SAML v2.0 Guide

/ ForgeRock Access Management 7.1.4

Latest update: 7.1.4

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 in ForgeRock® Access Management (AM). ForgeRock Access Management provides intelligent authentication, authorization, federation, and single sign-on functionality.

Table of Contents

Overview

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 Access Management for SAML v2.0 identity and service providers, and for anyone using the Fedlet as a SAML v2.0 service provider.

Quick Start

About SAML v2.0 Learn how AM servers can function as both the *identity provider* (IDP) and *service provider* (SP) in SAML v2.0 circles of trust (CoT).	Configuring SAML v2.0 Configure AM's SAML v2.0 support by using the AM console.	Configuring Single Sign-on Enable SAML v2.0 single sign-on (SSO) and single logout (SLO) so that your users can access multiple, independent services by logging in once with a single set of credentials.
Federate Identities Learn how to link identities in the identity provider with those at the service providers, either permanently or temporarily.	SAML v2.0 in Java Apps Learn how to use the AM *Fedlet* to integrate SAML v2.0 in a Java application, allowing it to act as a lightweight SAML v2.0 service provider.	SAML v2.0 Secure Attribute Exchange Discover how you can create a deployment where AM acts as a SAML v2.0 gateway to a legacy application that serves as an identity provider.

About ForgeRock Identity Platform™ Software

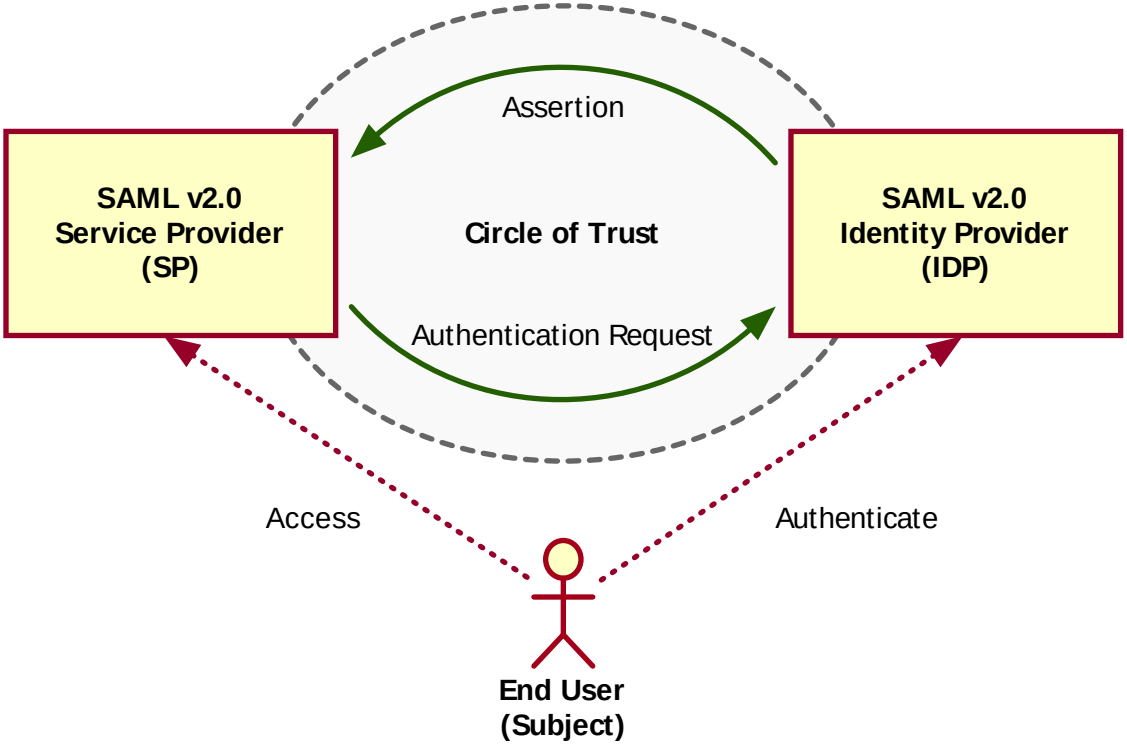
ForgeRock Identity Platform™ serves as the basis for our simple and comprehensive Identity and Access Management solution. We help our customers deepen their relationships with their customers, and improve the productivity and connectivity of their employees and partners. For more information about ForgeRock and about the platform, see https://www.forgerock.com.

Chapter 1

Introducing SAML v2.0

SAML v2.0 helps organizations share, or *federate* identities and services, without having to manage the identities or credentials themselves. The credentials are managed by a single entity, known as the *identity provider*. The services are provided by *service providers*. Both providers are configured to trust one another.

Overview of SAML v2.0 in AM



AM uses the concept of the *circle of trust* to manage the relationship between IDPs and SPs.

+ *SAML v2.0 Concepts*

Security Assertion Markup Language (SAML) v2.0 is a standard that enables users to access multiple services using only a single set of credentials. The services may be provided by different organizations, using multiple domains. In summary, SAML v2.0 provides cross-domain single sign-on (CDSSO).

SAML v2.0 Terminology

Term	Description
End User	The person who is attempting to access the resource or application. In SAML v2.0, the end user is often referred to as the *subject*. The end user uses a *user-agent*, usually a web-browser, when performing a SAML v2.0 flow.
Single Sign-On (SSO)	The ability for an end user to authenticate once but gain access to multiple applications, without having to authenticate separately to each one.
Single Log Out (SLO)	The ability for an end user to log out once but terminate sessions in multiple applications, without having to log out separately from each one.
Assertions	An assertion is a set of statements about an authenticated user that let services make authorization decisions, that is; whether to allow that user to access the service, and what functionality they can use. SAML assertions are XML-formatted tokens. Assertions issues by AM may contain the following pieces of information about an end user: 1. Their attributes, such as pieces of information from the user's profile. 2. The level of authentication they have performed.
Identity Provider (IDP)	The identity provider is responsible for authenticating end users, managing their account, and issuing SAML assertions about them.
Service Provider (SP)	The provider of the service or application that the end user is trying to access. The service provider has a trust relationship with the identity provider, which enables the SP to rely on the assertions it receives from the IDP.
Circle of Trust (CoT)	A circle of trust is an AM concept that groups at least one identity provider and at least one service provider who agree to share authentication information.
Metadata	Providers in SAML v2.0 share *metadata*, which represents the configuration of the provider, as well as the mechanisms it can use to communicate with other providers. For example, the metadata may contain necessary certificates for signing verification, as well as which of the SAML v2.0 bindings are supported. Sharing metadata greatly simplifies the creation of SAML v2.0 providers in a circle of trust. AM can import the XML-formatted metadata provided by other providers, which are referred to as *remote* providers. You can also export the

Term	Description
	metadata from providers created in an AM instance, referred to as *hosted* providers. For more information about metadata, see Metadata for the OASIS Security Assertion Markup Language (SAML) V2.0 in the *SAML V2.0 Standard*.

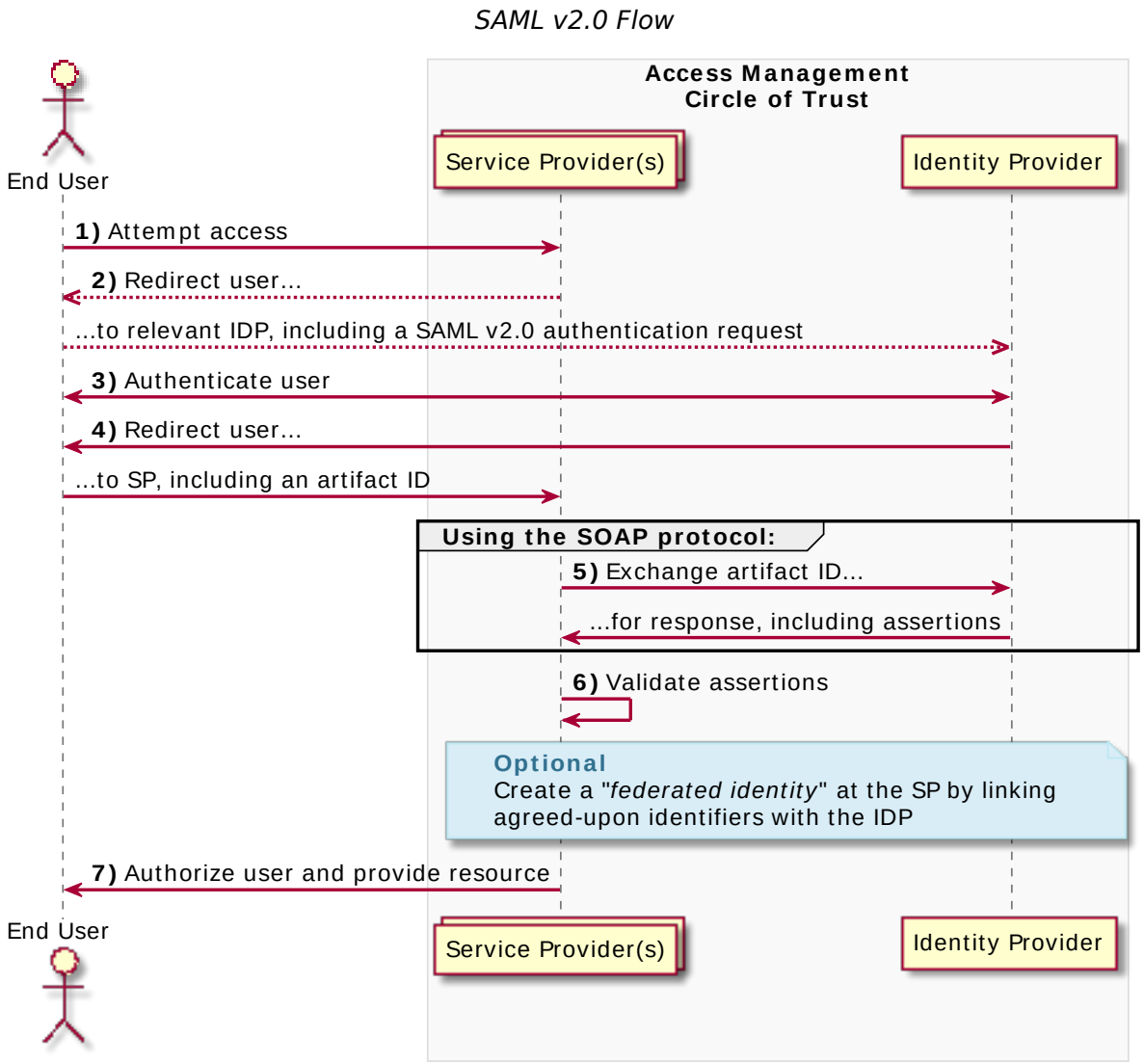
For more information, see Security Assertion Markup Language (SAML) v2.0.

When configuring AM to provide single sign-on using SAML v2.0, you can map accounts at the identity provider to accounts at the service provider, including mapping to an anonymous user.

The identity provider can then make assertions to the service provider, for example, to attest that the end 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.

+ *SAML v2.0 Example Flow*



1. An unauthenticated user attempts to access a SAML v2.0 service provider.

2. The service provider determines the identity provider associated with the end user, and redirects the user's browser to the IDP, using an HTTP 302 Redirect message. A SAML v2.0 authentication request is included in the query string.

This is an example of *HTTP-Redirect* binding, and is the default when requesting SAML authentication by AM. AM also supports the *HTTP-POST* binding for processing SAML v2.0 authentication requests.

AM provides two deployment models to support single sign-on (SSO) when contacting the SP initially. For details, see "*Implementing SSO and SLO*".

3. The identity provider validates the request, determines the authentication method that should be used, and authenticates the user.

   The SP may include certain requirements for the authentication it requires the user to perform in the authentication request, for example a requirement to use multiple factors.

4. The IDP creates a *SAML Artifact*, which contains a unique artifact ID for the SAML v2.0 response.

   The IDP redirects the end user's browser to the SP, and includes the SAML Artifact in the query parameters. Note that the browser only has access to the artifact ID rather than the SAML response itself, reducing risk of malicious interference.

5. The SP communicates directly with the IDP, using the SOAP protocol, to retrieve the SAML response relating to the artifact ID.

   The IDP returns the SAML response, including the assertion, using the SOAP protocol, directly to the SP.

   The information in the SAML response is not shared with the user agent. This is an example of *HTTP-Artifact* binding, and is the default when AM is returning SAML assertions. AM also supports the *HTTP-POST* binding for transmitting SAML v2.0 assertions.

6. The SP validates the SAML response, and that the signature matches the public key it holds for the IDP.

   Optionally, the SP can choose to create a new account locally for the user, or associate an identifier in the assertion with a user account it already has locally. Linked accounts are often referred to as a *federated identity*. See "*Federating Identities*".

> **Note**
>
> In order to link to an existing account, the SP may require the end user to also authenticate to the SP to determine the matching local account. Once linked, the user will only need to authenticate at the IDP when attempting access.

7. The SP can now use the information provided in the assertion, and any details in the local federated identity, to authorize the user, and decide whether to provide its services to the end user.

SAML v2.0 requires interoperability, and depends on standards for describing how the providers interact and exchange information. The SAML v2.0 standards describe the messages sent between providers, how they are relayed, how they are exchanged, and common use cases.

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

For information about implementing WS-Federation, refer to "How do I configure AM (All versions) as an Identity Provider for Microsoft Office 365 and Azure using WS-Federation?" in the *ForgeRock Knowledge Base*.

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*.

An AM 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 AM. For more information on fedlets, see "*Implementing SAML v2.0 Service Providers by Using Fedlets*".

Use case: Allow staff to access Google Workspace

A common use case for SAML v2.0 is to allow your internal staff to access and use the applications provided by Google Workspace, such as Google Docs and Google Sheets. This section highlights how AM provides the solution.

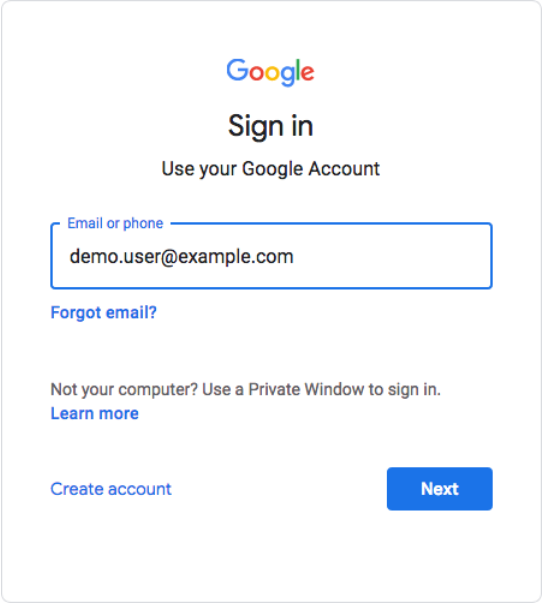
In this scenario, Google acts as the service provider, and AM acts as the identity provider for a hypothetical organization; *Example.com*.

+ *High-Level Steps to Let Staff Access G Suit Applications*

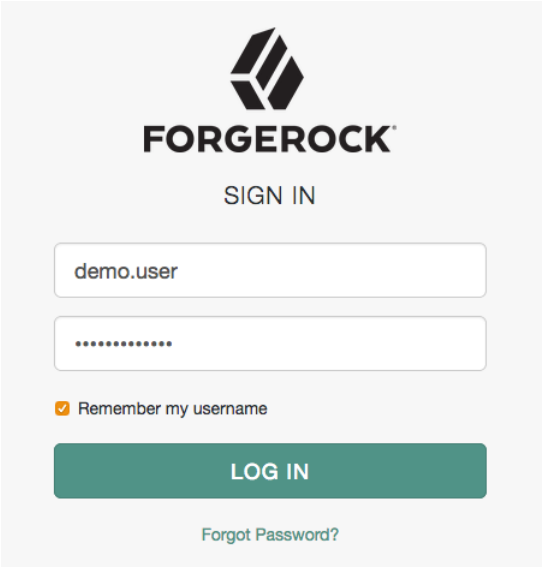
1. In AM, an administrative user configures an AM instance as the IDP.
2. The administrative user then configures Google Workspace as a remote SP, and provides the values that Google requires to use AM as its IDP. For example, the log in and log out URL, profile management URLs, and validation certificate.

3. The Google Workspace administrator enters the provided URLs and certificate into the Google Workspace Admin console for the domain being configured.

