way of obtaining and verifying credentials.
Authorization	The act of determining whether to grant or to deny a principal access to a resource.
Authorization Server	In OAuth 2.0, issues access tokens to the client after authenticating a resource owner and confirming that the owner authorizes the client to access the protected resource. OpenAM can play this role in the OAuth 2.0 authorization framework.
Auto-federation	Arrangement to federate a principal's identity automatically based on a common attribute value shared across the principal's profiles at different providers.
Bulk federation	Batch job permanently federating user profiles between a service provider and an identity provider based on a list of matched user identifiers that exist on both providers.
Circle of trust	Group of providers, including at least one identity provider, who have agreed to trust each other to participate in a SAML v2.0 provider federation.
Client	In OAuth 2.0, requests protected web resources on behalf of the resource owner given the owner's authorization. OpenAM can play this role in the OAuth 2.0 authorization framework.
Conditions	Defined as part of policies, these determine the circumstances under which which a policy applies. Environmental conditions reflect circumstances like the client IP address, time of day, how the subject authenticated, or the authentication level achieved.

	Subject conditions reflect characteristics of the subject like whether the subject authenticated, the identity of the subject, or claims in the subject's JWT.
Configuration datastore	LDAP directory service holding OpenAM configuration data.
Cross-domain single sign-on (CDSSO)	OpenAM capability allowing single sign-on across different DNS domains.
Delegation	Granting users administrative privileges with OpenAM.
Entitlement	Decision that defines which resource names can and cannot be accessed for a given subject in the context of a particular application, which actions are allowed and which are denied, and any related advice and attributes.
Extended metadata	Federation configuration information specific to OpenAM.
Extensible Access Control Markup Language (XACML)	Standard, XML-based access control policy language, including a processing model for making authorization decisions based on policies.
Federation	Standardized means for aggregating identities, sharing authentication and authorization data information between trusted providers, and allowing principals to access services across different providers without authenticating repeatedly.
Fedlet	Service provider application capable of participating in a circle of trust and allowing federation without installing all of OpenAM on the service provider side; OpenAM lets you create Java Fedlets.
Hot swappable	Refers to configuration properties for which changes can take effect without restarting the container where OpenAM runs.
Identity	Set of data that uniquely describes a person or a thing such as a device or an application.
Identity federation	Linking of a principal's identity across multiple providers.
Identity provider (IdP)	Entity that produces assertions about a principal (such as how and when a principal authenticated, or that the principal's profile has a specified attribute value).
Identity repository	Data store holding user profiles and group information; different identity repositories can be defined for different realms.
Java EE policy agent	Java web application installed in a web container that acts as a policy agent, filtering requests to other applications in the container with policies based on application resource URLs.

Metadata	Federation configuration information for a provider.
Policy	Set of rules that define who is granted access to a protected resource when, how, and under what conditions.
Policy Agent	Agent that intercepts requests for resources, directs principals to OpenAM for authentication, and enforces policy decisions from OpenAM.
Policy Administration Point (PAP)	Entity that manages and stores policy definitions.
Policy Decision Point (PDP)	Entity that evaluates access rights and then issues authorization decisions.
Policy Enforcement Point (PEP)	Entity that intercepts a request for a resource and then enforces policy decisions from a PDP.
Policy Information Point (PIP)	Entity that provides extra information, such as user profile attributes that a PDP needs in order to make a decision.
Principal	Represents an entity that has been authenticated (such as a user, a device, or an application), and thus is distinguished from other entities. When a Subject successfully authenticates, OpenAM associates the Subject with the Principal.
Privilege	In the context of delegated administration, a set of administrative tasks that can be performed by specified subjects in a given realm.
Provider federation	Agreement among providers to participate in a circle of trust.
Realm	OpenAM unit for organizing configuration and identity information. Realms can be used for example when different parts of an organization have different applications and user data stores, and when different organizations use the same OpenAM deployment. Administrators can delegate realm administration. The administrator assigns administrative privileges to users, allowing them to perform administrative tasks within the realm.
Resource	Something a user can access over the network such as a web page. Defined as part of policies, these can include wildcards in order to match multiple actual resources.
Resource owner	In OAuth 2.0, entity who can authorize access to protected web resources, such as an end user.

Resource server	In OAuth 2.0, server hosting protected web resources, capable of handling access tokens to respond to requests for such resources.
Response attributes	Defined as part of policies, these allow OpenAM to return additional information in the form of "attributes" with the response to a policy decision.
Role based access control (RBAC)	Access control that is based on whether a user has been granted a set of permissions (a role).
Security Assertion Markup Language (SAML)	Standard, XML-based language for exchanging authentication and authorization data between identity providers and service providers.
Service provider (SP)	Entity that consumes assertions about a principal (and provides a service that the principal is trying to access).
Session	The interval that starts with the user authenticating through OpenAM and ends when the user logs out, or when their session is terminated. For browser-based clients, OpenAM manages user sessions across one or more applications by setting a session cookie. See also Stateful session and Stateless session.
Session high availability	Capability that lets any OpenAM server in a clustered deployment access shared, persistent information about users' sessions from the CTS token store. The user does not need to log in again unless the entire deployment goes down.
Session token	Unique identifier issued by OpenAM after successful authentication. For a Stateful session, the session token is used to track a principal's session.
Single log out (SLO)	Capability allowing a principal to end a session once, thereby ending her session across multiple applications.
Single sign-on (SSO)	Capability allowing a principal to authenticate once and gain access to multiple applications without authenticating again.
Site	Group of OpenAM servers configured the same way, accessed through a load balancer layer. The load balancer handles failover to provide service-level availability. Use sticky load balancing based on `amlbcookie` values to improve site performance. The load balancer can also be used to protect OpenAM services.
Standard metadata	Standard federation configuration information that you can share with other access management software.
Stateful session	An OpenAM session that resides in the Core Token Service's token store. Stateful sessions might also be cached in memory on one or

	more OpenAM servers. OpenAM tracks stateful sessions in order to handle events like logout and timeout, to permit session constraints, and to notify applications involved in SSO when a session ends.
Stateless session	An OpenAM session for which state information is encoded in OpenAM and stored on the client. The information from the session is not retained in the CTS token store. For browser-based clients, OpenAM sets a cookie in the browser that contains the session information.
Subject	Entity that requests access to a resource When a subject successfully authenticates, OpenAM associates the subject with the Principal that distinguishes it from other subjects. A subject can be associated with multiple principals.
User data store	Data storage service holding principals' profiles; underlying storage can be an LDAP directory service, a relational database, or a custom `IdRepo` implementation.
Web policy agent	Native library installed in a web server that acts as a policy agent with policies based on web page URLs.