4. After configuration is complete, users attempting to access a Google Workspace service will be asked for their corporate email address:

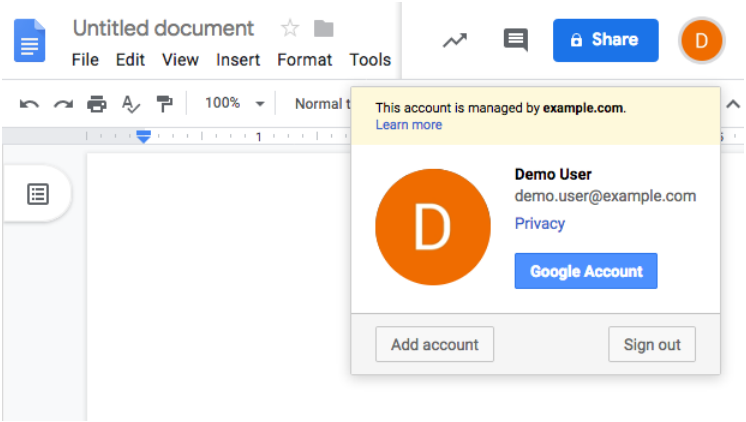


5. Based on the domain of the email address, Google redirects the user to sign in to AM, acting as the IDP:



6. After successfully authenticating with AM, the user is redirected back to the Google Workspace application; for example, Google Docs.

   Google, acting as the SP, creates a federated identity in its systems to manage local account options, such as privacy settings. The user can now access any of the Google Workspace apps, by authenticating to the IDP using their corporate *Example.com* account:



Chapter 2

Deployment Considerations

Before you set up SAML v2.0 in AM, you should:

- Know which providers will participate in circles of trust.
- Know how AM installations act as identity providers or service providers.
- 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 AM configuration directory. You can use the Java **keytool** command.

  For more information about AM keystores, including location and different types of keystores available and how to change the default keys, see "*Configuring Secrets, Certificates, and Keys*" in the *Security Guide*.
- Agree with other providers on a synchronized time service.
- Determine whether your session state configuration limits your usage of certain SAML v2.0 profiles. For more information, see "Session State Considerations".
- Consider increasing DS search size limits if you intend to create a large number of SAML v2.0 entities.

  To ensure searches of defined entities work as expected, configure the DS search property `ds-rlim-lookthrough-limit`.

  For more information, refer to *Important Considerations* in Issues with upgrades, Amster imports or exports, or registering clients (OAuth 2.0, OIDC and RADIUS) or Agents with reference to sunserviceID in AM in the *ForgeRock Knowledge Base*.

Session State Considerations

SAML v2.0 uses a combination of the CTS and browser-based data to store the progress of SAML v2.0 single sign-on SSO operations.

This combination lets AM instances to continue *single sign-on* flows that started at a different instance, without needing sticky load balancing in most scenarios.

Single sign-on progress is stored in a JSON web token (JWT) in the browser's local storage. The browser must support the `sessionStorage` API to handle SSO without the need for sticky load balancing of the AM instances. You **must** configure sticky load balancing to support SAML v2.0 SSO with clients that *do not* support session storage, and on IdP proxy implementations.

Session storage is similar to local storage but is more limited:

- Session storage exists only within the current browser tab.
- Another tab that displays the same page will have a different session storage.
- Session storage is shared between frames in the same tab (assuming they come from the same origin).
- Session storage data survives a page refresh, but not closing and opening the tab.

Tip

You can enable session storage support in WebView components on Android by using the following property:

```
settings.setDomStorageEnabled(true)
```

You cannot use session storage when using multiple WebView components simultaneously. For more information, see WebSettings - setDomStorageEnabled in the *Android Developers* documentation.

The JWT created in the browser's session storage is encrypted using the `am.global.services.saml2.client.storage.jwt.encryption` secret ID, which by default is mapped to the `directenctest` certificate alias. For more information, see "*Configuring Secrets, Certificates, and Keys*" in the *Security Guide*.

Performing single log out (SLO) operations with more than one AM instance has the following caveats:

- AM instances cache information about SLO progress in memory. After the initial request, you must send each subsequent request in an SLO flow to the same instance; for example, by enabling sticky load balancing.
- Use the HTTP-POST or HTTP-Redirect bindings for SAML v2.0 single logout (SLO). The SOAP binding is *not* supported.

In addition, browsers allow cookie sizes between 4,000 and 5,200 bytes, depending on the browser. If you are using client-based sessions, some SAML v2.0 features may cause the cookie to surpass the browser's supported cookie size; such as:

- In standalone mode, when performing single sign-on the IDP adds the full login URL (`FullLoginURL` property) to the session cookie, which includes the authentication request data, adding to cookie size.
- In integrated mode, AM adds to the session those SAML v2.0 attributes mapped to AM attributes.

If a client-based session cookie exceeds the supported size in any of these cases, you can configure a custom post-authentication plugin that removes unwanted properties or attributes, at the realm level. Note that removing properties or attributes in a custom SAML v2.0 service provider adapter is *not* supported.

For more information about post-authentication plugins, see "Creating Post-Authentication Plugins for Chains" in the *Authentication and Single Sign-On Guide*.

The following table summarizes the high-level tasks required to configure SAML v2.0:

Task	Resources
Configure an SP, an IDP, and a CoT. The first step is deciding if AM is the SP, the IDP, or both, and/or what metadata you need to import from other providers. For example, if AM is the IDP for another service in your environment, you will have to import the metadata of the remote SP. Ensure the SPs and IDPs that work together share the same CoT.	"*Configuring IDPs, SPs, and CoTs*"
Make Sure Your Providers Are Secure Configure signing and encryption secrets for your environment.	"*Signing and Encryption*"
Deploy the IDP Discovery Service If you have more than one IDP in your CoT, the IDP discovery service acts like a proxy between the SPs and the IPDs.	"*Deploying the IdP Discovery Service*"
Configure your Environment for SSO and SLO AM provides two options for implementing SSO and SLO: integrated mode, and standalone mode. There are several considerations to make before deciding which mode is more appropriate for your environment.	"*Implementing SSO and SLO*"
Decide How to Federate Identities AM supports different ways to federate identities depending of the configuration, and whether they exist or not in the SP.	"*Federating Identities*"
Configure AM as a SAML v2.0 Gateway for Legacy Applications Use AM Secure Attribute Exchange and IG to integrate legacy applications into your SAML deployment.	"*Implementing a SAML v2.0 Gateway by Using Secure Attribute Exchange*"
Use Fedlets instead of SPs AM provides Fedlets, which are a small Java applications that can act as the SP without installing AM. They can redirect to AM for SSO, and to retrieve SAML assertions.	"*Implementing SAML v2.0 Service Providers by Using Fedlets*"

Chapter 3

Configuring IDPs, SPs, and CoTs

This section covers configuration tasks to implement SAML v2.0 in AM.

During setup, you share metadata for providers that you host with other providers in the circle of trust. You must also configure remote providers, by importing their metadata.

In AM terms, a *hosted* provider is one served by the current AM instance; a *remote* provider is one hosted elsewhere.

To Create a Circle of Trust

A circle of trust is an AM concept that groups at least one identity provider and at least one service provider who agree to share authentication information.

1. Go to Realms > *Realm Name* > Applications > Federation > Circles of Trust, and then click Add Circle of Trust.
2. Provide a name, and then click Create.
3. On the Circle of Trust page, in the Entity Providers property, select at least one IDP and one SP.

   Note that entity providers can be added at any time, if you have not yet created them.
4. Customize any other properties as required, and then click Save Changes.

   For information about circle of trust properties, see "Circle of Trust Configuration Properties".

To Create a Hosted Entity Provider

This procedure provides steps for creating a hosted identity or service providers using the AM console.

Note

You can create identity and service provider roles in the AM console.

To create other roles, AM provides the `/realm-config/federation/entityproviders/saml2` REST endpoint.

For information on how to view the details of this endpoint, see "*REST API Explorer*" in the *Getting Started with REST*.

1. Go to Realms > *Realm Name* > Dashboard, and then click SAML Applications.

2. Click the Add Entity Provider drop-down button, and then click Hosted.

3. Enter an Entity ID, and verify the Entity Provider Base URL value is correct.

   Note

   AM truncates sequences of whitespace with a single whitespace character in values such as entity IDs. For example, if `ID value` (with one space) exists already, a new entity with the same name but multiple spaces would result in an error because the string values are treated as identical.

   AM uses the Entity Provider Base URL value for all SAML v2.0 related endpoints, so ensure other entities in your SAML deployment are able to access the specified URL.

4. In the Meta Aliases section, provide a URL-friendly value in either the Identity Provider Meta Alias, the Service Provider Meta Alias property, or both.

   Ensure the aliases for providers are unique in a circle of trust, as well as in the realm.

5. Click Create.

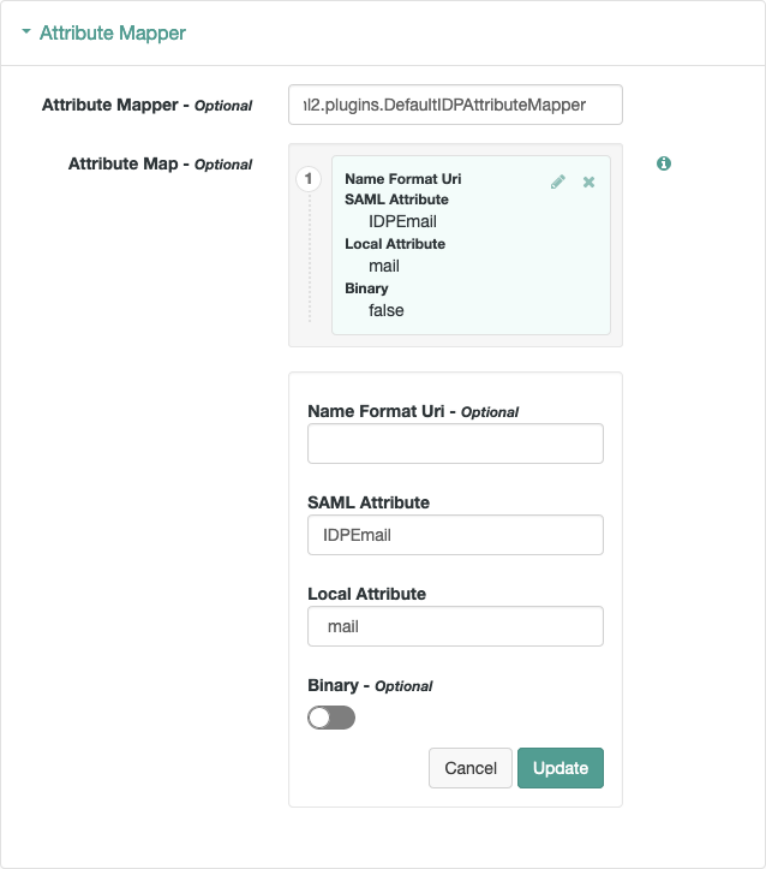
   + *How Do I Switch Between SP and IDP Configuration for a Given Provider?*

   AM only displays the configuration of a single role. Click on the labels to select the role view:

6. On the Assertion Processing tab, in the Attribute Mapper section, map SAML attribute names (Name in Assertion), to local attribute names.

Mapping SAML Attributes to Local Attributes



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 SAML Attribute with `"staticPartnerIDValue"` as the Local Attribute Name.

- Select the binary option when dealing with binary attribute values; for example, values that are Base64 encoded.

- Use the optional `Name Format Uri` property as required by the remote provider. For example, you may need to specify `urn:oasis:names:tc:SAML:2.0:attrname-format:uri`.

7. Customize any other properties as required, and then click Save Changes.

   For information about hosted entity provider properties, see:

   - "Hosted Identity Provider Configuration Properties"
   - "Hosted Service Provider Configuration Properties"

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

```
$ curl \
--output metadata.xml \
"https://openam.example.com:8443/openam/saml2/jsp/exportmetadata.jsp\
?entityid=myHostedProvider\
&realm=/mySubRealm"
```

   When you have configured your provider in the Top Level Realm, omit the `realm` query parameter.

   You may also be able to provide the URL in the above example to remote providers, if they can load the metadata by using a URL rather than a file.

To Import and Configure a Remote Entity Provider

The following procedure provides steps for importing and configuring one or more remote entity providers:

1. Obtain the entity provider metadata as an XML-formatted file.

2. Go to Realms > *Realm Name* > Dashboard, and then click SAML Applications.

3. Click the Add Entity Provider drop-down button, and then click Remote.

4. On the New Remote Entity Provider page, perform one of the following steps to import the XML file:

   - Drag and drop the XML file into the dotted box.
   - Click within the dotted box to open a file browser to select the XML file.

   Note

   You can import multiple remote entities in a single operation, as long as the entity ID is unique within each.

AM truncates sequences of whitespace with a single whitespace character in values such as entity IDs. For example, if `ID value` (with one space) exists already, a new entity with the same name but multiple spaces would result in an error because the string values are treated as identical.

5. (Optional) If you have already created a Circle of Trust, you can add the remote providers into one or more of them by using the Circles of Trust property.

6. Click Create.

7. After importing meta data, to edit the configuration of an entity provider, go to Realms > *Realm Name* > Applications > Federation > Entity Providers, and select the entity provider to edit.

   + *How Do I Switch Between SP and IDP Configuration for a Given Provider?*

   AM only displays the configuration of a single role. Click on the labels to select the role view:

   For information about remote entity provider properties, see:

   - "Remote Identity Provider Configuration Properties"
   - "Remote Service Provider Configuration Properties"

Chapter 4

Signing and Encryption

By default, IdPs and SPs do not sign or encrypt SAML v2.0 messages. While this is useful for test and demo environments, you should secure your production and production-like environments.

+ *How Does Signing Work?*

When AM needs to sign a SAML request or response for the consumption of a remote entity provider, it will determine the signing algorithm, and optionally, the digest method, based on the following logic, as recommended by the SAML v2.0 Metadata Profile for AlgorithmSupport Version 1.0 specification:

1. AM retrieves the remote entity provider's metadata, and examines the role-specific extensions for a configured digest method, or signing algorithm.

2. If there is no role-specific algorithm configured, AM checks for algorithms configured in the entity provider-level extensions.

3. If signing algorithms are specified at either role-specific level or entity provider-level, but AM is unable to find a suitable key, it does not sign the element, and displays an error.

   Possible reasons that AM may not be able to locate a suitable signing key include:

   - **Algorithm Mismatch** - the signing algorithm cannot be used with the private key configured in the relevant secret ID.

   - **Keysize Mismatch** - the required key size and actual key size are not equal.

4. If the entity provider does not specify supported signing and digest methods in the standard metadata, AM falls back to the global default algorithm settings.

   Tip

   To change the global default algorithms AM uses for signing and encrypting different SAML v2.0 components, go to Configure > Global Services > Common Federation Configuration.

5. If the global default algorithms are not configured, AM examines the configured signing key type, and uses RSA-SHA256 for RSA keys, DSA-SHA256 for DSA keys, and ECDSA-SHA512 for EC keys.

> **Note**
>
> AM has different default signing algorithm settings for XML signatures, and for query signatures.
>
> AM determines the correct default query signing algorithm based on the signing key's algorithm, be it RSA, DSA, or EC. It will only fall back to the same defaults for both XML and query signing algorithms when the settings are not correctly defined.

After determining the required algorithm, the sender uses their own private key to write the signature on the request. Then, the provider receiving the message uses the public key exposed in the sender's metadata to validate the signature.

+ *How Does Encryption Work?*

In summary, when encrypting SAML v2.0 messages, the sender uses the receiver's public key (exposed in the receiver's metadata) to encrypt the request. The receiver decrypts it with its private key.

As with signing, providers also expose in their metadata the algorithms that they can use to encrypt assertion content.

Since SAML v2.0 messages are in XML format, encrypting them requires an additional key that will be transported with the message, as explained in the XML Encryption Syntax and Processing Version 1.1 specification. AM refers to those keys as *transport keys*.

Consider the following example of an encryption/decryption flow:

1. The IdP generates a random symmetric transport key using the transport key algorithm exposed in the SP's metadata.
2. The IdP encrypts the assertion with the transport key.
3. The IdP encrypts the transport key with the public key of the SP (which is also exposed in its metadata).
4. The SP decrypts the transport key using its private key. Then, it uses the transport key to decrypt the assertion.

This ensures only this SP can decrypt the message.

See the following procedures to configure signing and encryption in your environment:

- "Configuring the Advertised Signing and Encryption Algorithms"
- "Configuring AM to Sign SAML v2.0 Metadata"

- "Configuring AM to Sign and Encrypt SAML v2.0 Assertion Content"
- "Certificates and Secrets"

Configuring the Advertised Signing and Encryption Algorithms

1. Configure the required signing algorithms and digests:

   Hosted IDPs and SPs can advertise the algorithms they can use to sign assertion content. This information appears as part of the provider's metadata extension.

   + *Signing/Digest Algorithm Metadata Example*

   ```
   <Extensions>
       <alg:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
       <alg:SigningMethod Algorithm="http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha256"/>
   </Extensions>
   ```

   a. In the AM console, go to Applications > Federation > Entity Providers > *Hosted Entity Provider*.

   b. On the Assertion Content tab, in the Signing Algorithm drop-down list, select the signing algorithms this provider can use.

      There is no default for this property.

   c. On the Assertion Content tab, in the Digest Algorithm drop-down list, select the digest algorithms this provider can use.

      There is no default for this property.

2. Configure the required encryption algorithms:

   Hosted SPs and IDPs advertise their encryption algorithms so that the remote providers know which ones they should use when sending encrypted data.

   + *Encryption Algorithm Metadata Example*

   Remote providers must specify the encryption method defined by Metadata for the OASIS Security Assertion Markup Language (SAML) V2.0. For example:

   ```
   <KeyDescriptor use="encryption">
       <EncryptionMethod Algorithm="http://www.w3.org/2009/xmlenc11#rsa-oaep"/>
       <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
   </KeyDescriptor>
   ```

   a. In the AM console, go to Applications > Federation > Entity Providers > *Hosted Entity Provider*.

b. On the Assertion Content tab, in the Encryption Algorithm drop-down list, select the algorithms this provider can use.

   Select one or more AES algorithms from the list to encrypt assertion content, and one or more asymmetric algorithms to encrypt the transport key.

   + *Key Transport Algorithms*

   - http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p (default).
   - http://www.w3.org/2009/xmlenc11#rsa-oaep.

     When this algorithm is configured, AM will use the Mask Generation Function Algorithm property (Configure > Global Services > Common Federation Configuration) to create the transport key.

     For a list of supported mask generation function algorithms, see "Algorithms" in the *Reference*.
   - http://www.w3.org/2001/04/xmlenc#rsa-1_5.

     For security reasons, we strongly recommend that you do not use this option.

   If unspecified, the provider uses the following algorithms:

   - `http://www.w3.org/2001/04/xmlenc#aes128-cbc`, for assertions.
   - `http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p`, for the transport key.

Configuring AM to Sign SAML v2.0 Metadata

You can configure the `am.services.saml2.metadata.signing.RSA` secret ID with an alias that AM uses to sign exported metadata.

Perform the following steps to map the alias:

1. In the AM console, go to Configure > Secret Stores
2. Select the Keystore or HSM store where you want to map the secret.
3. On the Mappings tab, select the `am.services.saml2.metadata.signing.RSA` secret ID.
4. In the Edit Mapping pane:

   a. To **edit** a mapping, click the pen icon.

   b. To **add** a mapping, enter the alias, and then click the Add button.

c. To **delete** a mapping, click the cross icon.

5. Save your changes.

6. Export the XML-based metadata from your hosted provider to share with other providers in your circle of trust, specifying the `sign=true` query parameter:

```
$ curl \
--output metadata.xml \
"https://openam.example.com:8443/openam/saml2/jsp/exportmetadata.jsp\
?entityid=myHostedProvider\
&sign=true\
&realm=/mySubRealm"
```

If you configure your provider in the Top Level Realm, you can omit the `realm` query parameter.

The XML output contains a `<ds:Signature>` element that the remote entity can use to verify the authenticity of the metadata.

Configuring AM to Sign and Encrypt SAML v2.0 Assertion Content

Note the following important points when configuring signing and encryption of assertion content:

- **Assertions**

  HTTP-POST bindings require *signed* assertions. If the response is not signed, AM defaults to signing the assertion and uses the SP configuration to determine encryption settings.

  You must configure signing secret IDs on the identity provider. For more information, see Secret ID Mappings for SAML v2.0 Signing and Encryption in the *Security Guide*.

  Failure to configure signing when using HTTP-POST bindings might result in errors such as the following:

```
ERROR: UtilProxySAMLAuthenticatorLookup.retrieveAuthenticationFromCache: Unable to do sso or federation.
com.sun.identity.saml2.common.SAML2Exception: Provider's signing certificate alias is missing.
```

  Or:

```
ERROR: SAML2Utils.verifyResponse:Assertion is not signed or signature is not valid.
```

- **SAML authentication requests**

  Signing is **recommended** to verify the request's authenticity, and also when using the `ForceAuthn` flag.

- **SAML assertion responses**

  Signing **AND** encrypting is **recommended**, because responses can contain user data.

- **SAML logout requests**

  Signing is **recommended** to verify the request's authenticity.

Tip

Configure key rollover by mapping more than one secret to the same secret ID. See "Mapping and Rotating Secrets" in the *Security Guide*.

AM provides global default secrets for signing and encrypting SAML v2.0 assertion content, in the following secret IDs:

- `am.default.applications.federation.entity.providers.saml2.idp.encryption`
- `am.default.applications.federation.entity.providers.saml2.idp.signing`
- `am.default.applications.federation.entity.providers.saml2.sp.encryption`
- `am.default.applications.federation.entity.providers.saml2.sp.signing`

If you configure these secret IDs by realm, every provider of the same role will use the same secrets to sign and encrypt assertion content. To allow for more granularity, AM lets you override these settings and use custom secrets for each hosted provider in the realm:

1. In the AM console, go to Applications > Federation > Entity Providers > *Hosted Entity Provider*.
2. On the Assertion Content tab, in the Secret ID Identifier property, enter a string value to identify the secret IDs this provider will use. For example, `mySamlSecrets`.

   + *How Do Secret Identifiers Work?*

   AM uses the secret identifier to know which secret IDs are relevant for a provider. You can reuse the identifier that another provider is already using if you want them to share the same secrets.

   When a provider is removed from the AM configuration, AM automatically removes the secret IDs related to their identifier, unless they are being used by another provider.

   If you do not specify a value for the secret ID identifier, AM will use the global default secrets relative to the entity provider's role, *in the realm*. If they are not mapped, AM will search for the global default secrets in the global secret stores.

3. Save your changes.

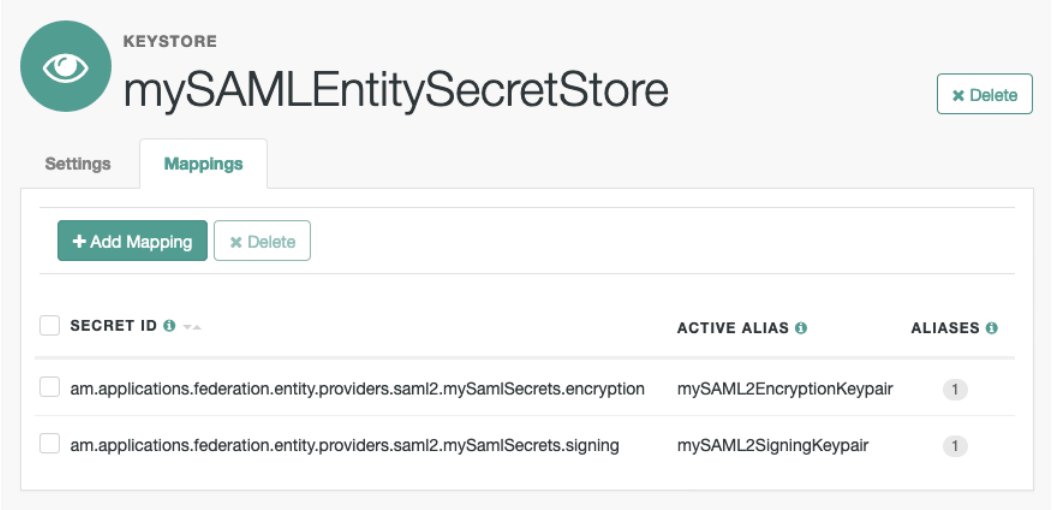
   AM creates two new secret IDs, at the realm level, based on the value you specified:

   - `am.applications.federation.entity.providers.saml2.`*`mySamlSecrets`*`.signing`
   - `am.applications.federation.entity.providers.saml2.`*`mySamlSecrets`*`.encryption`

4. (Optional) If you have not configured a secret store in the same realm as the entity provider, create one by following the steps in "Configuring Secret Stores" in the *Security Guide*.

5. Go to Realms > *Realm Name* > Secret Stores, and then select the secret store in which to map the new secret IDs.

6. On the Mappings tab, add mappings for the two custom secret IDs.



For information on creating mappings, see "Mapping and Rotating Secrets" in the *Security Guide*.

For information on the types of key pairs and secrets required, see Secret ID Mappings for SAML v2.0 Signing and Encryption in the *Security Guide*.

7. In the AM console, go to Applications > Federation > Entity Providers > *Hosted Entity Provider*.

8. On the Assertion Content tab, in the Signing and Encryption section, select the SAML v2.0 elements that AM should sign, and the elements to encrypt.

9. Save your changes.

   AM now uses the key pairs you configured in the realm's secret store to sign or encrypt the elements you selected.

Tip

For troubleshooting issues involving encryption, you can enable the `openam.saml.decryption.debug.mode` advanced property.

See "SAML v2.0 Advanced Properties".

Certificates and Secrets

SAML 2.0 secrets for hosted SP or IDP entities are managed by the secrets API, which lets you rotate certificates using secret mappings in the *Security Guide*. This only applies to *hosted* entities; certificates for remote entities are derived from SAML 2.0 metadata provided by the third party.

The following certificates are used in SAML 2.0 flows with the corresponding secret mappings.

Certificate	AM Role	Third-party Role	AM Use Case	Third-party Use Case	Secret
Hosted IDP signing certificate	Hosted IDP	Remote SP	Sign outbound SAML assertions	Validate inbound signed SAML assertion	`am.applications.federation.entity.providers.saml2.`*`secret identifier`*`.signing` [a] (`am.default.applications.federation.entity.providers.saml2.idp.signing`)
Hosted IDP encryption certificate	Hosted IDP	Remote SP	Decrypt inbound encrypted SAML requests	Encrypt outbound SAML requests	`am.applications.federation.entity.providers.saml2.`*`secret identifier`*`.encryption` [a] (`am.default.applications.federation.entity.providers.saml2.idp.encryption`)
Hosted SP signing certificate	Hosted SP	Remote IDP	Sign outbound signed SAML requests	Validate inbound signed SAML requests	`am.applications.federation.entity.providers.saml2.`*`secret identifier`*`.signing` [a] (`am.default.applications.federation.entity.providers.saml2.sp.signing`)
Hosted SP encryption certificate	Hosted SP	Remote IDP	Decrypt inbound SAML assertions	Encrypt outbound SAML assertion	`am.applications.federation.entity.providers.saml2.`*`secret identifier`*`.encryption` [a] (`am.default.applications.federation.entity.providers.saml2.sp.encryption`)

[a] If defined, this secret is used; otherwise the default (*in brackets*) is used

Chapter 5

Deploying the IdP 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 `sp.example.com` domain, the service provider has no way of knowing whether the user has possibly authenticated at `this-idp.example.com`, or at `that-idp.example.com`. The providers therefore host an identity provider discovery service in a common domain, such as `discovery.example.com`, 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.

Before you continue, ensure that you have a CoT with more than one IdP configured, and at least one SP. See "*Deployment Considerations*". You will configure the IdP discovery service in the CoT later.

Deploying the IdP discovery service involves the following stages:

1. Deploy the WAR file into your web application container.
2. Configure the discovery service.
3. Add the discovery service endpoints for writing cookies to and reading cookies from the common domain to the CoT.

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-7.1.4.war` file to the `webapps/` directory.

   ```
   $ cp ~/Downloads/openam/IDPDiscovery-7.1.4.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 `discovery.example.com:8443`, and Tomcat has unpacked the application under `/disco`, so the URL is `https://discovery.example.com:8443/disco`, which redirects to `Configurator.jsp`.

Completed Discovery Service Configuration Screen

The configuration screen shows the following fields:

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 AM 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, `example.com`.

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 AM 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.

Valid Redirects

A list of valid URLs the user can be redirected to once the IdP discovery process is complete. For example, the SPs in your CoT.

Incoming requests with URLs specified in the `RelayState` parameter that are not configured in this field are rejected.

Add each URL in a new line, for example, by pressing the enter key after each one.

Use wildcards (*) to match one or more resources in the URL.

> **Important**
>
> You must configure the same URLs in the Validation Service of each of the IdPs *in the Top Level Realm*.
>
> For more information, see "To Configure the Validation Service" in the *Authentication and Single Sign-On Guide*.

3. 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.

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, log in as AM Administrator.

2. On the service provider console, under Realms > *Realm Name* > Applications > Federation > 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 `https://discovery.example.com:8443/disco/saml2writer`, and the reader URL is `https://discovery.example.com:8443/disco/saml2reader`.

3. On each identity provider console, log in as AM Administrator.

4. On the identity provider console, under Realms > *Realm Name* > Applications > Federation > 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.

Chapter 6

Implementing SSO and SLO

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

Integrated Mode

Integrated mode single sign-on and single logout uses a SAML2 authentication node or module on a service provider (SP), thereby integrating SAML v2.0 authentication into the AM authentication process. The authentication node or module handles the SAML v2.0 protocol details for you.

Note that **integrated mode supports SP-initiated single sign-on only**, because the authentication service that includes the SAML v2.0 node or module resides on the SP. You cannot trigger IDP-initiated single sign-on in an integrated mode implementation.

Integrated mode with chains supports both IDP-initiated and SP-initiated SLO.

Integrated mode with trees does not support SLO.

Standalone Mode

Standalone mode requires that you invoke JSPs pages to initiate single sign-on and SLO. When implementing standalone mode, you do not require an AM authentication chain.

Tip

You can also configure Web and Java Agents to work alongside AM when performing SSO and SLO. See "Web or Java Agents SSO and SLO".

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

Integrated or Standalone Mode?

Deployment Task or Requirement	Implementation Mode
You want to deploy SAML v2.0 single sign-on and single logout 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.

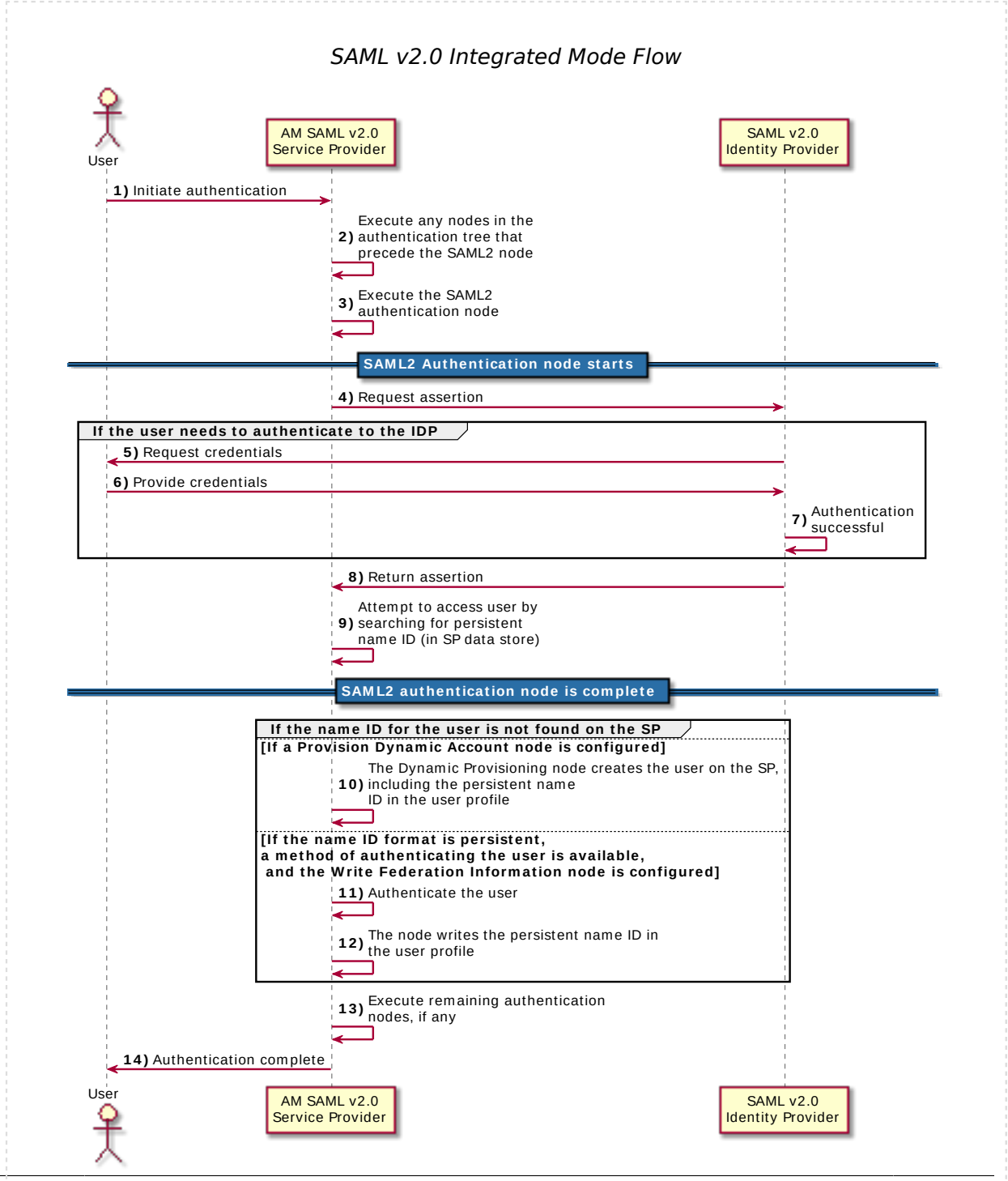
Deployment Task or Requirement	Implementation Mode
You want to use the SAML v2.0 Enhanced Client or Proxy (ECP) single sign-on profile.	Use standalone mode.
Your IDP and SP instances are using the same domain name; for example, `mydomain.net`. [a]	Use standalone mode.

[a]Due to the way integrated mode tracks authentication status by using a cookie, it cannot be used when both the IDP and SP share a domain name.

SSO and SLO in Integrated Mode

Authentication nodes and trees support SSO in integrated mode only. The "SAML2 Authentication Node" in the *Authentication and Single Sign-On Guide* handles the SAML v2.0 authentication flow, but relies on other nodes.

+ *Integrated Mode Flow (Trees)*



1. An unauthenticated user initiates authentication to an AM SAML v2.0 service provider. The login URL references an authentication tree that includes a SAML2 authentication node. For example, `https://openam.example.com:8443/openam/XUI/?service=mySAM2LTree.`

2. If there are any authentication nodes that precede the SAML2 Authentication node, AM executes them.

3. The SAML2 authentication node processing begins.

4. The authentication node requests an assertion from the IdP. The configuration of the SAML2 Authentication node determines the details of the request.

   If the user is not authenticated in the IdP, the IdP will request them to authenticate.

5. The IdP responds to the SP with a SAML assertion.

6. If the SAML assertion contains a non-transient name ID, AM searches the identity store, attempting to locate a user with the same name ID.

   If the name ID for the account exists, the tree ends in the success node.

   If the name ID does not exist...

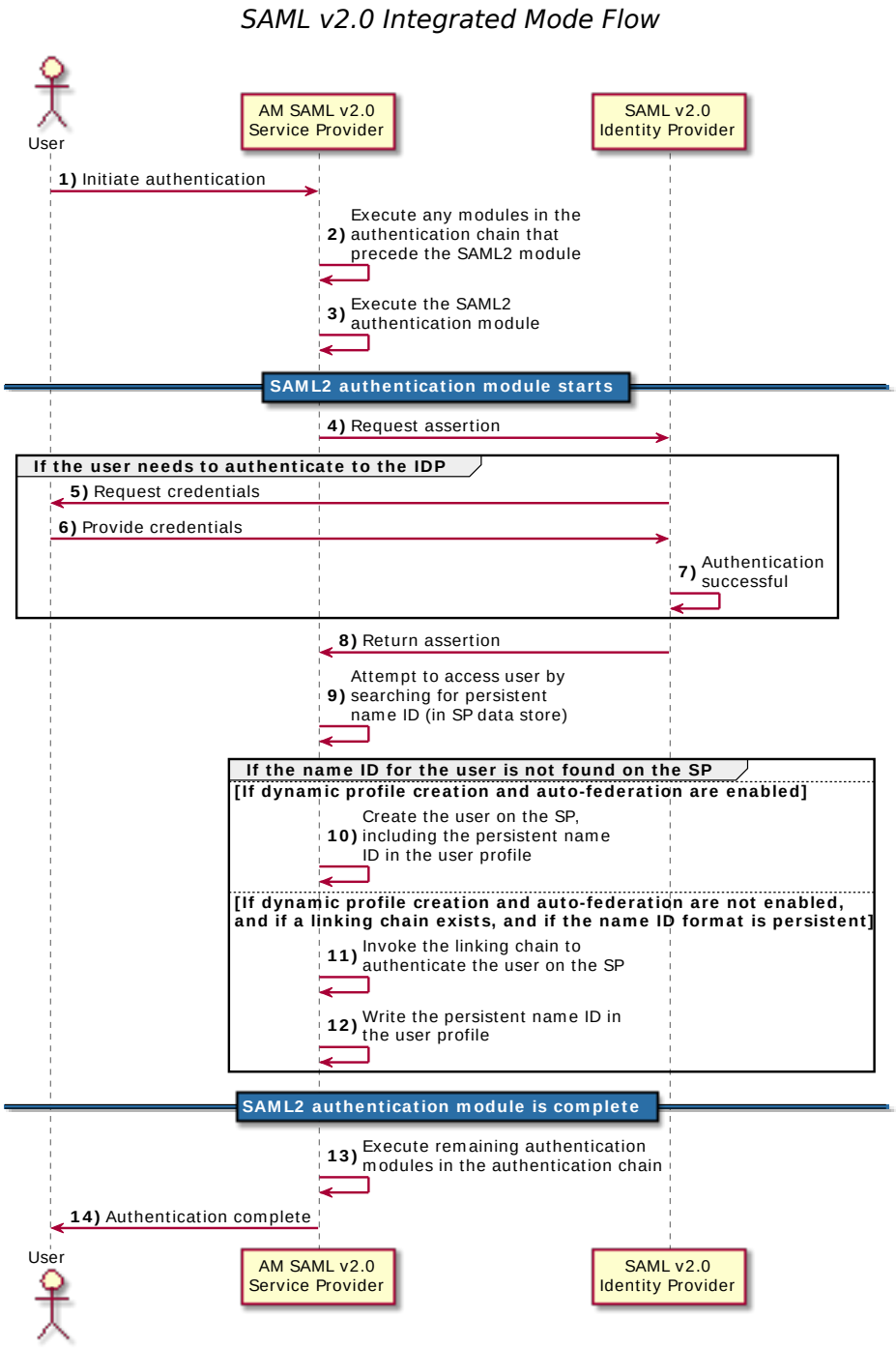
1. ... and a Create Object node(or a Provision Dynamic Account node for standalone AM environments) is configured in the tree, it creates a new account in the SP using auto-federation, including the name ID in the user profile.

2. ... and method of authenticating the user is available in the tree, a Write Federation Information node writes the persistent name ID in the user profile.

   For more information, see "Linking Identities by Using Authentication Trees or Chains".

Authentication chains support SSO and SLO in integrated mode, whereby the SAML2 authentication node handles all of the SAML v2.0 authentication flow

+ *Integrated Mode Flow (Chains)*



1. An unauthenticated user initiates authentication to an AM SAML v2.0 service provider. The login URL references an authentication chain that includes a SAML2 authentication module. For example, `https://openam.example.com:8443/openam/XUI/?service=mySAMLChain#login/`.
2. If there are any authentication modules that precede the SAML2 module in the authentication chain, AM 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 non-transient name ID, AM searches the user datastore, attempting to locate a user with the same name ID.

The flow varies here.

The following event occurs if the following are true:

- The name ID for the user is not found in the datastore
- Dynamic profile creation is configured in the Core Authentication Service
- Auto-federation is enabled on the service provider

10. AM 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 following are true:

- The name ID for the user is not found in the datastore
- A linking authentication chain has been configured in the SAML2 authentication module
- Dynamic profile creation is not configured in the Core Authentication Service
- 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, AM 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, AM executes them.

14. Authentication is complete.

Implementing SAML v2.0 Single Sign-On in Integrated Mode (Trees)

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

1. Preparing entity providers and a circle of trust, and changing several endpoints in the service provider configuration.

   See "To Configure SAML v2.0 for Integrated Mode".

2. Configuring a tree that contains, at least, the SAML2 authentication node.

   See "To Create Accounts Dynamically (Standalone AM)" and "To Create Accounts Dynamically (ForgeRock Identity Platform)".

To Configure SAML v2.0 for Integrated Mode

1. If you have not already done so, configure SAML v2.0 by performing the tasks listed in "*Deployment Considerations*".

2. In the AM console, create a hosted service provider by following the steps in "To Create a Hosted Entity Provider".

   Ensure that you have configured the attribute map (Assertion Processing > Attribute Mapper). It determines how AM will map assertion attributes coming from the IdP to the user's profile on the SP.

   During the authentication process, the mapping is used to find existing users on the SP, and to create or update user accounts on the SP.

3. Configure a remote identity provider by following the steps in "To Import and Configure a Remote Entity 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.

4. Change the Assertion Consumer Service locations in the hosted service provider configuration. The default locations support standalone mode. Therefore, you must change the locations when implementing integrated mode.

Change the locations as follows:

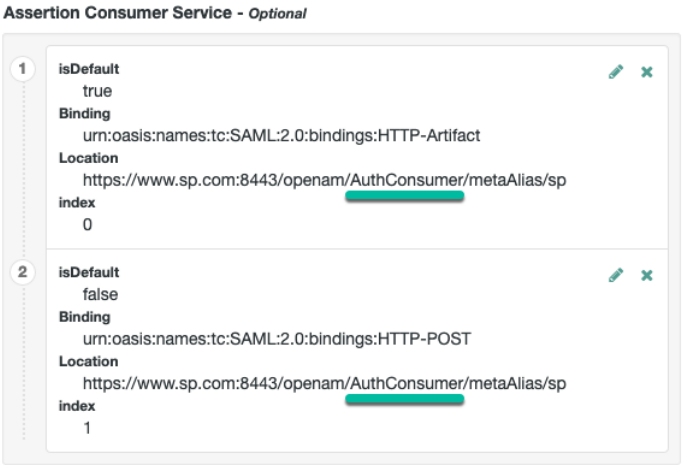
a. In the AM console, go to Realms > *Realm Name* > Applications > Federation > Entity Providers > *Service Provider Name* > Services > Assertion Consumer Service.

b. Change the location of the HTTP-Artifact consumer service to use `AuthConsumer`, rather than `Consumer`. For example, if the location is `https://www.sp.com:8443/openam/Consumer/metaAlias/sp`, change it to `https://www.sp.com:8443/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. The results will resemble the following:



   Save your changes.

Now you are ready to configure authentication trees.

To Create Accounts Dynamically (Standalone AM)

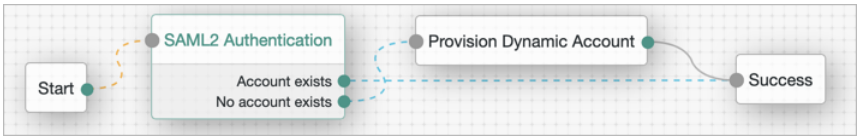
When using integrated mode, the SP can leverage the authentication tree's features to tailor the authentication experience to the users. Therefore, you can create very complicated trees, or even multiple trees to satisfy the requirements of your organization.

The example shown in this procedure uses the SAML v2.0 node to request an assertion from the IdP, and then creates an account for the user in the SP if one does not exist.

If you are not using auto-federation, you can also use authentication trees to create persistent links between the user accounts.

Perform the steps in this procedure to configure a tree similar to the following:

Example Tree With SAML2 Authentication Node



1. Add a "SAML2 Authentication Node" in the *Authentication and Single Sign-On Guide*. Remember that integrated mode is SP SSO-initiated only, and that SLO is not supported.

   The node processes the assertion, makes its contents available to the authentication tree's state in the `userInfo` object, and tries to map the assertion's nameID using the `uid` mapping in the SP's assertion map.

   If the node finds a match, the tree continues through the `Account Exists` output. Otherwise, the tree continues through the `No Account Exists` output.

2. Add a "Provision Dynamic Account Node" in the *Authentication and Single Sign-On Guide* to the No account exists outcome.

3. (Optional) If you have not configured auto-federation, you can add the "Write Federation Information Node" in the *Authentication and Single Sign-On Guide* to create a persistent link between the accounts. See "Linking Identities by Using Authentication Trees or Chains".

To Create Accounts Dynamically (ForgeRock Identity Platform)

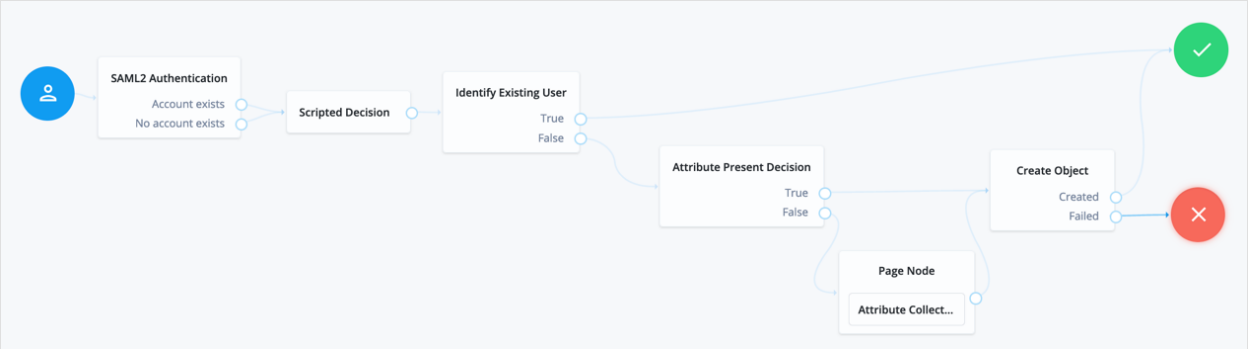
When using integrated mode, the SP can leverage the authentication tree's features to tailor the authentication experience to the users. Therefore, you can create very complicated trees, or even multiple trees to satisfy the requirements of your organization.

The example shown in this procedure uses the SAML v2.0 node to request an assertion from the IdP, and then creates an account for the user in the SP if one does not exist.

If you are not using auto-federation, you can also use authentication trees to create persistent links between the user accounts.

Perform the steps in this procedure to configure a tree similar to the following:

Example Tree to Create Accounts Dynamically



1. Add a "SAML2 Authentication Node" in the *Authentication and Single Sign-On Guide*. Remember that integrated mode is SP SSO-initiated only, and that SLO is not supported.

   The node processes the assertion, makes its contents available to the authentication tree's state in the `userInfo` object, and tries to map the assertion's nameID using the `uid` mapping in the SP's assertion map.

   If the node finds a match, the tree continues through the `Account Exists` output. Otherwise, the tree continues through the `No Account Exists` output.

   Note that the attribute the node uses to map the nameID is not configurable, and this example adds nodes to process the `userInfo` object and match its contents to the managed user's schema.

2. Add a "Scripted Decision Node" in the *Authentication and Single Sign-On Guide* to copy the information from the assertion to the authentication tree's shared state.

   + *Example Script*

```
outcome = "true";
if (sharedState.get("userInfo")) {
    if (sharedState.get("objectAttributes")) {
      sharedState.remove("objectAttributes");
    }
    var userName=null,sn=null,mail=null;

    try { userName=sharedState.get("userInfo").get("attributes").get("uid").get(0).toString(); }
 catch (e) {}
    try { sn=sharedState.get("userInfo").get("attributes").get("sn").get(0).toString(); } catch
 (e) {}
    try { mail=sharedState.get("userInfo").get("attributes").get("mail").get(0).toString(); }
 catch (e) {}

    sharedState.put("objectAttributes", {"userName":userName,"sn":sn,"mail":mail});
}
```

For more information, see "Scripted Decision Node API Functionality" in the *Authentication and Single Sign-On Guide*.

3. Add a "Identify Existing User Node" in the *Authentication and Single Sign-On Guide* to search the user with the appropriate attribute. For example, `userName`.

4. Complete the tree adding the required nodes to create the new account if it does not exist on the SP.

   The scripted decision node that you created before gathered the attributes that are now available to create the account. However, these may not be enough to satisfy your managed user rules. To ensure that the required attributes are available, use the "Required Attributes Present Node" in the *Authentication and Single Sign-On Guide* to check them, and the "Attribute Collector Node" in the *Authentication and Single Sign-On Guide* to collect the ones missing.

   Finally, to create the account, use the "Create Object Node" in the *Authentication and Single Sign-On Guide*.

   Ensure that you configure the appropriate identity resource in this node. For example, `managed/alpha_user`.

5. (Optional) If you have not configured auto-federation, you can add the "Write Federation Information Node" in the *Authentication and Single Sign-On Guide* to create a persistent link between the accounts. See "Linking Identities by Using Authentication Trees or Chains".

Implementing SSO in Integrated Mode (Chains)

The following list is an overview of the activities you perform when implementing SAML v2.0 single sign-on 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 including 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.

To Implement SAML v2.0 Single Sign-on in Integrated Mode

1. If you have not already done so, configure SAML v2.0 by performing the tasks listed in "*Deployment Considerations*".

2. In the AM console of the SP, create a hosted service provider by following the steps in "To Create a Hosted Entity Provider".

3. Configure a remote identity provider by following the steps in "To Import and Configure a Remote Entity 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.

4. Change the Assertion Consumer Service locations in the hosted service provider configuration. The default locations support standalone mode. Therefore, you must change the locations when implementing integrated mode.

   Change the locations as follows:

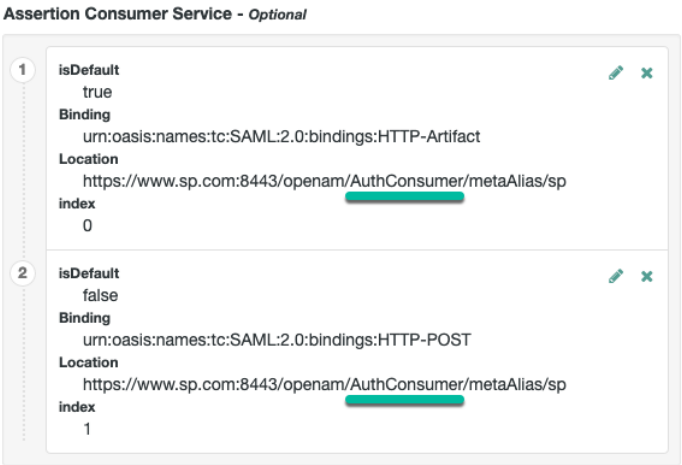
   a. In the AM console, go to Realms > *Realm Name* > Applications > Federation > Entity Providers > *Service Provider Name* > Services > Assertion Consumer Service.

   b. Change the location of the HTTP-Artifact consumer service to use `AuthConsumer`, rather than `Consumer`. For example, if the location is `https://www.sp.com:8443/openam/Consumer/metaAlias/sp`, change it to `https://www.sp.com:8443/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. The results will resemble the following:



   Save your changes.

5. Create a SAML2 authentication module:

   a. In the AM console, go to Realms > *Realm Name* > Authentication > Modules, and then select Add Module.

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

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

      Ensure that you set a value for the Linking Authentication Chain property, specifying the authentication chain that will be invoked.

   d. Save your changes.

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

7. Test your configuration. First, clear your browser's cache and cookies. Then, attempt to log in to AM using a login URL that references the authentication chain that includes the SAML2 module. For example, `https://www.sp.com:8443/openam/XUI/#login/&service=mySAMLChain`.

   AM will redirect you to the identity provider for authentication.

   If required, you can now configure AM to link identities in the IDP with those in the SP, generate new accounts on the SP, or link to a shared account; for example `anonymous`. For more information and instructions, see "*Federating Identities*".

Configuring SLO in Integrated Mode (Chains)

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:

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`.

Requirement	Configuration
	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`.

SSO and SLO in Standalone Mode

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

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.

The module must be of type `Federation`, and also have the *name* as `Federation`.

AM 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, create a new 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.

JSP Pages for SSO and SLO

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

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

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

> **Note**
>
> The JSP pages only support query parameters sent by using HTTP GET requests. Do not attempt to use HTTP POST or PUT requests to the pages.

IDP-Initiated SSO JSP

`idpSSOInit.jsp`

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

Also mapped to the endpoint `idpssoinit` under the context root.

URLs:

- `https://www.idp.com:8443/openam/saml2/jsp/idpSSOInit.jsp`
- `https://www.idp.com:8443/openam/idpssoinit`

Example:

- The following URL initiates single sign-on from the identity provider side, leaving the user at `http://forgerock.com`:

```
https://www.idp.com:8443/openam/saml2/jsp/idpSSOInit.jsp
?metaAlias=/idp
&spEntityID=https%3A%2F%2Fwww.sp.com%3A8443%2Fopenam
&RelayState=http%3A%2F%2Fforgerock.com
```

+ *idpSSOInit.jsp Query 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. 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=https://www.sp.com:8443/openam` as `spEntityID=https%3A%2F%2Fwww.sp.com%3A8443%2Fopenam`.

`affiliationID`

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

`binding`

(Optional) Use this parameter to indicate which binding to use for the operation.

For example, specify `binding=HTTP-POST` to use HTTP POST binding with a self-submitting form. You can also specify `binding=HTTP-Artifact`.

`NameIDFormat`

(Optional) Use this parameter to specify a SAML Name Identifier format identifier.

For example, `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 `RelayState`.

For example, if the query string `target=http%3A%2F%2Fforgerock.com&RelayStateAlias=target`, is equivalent to `RelayState=http%3A%2F%2Fforgerock.com`.

IDP-Initiated SLO JSP

`idpSingleLogoutInit.jsp`

Used to initiate single logout from the IDP.

Also mapped to the endpoint `IDPSloInit` under the context root.

URLs:

- `https://www.idp.com:8443/openam/saml2/jsp/idpSingleLogoutInit.jsp`
- `https://www.idp.com:8443/openam/IDPSloInit`

Example:

- The following URL performs single logout from the identity provider side, using a self-submitting form rather than a redirect, and leaving the user at `https://forgerock.com`:

```
https://www.idp.com:8443/openam/saml2/jsp/idpSingleLogoutInit.jsp
?binding=urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST
&RelayState=https%3A%2F%2Fforgerock.com
```

+ *idpSingleLogoutInit.jsp Query Parameters*

`binding`

(Required) Use this parameter to indicate which 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`
- `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`.

SP-Initiated SSO JSP

`spSSOInit.jsp`

Use this page to initiate single sign-on from the service provider side.

Also mapped to the endpoint `spssoinit` under the context root.

URLs:

- `https://www.sp.com:8443/openam/saml2/jsp/spSSOInit.jsp`
- `https://www.sp.com:8443/openam/spssoinit`

Example:

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

```
https://www.sp.com:8443/openam/saml2/jsp/spSSOInit.jsp
?metaAlias=/sp
&idpEntityID=https%3A%2F%2Fwww.idp.com%3A8443%2Fopenam
&RelayState=https%3A%2F%2Fwww.sp.com%3A8443%2Fopenam%2FXUI%2F%23profile%2Fdetails
```

+ *spSSOInit.jsp Query Parameters*

`idpEntityID`

(Required) Use this parameter to indicate the remote identity provider. Make sure you URL-encode the value.

For example, encode `idpEntityID=https://www.idp.com:8443/openam` as: `idpEntityID=https%3A%2F%2Fwww.idp.com%3A8443%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. Do not repeat the slash for the Top Level Realm, for example `metaAlias=/sp`.

`affiliationID`

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

`AllowCreate`

(Optional) When set to `true`, the identity provider can create a new identifier for the principal if none exists.

`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.

AM accepts the following values:

- **better**. Specifies that the authentication context statement in the assertion must be better (stronger) than one of the provided authentication contexts.
- **exact**. Specifies that the authentication context statement in the assertion must exactly match at least one of the provided authentication contexts.
- **maximum**. Specifies that the authentication context statement in the assertion must not be stronger than any of the other provided authentication contexts.
- **minimum**. Specifies that the authentication context statement in the assertion must be at least as strong as one of the provided authentication contexts.

`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 AM should use to authenticate the user.

`binding`

(Optional) Use this parameter to indicate which binding to use for the operation.

For example, specify `binding=HTTP-POST` to use HTTP POST binding with a self-submitting form. You can also specify `binding=HTTP-Artifact`.

`Destination`

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

`ForceAuthn`

(Optional) When set to `true` the identity provider should force authentication.

> **Tip**
>
> Configure the `org.forgerock.openam.saml2.authenticatorlookup.skewAllowance` advanced property to specify the maximum permissible time since authentication by the IDP. See "SAML v2.0 Advanced Properties".

When false, the IDP can reuse existing security contexts.

`isPassive`

(Optional) When set to `true` the identity provider authenticates passively.

`NameIDFormat`

(Optional) Use this parameter to specify a SAML Name Identifier format identifier.

For example, `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, the query string `target=http%3A%2F%2Fforgerock.com&RelayStateAlias=target`, is the same as `RelayState=http%3A%2F%2Fforgerock.com`.

`reqBinding`

(Optional) Use this parameter to indicate the 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>
```

SP-Initiated SLO JSP

`spSingleLogoutInit.jsp`

Used to initiate single logout from the SP.

Also mapped to the endpoint `SPSloInit` under the context root.

URLs:

- `https://www.sp.com:8443/openam/saml2/jsp/spSingleLogoutInit.jsp`
- `https://www.sp.com:8443/openam/SPSloInit`

Example:

- The following URL initiates single logout from the service provider side, using the HTTP redirect method, leaving the user at `http://forgerock.com`:

```
https://www.sp.com:8443/openam/saml2/jsp/spSingleLogoutInit.jsp
?binding=urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect
&RelayState=http%3A%2F%2Fforgerock.com
```

+ *spSingleLogoutInit.jsp Query Parameters*

`binding`

(Required) Use this parameter to indicate which 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. You can also specify `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` property is not set, then AM uses this parameter to find the default binding. Make sure you URL-encode the value.

For example, specify `idpEntityID=https://www.idp.com:8443/openam` as `idpEntityID=https%3A%2F%2Fwww.idp.com%3A8443%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.

The `RelayState` parameter 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, AM uses the first entity ID in the metadata.

Modifying the SSO JSP Page to Indicate Progress

During single sign-on login in standalone mode, AM presents users with a self-submitting form when access has been validated. This page is otherwise blank.

Perform the steps in the following procedure if you want to present users with something to indicate that the operation is in progress:

To Indicate Progress During SSO in Standalone Mode

1. To modify the templates to add feedback that single sign-on is in progress, such as an image, edit the source of the AM Java Server Page, `saml2/jsp/autosubmitaccessrights.jsp`, under the file system directory where the AM WAR file 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 the AM-7.1.4.war file.

3. Overwrite the modified `saml2/jsp/autosubmitaccessrights.jsp` file, where you unpacked the `.war` file. Also, include any images referenced in your files.

4. Pack up your custom version of AM, and then deploy it in your web container.

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 single sign-on 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.

AM 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 AM 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 AM 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 > Federation > Entity Providers > *IDP Name* > IDP > Advanced > ECP Configuration.

By default, an AM service provider uses the `com.sun.identity.saml2.plugins.ECPIDPFinder` class to return identity providers from the list under Realms > *Realm Name* > Applications > Federation > 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 AM service provider is `/SPECP` as in `https://www.sp.com:8443/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 URL-encoded.

For example, to access the service provider followed by the resource at `http://forgerock.org/index.html`, use `https://www.sp.com:8443/openam/SPECP?metaAlias=/sp&RelayState=http%3A%2F%2Fforgerock.org%2Findex.html`.

Web or Java Agents SSO and SLO

You can use Web Agents and Java Agents in a SAML v2.0 Federation deployment.

Configuring agents to work alongside AM when performing SAML v2.0 single sign-on and single logout involves altering the URLs the agents use for logging in unauthenticated users, and logging users out.

To Use Web or Java Agents with a SAML v2.0 Service Provider

This procedure applies when AM is configured as an IDP in one domain, and a Web or Java agent protects resources on behalf of a second AM server, configured as an SP, on a second domain.

1. Install the web or Java agent, as described in the relevant user documentation.

   The following steps will guide you to configure the agent through the AM console. If your agent is not using the centralized configuration mode, make the changes to the noted properties in the `OpenSSOAgentConfiguration.properties` configuration file of the agent instead.

2. When using *Web* agents:

   a. In the AM console of the SP, go to Realms > *Realm Name* > Applications > Agents > Web > *Agent Name* > AM Services.

   b. When using integrated mode SSO:

      - Set the AM Login URL List property (`com.sun.identity.agents.config.login.url`) to the authentication chain that contains the "SAML2 Authentication Module" in the *Authentication and Single Sign-On Guide*, or the authentication tree that contains the "SAML2 Authentication Node" in the *Authentication and Single Sign-On Guide*. For example:

        ```
        https://www.sp.com:8443/openam/XUI/#login/&realm=alphas&ervice=mySAMLTree
        ```

   c. When using standalone mode SSO:

      i. Set the AM Login URL List property (`com.sun.identity.agents.config.login.url`) to the URL of the SP-initiated SSO JSP file, including the parameters necessary for initiating SSO. For example:

         ```
         https://www.sp.com:8443/openam/saml2/jsp/spSSOInit.jsp
         ?metaAlias=/sp
         &idpEntityID=https%3A%2F%2Fwww.idp.com%3A8443%2Fopenam
         ```

      ii. Add the URL of the SP-initiated SLO JSP file to the AM Logout URL property (`com.sun.identity.agents.config.logout.url`). For example:

         ```
         https://www.sp.com:8443/openam/saml2/jsp/spSingleLogoutInit.jsp
         ?binding=urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect
         &RelayState=http%3A%2F%2Fwww.sp.com
         ```

   d. Save your changes.

3. Set the Enable Custom Login Mode (`org.forgerock.openam.agents.config.allow.custom.login`) property to `1`.

4. Disable the Invalidate Logout Session property (`org.forgerock.agents.config.logout.session.invalidate` set to `false`).

5. When using *Java* agents:

   a. In the AM console of the SP, go to Realms > *Realm Name* > Applications > Agents > Java > *Agent Name* > AM Services.

   b. When using integrated mode SSO:

      - Set the AM Login URL List property (`com.sun.identity.agents.config.login.url`) to the authentication chain that contains the "SAML2 Authentication Module" in the *Authentication and Single Sign-On Guide*, or the authentication tree that contains the "SAML2 Authentication Node" in the *Authentication and Single Sign-On Guide*. For example:

        ```
        https://www.sp.com:8443/openam/XUI/#login/&service=mySAMLTree
        ```

   c. When using standalone mode SSO:

      i. Set the AM Login URL List property (`com.sun.identity.agents.config.login.url`) to the URL of the SP-initiated SSO JSP file, including the parameters necessary for initiating SSO. For example:

         ```
         https://www.sp.com:8443/openam/saml2/jsp/spSSOInit.jsp
           ?metaAlias=/sp
           &idpEntityID=https%3A%2F%2Fwww.idp.com%3A8443%2Fopenam
         ```

      ii. Add the URL of the SP-initiated SLO JSP file to the AM Logout URL property (`com.sun.identity.agents.config.logout.url`). For example:

         ```
         https://www.sp.com:8443/openam/saml2/jsp/spSingleLogoutInit.jsp
         ?binding=urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect
         &RelayState=http%3A%2F%2Fwww.sp.com
         ```

   d. Enable the Enable Custom Login Mode property (set the `org.forgerock.openam.agents.config.allow.custom.login` to `true`).

   e. Enable the Convert SSO Tokens Into OIDC JWTs property (set the `org.forgerock.agents.accept.ipdp.cookie` to `true`).

   f. Save your changes.

Chapter 7

Federating Identities

AM supports linking, or *federating*, identities between the IDP and the SP.

See the following table for a list of tasks to configure how AM federates identities:

Task	Resources
Decide Whether to Permanently Link Identities AM lets you choose whether to maintain the link between federated entities after logout (persistent federation) or to create a new link each time the user logs in (transient federation). Also, learn how to manage persistent federation.	• "Persistent or Transient Federation"
Link Identities Automatically Configure AM to link identities automatically when they exist in both the IDP and the SP, or to create an account on the SP when the `NameID` that the IDP provides unequivocally identifies the identity.	• "Linking Identities Automatically with Auto-Federation" • "Creating Identities Automatically with Auto-Federation"
Link Identities Using Trees or Chains Configure AM to link identities when the `NameID` that the IDP provides is not enough to unequivocally identify the identity.	• "Linking Identities by Using Authentication Trees or Chains"
Link Identities in the IDP to a Single, Shared Account on the SP Configure AM to temporarily link an identity in the IDP to, for example, the `anonymous` user in the SP.	• "Linking Identities to a Single, Shared Account"

Persistent or Transient Federation

You can choose to permanently link identities, known as *persistent federation*. AM lets you configure the service provider to persistently link identities, 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 another unique user identifier, you can use this method to link accounts without user interaction. See "To Link Identities Automatically Based on an Attribute Value".

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. This is known as *transient federation*.

Transient federation 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.

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

Authentication *is* required on the service provider side, however, when the service provider is unable to map the identity in the assertion from the identity provider to a local user account.

This can happen the first time accounts are linked, for example. After which, the persistent identifier establishes the mapping.

Authenticating to the SP may also be required when transient federation is used when linking identities. The service provider must authenticate the user for every SAML assertion received. This is because the identifier used to link the identities is transient; it does not provide a repeatable, durable means to link the identities.

Note

You can prevent the ability to persistently link accounts on the SP side by setting the `spDoNotWriteFederationInfo` property to `true`, and on the IDP side by setting the `idpDisableNameIDPersistence` to `true`.

To Enable Persistent Federation

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

Before attempting to configure persistent federation, ensure that you have configured AM for SAML v2.0 single sign-on, created the identity and service providers, and configured a circle of trust. For information on performing those tasks, see "*Deployment Considerations*" and "*Implementing SSO and SLO*".

1. If you are using integrated mode SSO with chains:

   a. Navigate to Realms > *Realm Name* > Authentication > Modules, and then select the module name of your SAML2 authentication module.

   b. In the NameID Format field, specify the value `urn:oasis:names:tc:SAML:2.0:nameid-format:persistent`.

   Note

   You can also link accounts together using different nameid formats. For example, you could use the `urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified` value, and receive the IDP user's e-mail address in the NameID value. The SP would display the login page to identify the local user account and persistently link them.

   c. Save your work.

d. Initiate single sign-on by accessing a URL that calls a SAML v2.0 authentication service:

   For example, `https://www.sp.com:8443/openam/XUI/#login/&service=mySAMLChain`.

2. If you are using integrated mode SSO with trees:

   a. Create an authentication tree that contains a SAML v2.0 authentication node.

      If you have not created a journey yet, see "SSO and SLO in Integrated Mode" for an example.

   b. In the NameID Format field, specify the value `urn:oasis:names:tc:SAML:2.0:nameid-format:persistent`.

      > **Note**
      >
      > You can also link accounts together using different nameid formats. For example, you could use the `urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified` value, and receive the IDP user's e-mail address in the NameID value. The SP would display the login page to identify the local user account and persistently link them.

   c. Save your work.

   d. Initiate single sign-on by accessing a URL that calls the journey you just modified.

      For example, `https://www.sp.com:8443/openam/XUI/#login/&service=mySAMLTree`.

3. If you are using standalone mode single sign-on:

   - Initiate single sign-on with either the `spSSOInit.jsp` or `idpSSOInit.jsp` JSP page, including `NameIDFormat=urn:oasis:names:tc:SAML:2.0:nameid-format:persistent` as a query parameter.

     For example, to initiate single sign-on from the service provider, access a URL similar to the following:

     ```
     https://www.sp.com:8443/openam/saml2/jsp/spSSOInit.jsp
     ?idpEntityID=https%3A%2F%2Fwww.idp.com%3A8443%2Fopenam
     &metaAlias=/sp
     &NameIDFormat=urn:oasis:names:tc:SAML:2.0:nameid-format:persistent
     ```

     To initiate single sign-on from AM acting as the identity provider, access a URL similar to the following:

     ```
     https://www.idp.com:8443/openam/saml2/jsp/idpSSOInit.jsp
     ?spEntityID=https%3A%2F%2Fwww.sp.com%3A8443%2Fopenam
     &metaAlias=/idp
     &NameIDFormat=urn:oasis:names:tc:SAML:2.0:nameid-format:persistent
     ```

4. To test your work:

   a. Authenticate to the IDP as the user you want to persistently link.

On success, you will be redirected to the SP.

Tip

If there was no login page displayed at the SP, you might have enabled auto-federation, or AM was able to find a link between the two identities without requiring authentication at the SP.

To ensure there are no existing links, create a new identity in the IDP, and initiate single sign-on again, authenticating to the IDP as the new user.

b. Authenticate to the SP as the local user to link with.

The accounts are persistently linked, with persistent identifiers stored in the user's profile on both the IDP and the SP.

Subsequent attempts to access the SP will only require that the user authenticates to the IDP, as the identities are now permanently linked.

Note

You can prevent the ability to persistently link accounts on the SP side by setting the `spDoNotWriteFederationInfo` property to `true`, and on the IDP side by setting the `idpDisableNameIDPersistence` to `true`.

To Enable Transient Federation

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

Before attempting to configure transient federation, ensure that you have configured AM for SAML v2.0, created the identity and service providers, and configured a circle of trust. You must also have configured AM to support single sign-on. For information on performing those tasks, see "*Deployment Considerations*" and "*Implementing SSO and SLO*".

1. If you are using integrated mode SSO with chains:

   a. Navigate to Realms > *Realm Name* > Authentication > Modules, and then select the module name of your SAML2 authentication module.

   b. In the NameID Format field, specify the value `urn:oasis:names:tc:SAML:2.0:nameid-format:transient`.

   c. Save your work.

   d. Initiate single sign-on by accessing a URL that calls an authentication chain that includes the SAML2 module.

For example, `https://www.sp.com:8443/openam/XUI/#login/&service=mySAMLChain`.

2. If you are using integrated mode SSO with trees:

   a. Create an authentication tree that contains a SAML v2.0 authentication node.

      If you have not created a journey yet, see "SSO and SLO in Integrated Mode" for an example.

   b. In the NameID Format field, specify the value `urn:oasis:names:tc:SAML:2.0:nameid-format:transient`.

   c. Save your work.

   d. Initiate single sign-on by accessing a URL that calls the journey you just modified.

      For example, `https://www.sp.com:8443/openam/XUI/#login/&service=mySAMLChain`.

3. If you are using standalone mode SSO:

   - Initiate single sign-on with either the `spSSOInit.jsp` or `idpSSOInit.jsp` JSP page, including `NameIDFormat=urn:oasis:names:tc:SAML:2.0:nameid-format:transient` as a query parameter.

     For example, to initiate single sign-on from the service provider, access a URL similar to the following:

     ```
     https://www.sp.com:8443/openam/saml2/jsp/spSSOInit.jsp
     ?idpEntityID=https%3A%2F%2Fwww.idp.com%3A8443%2Fopenam
     &metaAlias=/sp
     &NameIDFormat=urn:oasis:names:tc:SAML:2.0:nameid-format:transient
     ```

     To initiate single sign-on from the identity provider, access a URL similar to the following:

     ```
     https://www.idp.com:8443/openam/saml2/jsp/idpSSOInit.jsp
     ?spEntityID=https%3A%2F%2Fwww.sp.com%3A8443%2Fopenam
     &metaAlias=/idp
     &NameIDFormat=urn:oasis:names:tc:SAML:2.0:nameid-format:transient
     ```

4. To test your work:

   a. Authenticate to the IDP as the user you want to temporarily link.

      On success, you will be redirected to the SP.

   b. Authenticate to the SP as the local user.

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

      Nothing is written in the user's profile on either the identity provider and the service provider.

Subsequent attempts to access the SP will require that the user authenticates to the IDP *AND* the SP (assuming no existing session exists), as the identities are not linked.

Managing Federation of Persistently Linked Accounts

AM 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]

AM provides a pair of JSP files for managing persistently linked accounts; `idpMNIRequestInit.jsp` for initiating changes from the IDP side, and `spMNIRequestInit.jsp` for initiating changes from the SP side.

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=https://www.sp.com:8443/openam` as `spEntityID=https%3A%2F%2Fwww.sp.com%3A8443%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.

[1] To configure these attribute types, in the AM console, go to Configure > Global Services, and then select SAMLv2 Service Configuration.

`binding`

(Optional) Use this parameter to indicate which 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=https://www.idp.com:8443/openam` as `idpEntityID=https%3A%2F%2Fwww.idp.com%3A8443%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 which 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`.

To Change Federation of Persistently Linked Accounts

1. Retrieve the name identifier value, used to manage the federation in the second step.

   - You can retrieve the name identifier value on the IDP side by checking the value of the `sun-fm-saml2-nameid-infokey` property.

     For example, if the user's entry in the directory shows:

     ```
     sun-fm-saml2-nameid-infokey:
       https://www.idp.com:8443/openam|
       https://www.sp.com:8443/openam|
       XyfFEsr6Vixbnt0BSqIglLFMGjR2
     ```

   - Then, the name identifier on the IDP side is `XyfFEsr6Vixbnt0BSqIglLFMGjR2`.

     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 the following:

     ```
     sun-fm-saml2-nameid-info: https://www.sp.com:8443/openam|
       https://www.idp.com:8443/openam|
       ATo9TSA9Y2Ln7DDrAdO3HFfH5jKD|
       https://www.idp.com:8443/openam|
       urn:oasis:names:tc:SAML:2.0:nameid-format:persistent|
       9B1OPy3m0ejv3fZYhlqxXmiGD24c|
       https://www.sp.com:8443/openam|
       SPRole|
       false
     ```

     Then, the name identifier on the SP side is `9B1OPy3m0ejv3fZYhlqxXmiGD24c`.

2. Use the identifier to initiate a change request, as in the following examples:

   - To initiate a change request from the service provider, use a URL similar to the following example:

```
https://www.sp.com:8443/openam/saml2/jsp/spMNIRequestInit.jsp
?idpEntityID=https%3A%2F%2Fwww.idp.com%3A8443%2Fopenam
&metaAlias=/sp
&requestType=NewID
&IDPProvidedID=XyfFEsr6Vixbnt0BSqIglLFMGjR2
```

You can substitute `openam/SPMniInit` for `openam/saml2/jsp/spMNIRequestInit.jsp`.

- To initiate a change request from the identity provider, use a URL similar to the following example:

```
https://www.idp.com:8443/openam/saml2/jsp/idpMNIRequestInit.jsp
?spEntityID=https%3A%2F%2Fwww.sp.com%3A8443%2Fopenam
&metaAlias=/idp
&requestType=NewID
&SPProvidedID =9B1OPy3m0ejv3fZYhlqxXmiGD24c
```

You can substitute `openam/IDPMniInit` for `openam/saml2/jsp/idpMNIRequestInit.jsp`

To Terminate Federation of Persistently Linked Accounts

AM lets you terminate account federation, where the accounts have been linked with a persistent identifier, as described in "To Enable Persistent Federation".

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 AM on port 8443 under deployment URI `/openam`.

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

```
https://www.sp.com:8443/openam/saml2/jsp/spMNIRequestInit.jsp
?idpEntityID=https%3A%2F%2Fwww.idp.com%3A8443%2Fopenam
&metaAlias=/sp
&requestType=Terminate
```

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

```
https://www.idp.com:8443/openam/saml2/jsp/idpMNIRequestInit.jsp
?spEntityID=https%3A%2F%2Fwww.sp.com%3A8443%2Fopenam
&metaAlias=/idp
&requestType=Terminate
```

Linking Identities Automatically with Auto-Federation

AM lets you configure the service provider to automatically link identities based on an attribute value in the assertion returned from the identity provider, known as *auto-federation*.

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 configure

AM to map the attributes to each other, and link identities, without the user having to authenticate to the SP.

To Link Identities Automatically Based on an Attribute Value

This procedure demonstrates how to automatically link identities based on an attribute value that is the same in both accounts.

Before attempting to configure auto-federation, ensure that you have configured AM for SAML v2.0, created the identity and service providers, and configured a circle of trust. You must also have configured AM to support single sign-on. For information on performing those tasks, see "*Deployment Considerations*" and "*Implementing SSO and SLO*".

Perform the following steps on the hosted IDP(s), and again on the hosted SP(s):

1. Go to Realms > *Realm Name* > Applications > Federation > Entity Providers, and click on the name of the hosted provider.

   + *How Do I Switch Between SP and IDP Configuration for a Given Provider?*

   AM only displays the configuration of a single role. Click on the labels to select the role view:

2. On the hosted IDP:

   a. Go to the Assertion Processing tab.

   b. Review the Attribute Map configuration. If the attributes you want to use to link the accounts on the IDP and the SP are not in the map already, add them.

      The IDP will send these attributes in the assertion, and the SP will then map them using its own attribute map.

      + *Tips to Configure the Attribute Map on the IDP*

      The user profile attributes used here must both be allowed in user profiles, and also be specified for the identity repository. See "Adding User Profile Attributes" in the *Setup Guide*, for instructions on allowing additional attributes in user profiles.

      To see the profile attributes available for an LDAP identity repository, log in to the AM console, and go to Realms > *Realm Name* > Identity Stores > User Configuration. Check the LDAP User Attributes list.

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 example:

c. Save your work.

3. On the hosted SP:

   a. Go to the Assertion Processing tab.

   b. Review the Attribute Map configuration, and ensure that the attribute mappings you created on the IDP are represented in the map.

   + *Tips to Configure the Attribute Map on the SP*

   The value of Key is a SAML attribute sent in an assertion, and the value of Value is a property in the user's session, or an attribute of the user's profile.

   By default, the SP maps the SAML attributes it receives to equivalent-named session properties. However, when the SP is configured to create identities during autofederation and the identity does not exist yet, the SP maps the SAML attributes to their equivalents in the newly-created user profile.

   The special mapping `Key: *, Value: *` means that the SP maps each attribute it receives in the assertion to equivalent-named properties or attributes. For example, if the SP receives `mail` and `firstname` in the assertion, it maps them to `mail` and `firstname` respectively.

   Remove the special mapping and add key pairs to the map if:

   - (During autofederation) The attributes in the IdP's and the SP's identity stores do not match.
   - You need control over the names of the session properties.
   - You need control over which attributes the SP should map, because the IdP adds too many to the assertion.

For example, if the the SAML attribute is `firstname` and you want the SP to map it to a session property/user profile attribute called `cn`, create a mapping similar to `Key: firstname, Value: cn`.

c. Enable Auto Federation. In the Attribute property, enter the SAML attribute name that the SP will use to link accounts, as configured in the Attribute Map.

d. Save your work.

4. To test your work, initiate single sign-on; for example, as described in "IDP-Initiated SSO JSP".

   Authenticate to the IDP as the `demo` user. Attempt to access the SP, and you will notice that the user has a session, and can access their profile page on the SP without having to authenticate again.

Creating Identities Automatically with Auto-Federation

On occasion, there may not yet be an identity to link with on the SP. For example, if it is the first time a user is attempting to access the service, and they do not have an account in the SP identity store.

You can configure AM to dynamically create an account for the user in the SP identity store, using the values in the assertion as profile properties, as defined in the attribute mappings.

To Create and Link Identities Based on Attribute Values

Before attempting to configure auto-federation to create identities based on attribute values, ensure that you have configured AM for SAML v2.0, created the identity and service providers, and configured a circle of trust. You must also have configured AM to support single sign-on. For information on performing those tasks, see "*Deployment Considerations*" and "*Implementing SSO and SLO*".

The following steps demonstrate how to dynamically create missing accounts on the SP:

1. Go to Realms > *Realm Name* > Applications > Federation > Entity Providers, and click on the name of the hosted provider.

   + *How Do I Switch Between SP and IDP Configuration for a Given Provider?*

   AM only displays the configuration of a single role. Click on the labels to select the role view:

2. On the hosted IDP:

   a. Go to the Assertion Processing tab.

   b. Review the Attribute Map configuration. If the attributes you want to populate when creating the new user are not in the map already, add them.

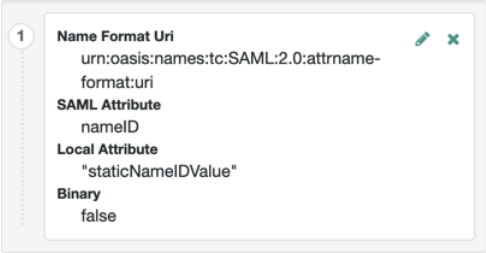
      The IDP will send these attributes in the assertion, and the SP will then map them using its own attribute map.

      + *Tips to Configure the Attribute Map on the IDP*

      The user profile attributes used here must both be allowed in user profiles, and also be specified for the identity repository. See "Adding User Profile Attributes" in the *Setup Guide*, for instructions on allowing additional attributes in user profiles.

      To see the profile attributes available for an LDAP identity repository, log in to the AM console, and go to Realms > *Realm Name* > Identity Stores > User Configuration. Check the LDAP User Attributes list.

      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 example:

   c. Save your work.

3. On the hosted SP:

   a. Go to the Assertion Processing tab.

b. Review the Attribute Map configuration, and ensure that the attribute mappings on the IDP are represented in the map.

+ *Tips to Configure the Attribute Map on the SP*

The value of Key is a SAML attribute sent in an assertion, and the value of Value is a property in the user's session, or an attribute of the user's profile.

By default, the SP maps the SAML attributes it receives to equivalent-named session properties. However, when the SP is configured to create identities during autofederation and the identity does not exist yet, the SP maps the SAML attributes to their equivalents in the newly-created user profile.

The special mapping `Key: *, Value: *` means that the SP maps each attribute it receives in the assertion to equivalent-named properties or attributes. For example, if the SP receives `mail` and `firstname` in the assertion, it maps them to `mail` and `firstname` respectively.

Remove the special mapping and add key pairs to the map if:

- (During autofederation) The attributes in the IdP's and the SP's identity stores do not match.
- You need control over the names of the session properties.
- You need control over which attributes the SP should map, because the IdP adds too many to the assertion.

For example, if the the SAML attribute is `firstname` and you want the SP to map it to a session property/user profile attribute called `cn`, create a mapping similar to `Key: firstname, Value: cn`.

c. Enable Auto Federation. In the Attribute property, enter the SAML attribute name that the SP will use to link accounts, as configured in the Attribute Map.

Tip

The value of the named attribute is used as the username of the created user when auto-federation is enabled.

d. Save your work.

e. Navigate to Realms > *Realm Name* > Authentication > Settings.

f. On the User Profile tab, in the User Profile field, select Dynamic or Dynamic with User Alias.

For more information the user profile property, see User Profile in the *Authentication and Single Sign-On Guide*.

g. Save your work.

4. To test your work:

   a. Create a new user on the identity provider, including values for any attributes you mapped in the providers.

   b. Log out of the AM console, and initiate SSO; for example, as described in "IDP-Initiated SSO JSP".

   c. Authenticate as the new user you created in the IDP.

   d. On success, check `https://www.sp.com:8443/openam/XUI/#profile/details` to see the new user account created on the SP, and the attributes that were copied from the assertion.

Linking Identities by Using Authentication Trees or Chains

Identity providers and service providers must be able to communicate about users. Yet, in some cases, the identity provider chooses to communicate a minimum of information about an authenticated user; for example, a generated, opaque `NameID` that cannot directly be used to locate to an identity in the SP identity store.

AM can use these pseudonym identifiers for establishing links between otherwise unrelated accounts, by requiring that the user authenticates to the SP using a linking authentication tree or chain.

The following list describes the sequence of events that occurs the first time a user attempts to authenticate to the AM service provider:

1. **Accessing the service provider.**

   A user attempts to access a service and is redirected to the AM server acting as the service provider, specifying the SAML v2.0 service in the login URL. For example:

   - An authentication chain containing the "SAML2 Authentication Module" in the *Authentication and Single Sign-On Guide*.

     For example, `https://www.sp.com:8443/openam/XUI/#login/&service=spSAMLChain`.

   - An authentication tree containing the "SAML2 Authentication Node" in the *Authentication and Single Sign-On Guide*.

     For example, `https://www.sp.com:8443/openam/XUI/#login/&service=spSAMLTree`.

2. **Authentication at the identity provider.**

AM redirects the user to the identity provider. The user authenticates successfully at the identity provider. The identity provider returns a SAML assertion to the SP.

3. **Service provider attempts to access a federated identity.**

   AM attempts to locate the identity in its own user store. No link between the IDP identity and a local one is found.

4. **Invocation of the linking chain, or equivalent authentication node(s).**

   As no link is found, AM does one of the following:

   - (Authentication chains) Invokes a linking authentication chain to determine which local user account to use.
   - (Authentication trees) Goes through a path in the authentication tree that lets the user authenticate to the SP.

5. **Identity federation.**

   After successful authentication at the SP, AM then writes the name ID from the assertion into the local user's profile, creating a permanent link between the two identities.

   For more information on creating permanent links between identities, see "Persistent or Transient Federation".

   For an example of an authentication tree that links identities, see "SSO and SLO in Integrated Mode".

The following list describes the sequence of events that occurs during subsequent authentication attempts, after the user's identities on the identity and service providers have been federated:

1. **Accessing the service provider.**

   A returning user attempts to access their service and is redirected to the AM server acting as the service provider. Their login URL specifies the SAML v2.0 login service. For example:

   - An authentication chain containing the "SAML2 Authentication Module" in the *Authentication and Single Sign-On Guide*.

     For example, `https://www.sp.com:8443/openam/XUI/#login/&service=spSAMLChain`.

   - An authentication tree containing the "SAML2 Authentication Node" in the *Authentication and Single Sign-On Guide* and the "Write Federation Information Node" in the *Authentication and Single Sign-On Guide*.

     For example, `https://www.sp.com:8443/openam/XUI/#login/&service=spSAMLTree`.

2. **Authentication at the identity provider.**

AM redirects the user to the identity provider, and the user authenticates successfully at the identity provider. The identity provider returns a SAML assertion to the SP.

3. **Service provider attempts to access a federated identity.**

   AM attempts to locate the name ID in its user store. The search for the name ID succeeds.

   As there is a match, the user does not need to log in to the SP, and is given access to the service.

Configure Authentication Trees or Chains to Link Accounts

To configure AM to persistently link accounts, see:

- "To Link Identities by Using a Linking Authentication Chain"
- "To Link Accounts using Authentication Trees (Standalone AM)"
- "To Link Accounts Persistently (ForgeRock Identity Platform)"

To Link Identities by Using a Linking Authentication Chain

This procedure demonstrates how to link identities by using a linking authentication chain on the SP to identify the local user.

Before attempting to configure a linking authentication chain, ensure that you have configured AM for SAML v2.0, created the identity and service providers, and configured a circle of trust. You must also have configured AM to support single sign-on. For information on performing those tasks, see "*Deployment Considerations*" and "*Implementing SSO and SLO*".

1. On the hosted service provider, log in to the AM console.

2. Create an authentication chain; for example, named `myLinkingChain`. This chain is used to identify the user account in the SP to link to the account in the IDP. The chain can use any of the available methods for authentication as required; for example, multi-factor authentication.

   For more information on creating authentication chains, see "*Configuring AM for Authentication*" in the *Authentication and Single Sign-On Guide*.

3. If you are using integrated mode SSO:

   a. Navigate to Realms > *Realm Name* > Authentication > Modules, and then select the module name of your SAML2 authentication module.

   b. In the Linking Authentication Chain field, enter the name of the authentication chain you created earlier; for example, `myLinkingChain`.

   c. Save your work.

4. If you are using standalone mode SSO:

a. Navigate to Realms > *Realm Name* > Authentication > Settings > Core.

b. In the Organization Authentication Configuration property, select the authentication chain you created earlier; for example, `myLinkingChain`.

c. Save your work.

For more information on setting the default chain for a realm, see Configure Sensible Default Authentication Services in the *Security Guide*.

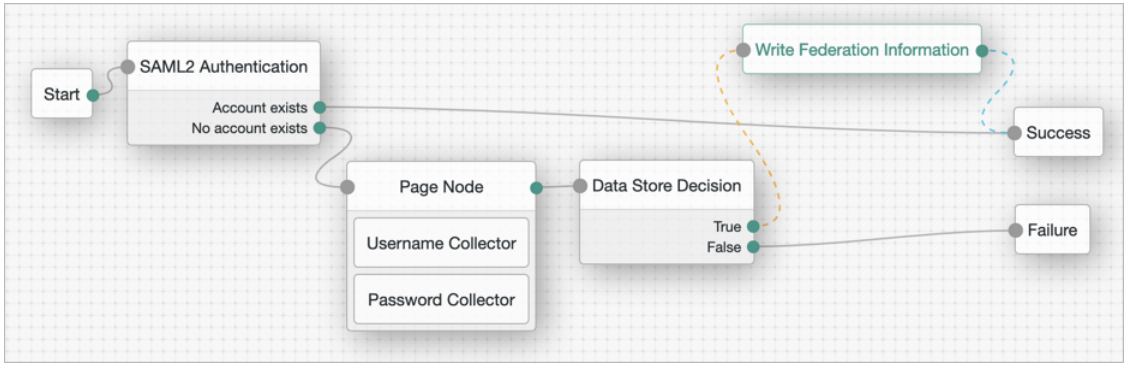
5. To test your work, initiate single sign-on; for example, as described in "SP-Initiated SSO JSP".

   Authenticate to the IDP as the `demo` user. You will then be redirected to the SP and asked to authenticate using the linking authentication chain specified. If persistent linking is enabled (the default), then initiating single sign-on a second time will require authentication only to the IDP.

To Link Accounts using Authentication Trees (Standalone AM)

If you are not using auto-federation, perform the steps in this procedure to configure a tree similar to the following to link accounts persistently:

Example Tree to Link Accounts Persistently



1. Add a "SAML2 Authentication Node" in the *Authentication and Single Sign-On Guide*. Remember that integrated mode is SP SSO-initiated only, and that SLO is not supported.

   Ensure that the NameID Format specified is `persistent.`

   The node processes the assertion, makes its contents available to the authentication tree's state in the `userInfo` object, and tries to map the assertion's nameID using the `uid` mapping in the SP's assertion map.

   If the node finds a match, the tree continues through the `Account Exists` output. Otherwise, the tree continues through the `No Account Exists` output.

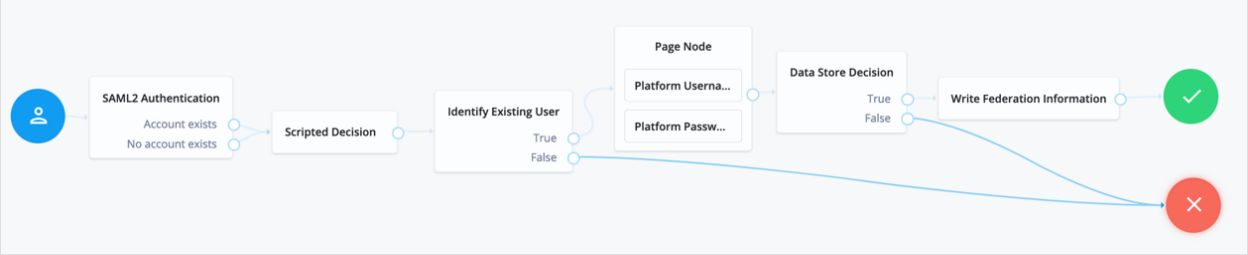
2. On the `No Account Exists` outcome, configure nodes to authenticate the user to the SP.

3. Add a "Write Federation Information Node" in the *Authentication and Single Sign-On Guide*.

To Link Accounts Persistently (ForgeRock Identity Platform)

If you are not using auto-federation, perform the steps in this procedure to configure a tree similar to the following to link accounts persistently:

Example Tree to Link Accounts Persistently



1. Add a "SAML2 Authentication Node" in the *Authentication and Single Sign-On Guide*. Remember that integrated mode is SP SSO-initiated only, and that SLO is not supported.

   Ensure that the NameID Format specified is `persistent.`

   The node processes the assertion, makes its contents available to the authentication tree's state in the `userInfo` object, and tries to map the assertion's nameID using the `uid` mapping in the SP's assertion map.

   If the node finds a match, the tree continues through the `Account Exists` output. Otherwise, the tree continues through the `No Account Exists` output.

   Note that the attribute the node uses to map the nameID is not configurable, and this example adds nodes to process the `userInfo` object and match its contents to the managed user's schema instead.

2. Add a "Scripted Decision Node" in the *Authentication and Single Sign-On Guide* to copy the information from the assertion to the authentication tree's shared state.

   + *Example Script*

```
outcome = "true";
if (sharedState.get("userInfo")) {
    if (sharedState.get("objectAttributes")) {
      sharedState.remove("objectAttributes");
    }
    var userName=null,sn=null,mail=null;

    try { userName=sharedState.get("userInfo").get("attributes").get("uid").get(0).toString(); }
 catch (e) {}
    try { sn=sharedState.get("userInfo").get("attributes").get("sn").get(0).toString(); } catch
 (e) {}
    try { mail=sharedState.get("userInfo").get("attributes").get("mail").get(0).toString(); }
 catch (e) {}

  sharedState.put("objectAttributes", {"userName":userName,"sn":sn,"mail":mail});
}
```

For more information, see "Scripted Decision Node API Functionality" in the *Authentication and Single Sign-On Guide*.

3. Add a "Identify Existing User Node" in the *Authentication and Single Sign-On Guide* to search the user with the appropriate attribute. For example, `userName`.

4. Authenticate the user to the SP.

5. Add the "Write Federation Information Node" in the *Authentication and Single Sign-On Guide* to the successful outcome of the authentication process to create the link between the accounts.

   If a transient link exists, it will be converted into a persistent one.

Linking Identities to a Single, Shared Account

AM lets you map identities on the identity provider temporarily to a single account on the service provider; for example, the `anonymous` account, 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 create or retain identity data on the service provider, but instead you make authorization decisions based on attribute values from the identity provider.

To Link Identities to a Single Service Provider Account

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

Before attempting these steps, ensure that you have configured AM for SAML v2.0, created the identity and service providers, and configured a circle of trust. You must also have configured AM to

support single sign-on. For information on performing those tasks, see "*Deployment Considerations*" and "*Implementing SSO and SLO*".

1. On the hosted identity provider:

   a. In the AM console, go to Realms > *Realm Name* > Applications > Federation > Entity Providers > *Hosted Identity Provider Name*.

   b. On the Assertion Processing tab, if the attributes you want to access from the SP are not yet included in the Attribute Map property, add the attribute mappings.

      Enter attribute map values using the following format: `SAML Attribute Name=Profile Attribute Name`.

   c. Save your work.

2. On the hosted service provider:

   a. In the AM console, go to Realms > *Realm Name* > Applications > Federation > Entity Providers > *Hosted Service Provider Name*.

   b. On the Assertion Processing tab, if the attributes you want to access from the IDP are not yet included in the Attribute Map property, add the attribute mappings.

      Enter attribute map values using the following format: `SAML Attribute Name=Profile Attribute Name`.

      Tip

      You can use a special wildcard mapping of `*=*`, which maps each attribute in the assertion to an identically named attribute on the SP, using the relevant value.

   c. Ensure that the Auto Federation property is not selected.

   d. In the Transient User property, add the account name AM will use to link all identities from the IDP, for example; `anonymous`.

   e. Save your work.

3. To test your work:

   a. Create a new user on the identity provider, including values for any attributes you mapped in the providers.

   b. Log out of the AM console, and initiate SSO using transient federation; for example, as described in "To Enable Transient Federation".

   c. Authenticate to the IDP as the new user you created.

d. After successfully authenticating to the IDP, check that the identity is linked to a transient account by performing the following steps:

   i. In a separate browser or private window, log in to the AM console of the SP.

   ii. Go to Realms > *Realm Name* > Sessions.

   iii. Enter the transient user name you configured earlier; for example, `anonymous`.

   You will see one or more sessions of users who have initiated single sign-on and been temporarily linked to the transient user account.

Linking Identities in Bulk

If you manage both the identity provider and service provider, you can link accounts in bulk 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 "*Configuring IDPs, SPs, and CoTs*", and install the **ssoadm** tool. See "*Setting Up Administration Tools*" in the *Installation Guide*.

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

The **ssoadm** command requires 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=sp,dc=com|id=demo,ou=user,dc=idp,dc=com
```

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=sp,dc=com|id=demo,ou=user,dc=idp,dc=com

$ ssoadm do-bulk-federation \
  --metaalias /sp \
  --remoteentityid https://www.idp.com:8443/openam \
  --useridmapping /tmp/user-map.txt \
  --nameidmapping /tmp/name-map.txt \
  --adminid uid=amAdmin,ou=People,dc=openam,dc=forgerock,dc=org \
  --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@www.idp.com:/tmp/name-map.txt

openam@www.idp.com'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 uid=amAdmin,ou=People,dc=openam,dc=forgerock,dc=org \
  --password-file /tmp/pwd.txt \
  --metaalias /idp \
  --bulk-data-file /tmp/name-map.txt

Bulk Federation for this host was completed.
```

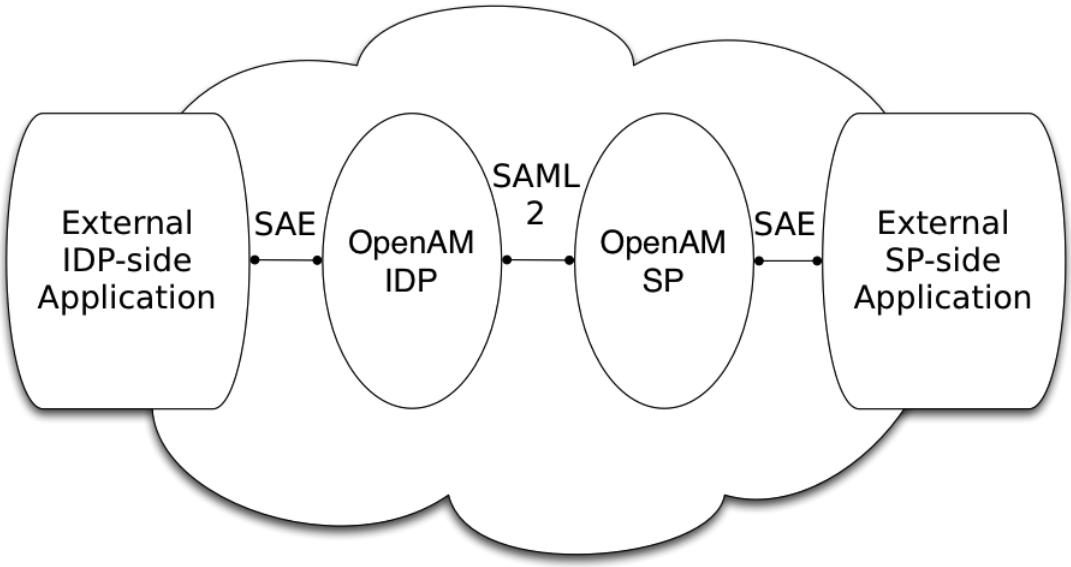
At this point the accounts are linked.

Chapter 8

Implementing a SAML v2.0 Gateway by Using Secure Attribute Exchange

Most deployments can rely on AM to handle authentication and provide identity assertions. AM supports a wide variety of authentication scenarios out of the box, but AM also makes it possible to add custom authentication modules. Furthermore, IG lets you integrate legacy systems into your access management deployment.

In a deployment where you need AM to act as a SAML v2.0 gateway to a legacy application that serves as an identity provider, you can use AM Secure Attribute Exchange (SAE). On the identity provider side, SAE lets AM retrieve the information needed to create assertions from an external authentication service, bypassing AM authentication and trusting the external service as the authoritative source of authentication. On the service provider side, SAE lets AM securely provide attributes to an application that makes its own policy decision based on the attributes rather than rely on AM 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 AM to create

a session by sending a secure HTTP GET or POST to AM that asserts the identity of the user. AM 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 AM to assert both the user's identity and also any necessary attributes related to the user. AM 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 AM server by sending the `sun.cmd=logout` attribute to AM using SAE.

On the service provider side, AM communicates using SAML v2.0 with AM on the identity provider side. AM 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 AM and external applications.

AM ships with sample JSPs that demonstrate secure attribute exchange. To try the sample, you must set up an AM 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.

Installing the SAE Samples

Set up an AM 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 "*Configuring IDPs, SPs, and CoTs*". This chapter assumes you set up the hosted identity provider at `https://www.idp.com:8443/openam` and the hosted service provider at `https://www.sp.com:8443/openam`. Make sure federation is working before you add secure attribute exchange applications that rely on functioning SAML v2.0 communications between the providers.

The SAE samples are not delivered with AM. Instead, you find them with the client SDK samples in `openam-examples/openam-example-clientsdk-war` folder of the `openam-public` repo. The SAE samples are under `/saml2/sae` where you install the samples. `saeIDPApp.jsp` is the identity provider side external application. `saeSPApp.jsp` is the service provider side external application.

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 AM server acting as identity provider, and sets up a trust relationship between the application on the service provider side and the AM 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 `https://www.idp.com:8443/openam/encode.jsp` and `https://www.sp.com:8443/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 AM 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 AM's public key. Also note the path to the keystore for your application, the keystore password, and the private key password.

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 AM 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 > Federation > Entity Providers, select 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 `https://www.idp.com:8443/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=https://www.idp.com:8443/samples/saml2/sae/saeIDPApp.jsp|type=symmetric|secret=`*`encoded-secret`*, where the AM 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 > Federation > Entity Providers, select 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 `https://www.sp.com:8443/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 `https://www.sp.com:8443/samples/saml2/sae/saeSPApp.jsp`, and then save your work.

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 AM 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 > Federation > Entity Providers, select 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 `https://www.sp.com:8443/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 `https://www.sp.com:8443/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=https://www.sp.com:8443/samples/saml2/sae/saeSPApp.jsp|type=symmetric|secret=`*`encoded-secret`*, where the AM 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.

Trying It Out

After completing the setup described above, go to the IDP-side SAE application, for example at `https://www.idp.com:8443/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

`https://www.sp.com:8443/samples/saml2/sae/saeSPApp.jsp`

SAE URL on IDP end

`https://www.idp.com:8443/openam/idpsaehandler/metaAlias/idp`

Check the settings, and then select Generate URL to open the Secure Attributes Exchange IDP APP SAMPLE page.

Select 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 https://www.idp.com:8443/openam
sun.spentityid  https://www.sp.com:8443/openam
branch          mainbranch
sun.authlevel   0
```

Chapter 9

Implementing SAML v2.0 Service Providers by Using Fedlets

An AM *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 AM.

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.

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

+ *Fedlet Support for SAML v2.0 Features*

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
Multiple IDPs	Supported
External IDP Discovery Service	Supported
Bundled IDP Reader Service for Discovery	Supported

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

The following table summarizes the high-level tasks required to configure Fedlets:

Task	Resources
Create and Configure the Fedlet Configure the Fedlet files and its keystore for your environment, add the metadata from the IDPs to it, and share the Fedlet's metadata with the IDPs.	• "Creating and Configuring the Fedlet"
Ensure the Fedlet is Secure By default, signing and encryption are not configured. You should configure them to sign and encrypt data, such as assertions.	• "Enabling Signing and Encryption in a Fedlet"
Test the Fedlet You can test the Fedlet as a standalone application, or by integrating it inside one of your applications.	• "Deploying and Testing the Fedlet on the SP" • "Integrating with the Fedlet WAR File"

Creating and Configuring the Fedlet

An AM 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 full AM distribution file, `AM-7.1.4.zip`, includes the Java Fedlet package, `Fedlet-7.1.4.zip`, that you can use as the basis of your Fedlet. This section covers how to configure a Java Fedlet using that distribution, by editing the circle of trust, Java properties, and IDP and SP XML configuration templates.

The high-level steps are:

- Determine the roles that the IDP(s) and Fedlet play in SAML v2.0 Circles of Trust.
- Unpack the Fedlet from the full AM distribution ZIP file to access the Fedlet WAR file and template configuration files.
- Prepare 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.
- Finish preparing the Fedlet configuration by editing the remaining Fedlet template configuration files.
- Share the Fedlet SAML v2.0 configuration files with the IDP(s).

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

Contents of the Java Fedlet Distribution ZIP File

Unpack the Java Fedlet distribution ZIP file into a working directory:

```
$ mkdir fedlet && cd fedlet
$ unzip ../Fedlet-7.1.4.zip
```

The `Fedlet-7.1.4.zip` file contains the following files:

`fedlet.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 describes 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 the `/home/user` folder, that user could have a `/home/user/fedlet` folder for Fedlet configuration files:

```
$ mkdir ~/fedlet
```

Tip

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 "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, and should be named `fedlet.cot` after configuration.

For more about this file, see "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 an `idp.xml` file here, alongside the other SAML v2.0 configuration files. How you obtain this file from the IDP depends on the IDP implementation.

To obtain this information from an AM instance, see "To Create a Hosted Entity Provider".

`conf/idp-extended.xml-template`

This template holds extended SAML v2.0 IDP settings that AM uses, and should be named `idp-extended.xml` after configuration.

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

`conf/sp.xml-template`

This template describes standard SAML v2.0 SP settings, and should be named `sp.xml` after configuration.

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

`conf/sp-extended.xml-template`

This template describes extended SAML v2.0 SP settings that the Fedlet uses, and should be named `sp-extended.xml` after configuration.

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

To configure a Fedlet, make copies of the template files listed above, configure the necessary properties and values, and provide the resulting files to the person administering the SP, ready to deploy. See "Deploying and Testing the Fedlet on the SP".

Configuring Java Fedlet Properties

File: `FederationConfig.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 `www.sp.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:

1. `off`
2. `error`
3. `warning`

4. `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 "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 "Changing Default Key Aliases" in the *Security Guide*.

Example: `@FEDLET_HOME@/keystore.jceks`

`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 AM.

- In the AM console, go to Deployment > Servers > *Server Name* > Security > Encryption, and set the Password Encryption Key to your symmetric key.

  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 `https://openam.example.com:8443/openam/encode.jsp`, enter the cleartext password to encode with your symmetric key, and select 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` (`JCEKS` is recommended.)

`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.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`

Despite the name, this provider supports JCEKS keystores.

`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@`

Configuring Circles of Trust

File: `fedlet.cot`

This file 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:

- "To Configure a Circle of Trust With a Single IDP"
- "To Configure a Circle of Trust With Multiple IDPs"
- "To Configure Multiple Circles of Trust"

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 a `fedlet.cot` file 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 AM 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
   ```

To Configure a Circle of Trust With Multiple IDPs

When the circle of trust involves multiple IDPs, use the Fedlet in combination with the AM 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 AM IDP Discovery service.

   For details see "*Deploying the IdP Discovery Service*".

2. Configure the circle of trust as described in "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.

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 "Configuring the Identity Providers".

3. For each circle of trust, set up the cot file as described in "To Configure a Circle of Trust With a Single IDP".

4. In the extended SP XML file described in "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.

Configuring the Identity Providers

Files: `idp.xml`, `idp-extended.xml`

As described in "Contents of the Java Fedlet Distribution ZIP File", the IDP provides its standard SAML v2.0 metadata as XML, which you save in the configuration folder as a `idp.xml` file. If the IDP uses AM, the IDP can also provide extended SAML v2.0 metadata as XML, which you save in the configuration folder as a `idp-extended.xml` file, 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.

Identity Provider Standard XML

This section covers the configuration in the `idp.xml` file. 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 AM 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 AM deployment URL.

If the IDP uses an implementation different from AM, 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.

Identity Provider Extended XML

This section covers the configuration in the `idp-extended.xml` file. Most extended metadata are specific to the AM implementation of SAML v2.0. If the IDP runs AM, 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 the `entity-config-schema.xsd` file, available as part of the AM source code, though not included in the AM 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 the `fedlet.cot` file, 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 single sign-on 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 AM. The basic example in the IDP template shows the minimal configuration for the single sign-on 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>
</IDPSSOConfig>
<AttributeAuthorityConfig>
   <Attribute name="cotlist">
     <Value>fedletcot</Value>
   </Attribute>
</AttributeAuthorityConfig>
<XACMLPDPConfig>
   <Attribute name="wantXACMLAuthzDecisionQuerySigned">
     <Value></Value>
   </Attribute>
   <Attribute name="cotlist">
     <Value>fedletcot</Value>
   </Attribute>
</XACMLPDPConfig>
```

When functioning as IDP, AM can take many other `Attribute` values. These are implementation dependent. You can obtain the extended metadata from AM 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>
```

Identity Provider Extended XML: IDPSSOConfig Settings

This section covers elements for the IDP single sign-on role, arranged in the order they appear in the template.

`description`

Description of the file.

`cotlist`

Specifies the circle of trust(s) to which the provider belongs.

Default: `fedletcot`

Configuring the Service Providers

Files: `sp.xml`, `sp-extended.xml`

As mentioned in "Contents of the Java Fedlet Distribution ZIP File", the Fedlet SAML v2.0 configuration is defined in two XML files, the standard metadata in a `sp.xml` file and the extended metadata in a `sp-extended.xml` file.

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.

Service Provider Standard XML

This section covers the configuration in the `sp.xml` file. 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 "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 file, `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.

  AM 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.

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 "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.

Service Provider Extended XML

This section covers the configuration in the `sp-extended.xml` file. The extended metadata are specific to the AM 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 the `entity-config-schema.xsd` file, available as part of the AM 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 the `fedlet.cot` file.

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 AM, 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.

Service Provider Extended XML: SPSSOConfig Settings

This section covers elements for the SP single sign-on role, arranged in the order they appear in the template.

`description`

Human-readable description of the Fedlet in the SP single sign-on 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 "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 "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 AM 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 AM based on the value of a profile attribute that is common to user profiles both in AM 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 is an empty value.

`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 "Configuring Circles of Trust".

Default: `fedletcot`

`saeAppSecretList`

When using Secure Attribute Exchange with AM 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 AM 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 AM 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

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 "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 "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 "Configuring Circles of Trust".

Default: `fedletcot`

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 "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 "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 AM 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 "Configuring Circles of Trust".

Default: `fedletcot`

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 AM and the Fedlet to sign and to verify XML signatures, and to encrypt and to decrypt data such as SAML assertions.

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 > Federation > Entity Providers > *IDP Name* > Signing and Encryption in the AM console.

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

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

- On the Fedlet side, set up a JCEKS keystore used for signing and encryption. For evaluation, you can use copy the `keystore.jceks` file delivered with AM. You can find the file in the `$HOME/openam/security/keystores/` directory for a server instance with the base URI `openam`. The built-in keystore includes the `test` certificate.

  You must also set up the `.storepass` and `.keypass` files using the `fedletEncode.jsp` page, such as `https://openam.example.com:8443/fedlet/fedletEncode.jsp`, to encode passwords on the Fedlet side.

  The passwords for the test keystore and private key are recorded in the AM `.storepass` and `.keypass` files. These files are located in the `/path/to/openam/security/secrets/defaults/` directory.

- 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.

To Configure the Fedlet For Signing and Encryption

The `FederationConfig.properties` file specifies the paths to the 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.jceks -alias test
Enter keystore password:
-----BEGIN CERTIFICATE-----
MIIDaDCCAlCgAwIBAgIDcB/YMA0GCSqGSIb3DQEBCwUAMGUxCzAJBgNVBAYTAlVL
MRAwDgYDVQQIEwdCcmlzdG9sMRAwDgYDVQQHEwdCcmlzdG9sMRIwEAYDVQQKEwlG
b3JnZVJvY2sxDzANBgNVBAsTBk9wZW5BTTENMAsGA1UEAxMEdGVzdDAeFw0xNjAz
MTgxMTU2MjhaFw0yNjAzMTYxMTU2MjhaMGUxCzAJBgNVBAYTAlVLMRAwDgYDVQQI
EwdCcmlzdG9sMRAwDgYDVQQHEwdCcmlzdG9sMRIwEAYDVQQKEwlGb3JnZVJvY2sx
DzANBgNVBAsTBk9wZW5BTTENMAsGA1UEAxMEdGVzdDCCASIwDQYJKoZIhvcNAQEB
BQADggEPADCCAQoCggEBAKNbl89eP6B8kZATNSPe3+OZ3esLx31hjX+dakHtPwXC
AaCKqJFwjwKdxyRuPdsVG+8Dbk3PGhk26aJrSE93EpxeqmQqxNPMeD+N0/8pjkuV
YWwPIQ/ts2iTiWOVn7wzlE4ASfvupqOR5pjuYMWNo/pd4L7QNjUCKoAt9H11HMyi
P+6roo/EYgX4AH7OAhfUMncYsopWhkW/ze9z8wTXc8BAEgDmt8zFCez1CtqJB/Ml
SBUGDgk8oHYDsHKmx05baBaOBQ8LRGP5SULSbRtu34eLFootBIn0FvUZSnwTiSpb
aHHRgWrMOVm07oSLWBuO3h/bj38zBuuqqVsAK8YuyoECAwEAAaMhMB8wHQYDVR0O
BBYEFHxfAbr6PQ5Xgc+jVx+AGTPnnpWZMA0GCSqGSIb3DQEBCwUAA4IBAQAZBMJ2
9/2idv1ztC6ArHtB4kw/nHHwthXFwtWAN7sRPB8tLW7fD8aJ43RQr5107Bg1Lgkm
t+FZxpafqUC/mukjIzGzbW0COMSOTcWUGss+HxK6M6Fl9aOzKJMct1uOSpPFgjIt
cGqydGZXR2FH93vXWoAotUwtZ119IixIdxpOJwYJg0HFn+GEfpU1PmiLfq2/uwqJ
0hGCNfNcm9puagzhQrcDFOnolxjnYPSfSkU5wxlGo99yE5eJwoHXXU7csaZVttmx
7sPj1lUENogXUM6JMqzSyEIm1XCOCL8rZJkZ781W5CwZhuJTNzV31sBREs8FaaCe
ksu7Y48BmkUqw6E9
-----END CERTIFICATE-----
```

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>
      MIIDaDCCAlCgAwIBAgIDcB/YMA0GCSqGSIb3DQEBCwUAMGUxCzAJBgNVBAYTAlVL
      MRAwDgYDVQQIEwdCcmlzdG9sMRAwDgYDVQQHEwdCcmlzdG9sMRIwEAYDVQQKEwlG
      b3JnZVJvY2sxDzANBgNVBAsTBk9wZW5BTTENMAsGA1UEAxMEdGVzdDAeFw0xNjAz
      MTgxMTU2MjhaFw0yNjAzMTYxMTU2MjhaMGUxCzAJBgNVBAYTAlVLMRAwDgYDVQQI
      EwdCcmlzdG9sMRAwDgYDVQQHEwdCcmlzdG9sMRIwEAYDVQQKEwlGb3JnZVJvY2sx
      DzANBgNVBAsTBk9wZW5BTTENMAsGA1UEAxMEdGVzdDCCASIwDQYJKoZIhvcNAQEB
      BQADggEPADCCAQoCggEBAKNbl89eP6B8kZATNSPe3+OZ3esLx31hjX+dakHtPwXC
      AaCKqJFwjwKdxyRuPdsVG+8Dbk3PGhk26aJrSE93EpxeqmQqxNPMeD+N0/8pjkuV
      YWwPIQ/ts2iTiWOVn7wzlE4ASfvupqOR5pjuYMWNo/pd4L7QNjUCKoAt9H11HMyi
      P+6roo/EYgX4AH7OAhfUMncYsopWhkW/ze9z8wTXc8BAEgDmt8zFCez1CtqJB/Ml
      SBUGDgk8oHYDsHKmx05baBaOBQ8LRGP5SULSbRtu34eLFootBIn0FvUZSnwTiSpb
      aHHRgWrMOVm07oSLWBuO3h/bj38zBuuqqVsAK8YuyoECAwEAAaMhMB8wHQYDVR0O
      BBYEFHxfAbr6PQ5Xgc+jVx+AGTPnnpWZMA0GCSqGSIb3DQEBCwUAA4IBAQAZBMJ2
      9/2idv1ztC6ArHtB4kw/nHHwthXFwtWAN7sRPB8tLW7fD8aJ43RQr5107Bg1Lgkm
      t+FZxpafqUC/mukjIzGzbW0COMSOTcWUGss+HxK6M6Fl9aOzKJMct1uOSpPFgjIt
      cGqydGZXR2FH93vXWoAotUwtZ119IixIdxpOJwYJg0HFn+GEfpU1PmiLfq2/uwqJ
      0hGCNfNcm9puagzhQrcDFOnolxjnYPSfSkU5wxlGo99yE5eJwoHXXU7csaZVttmx
      7sPj1lUENogXUM6JMqzSyEIm1XCOCL8rZJkZ781W5CwZhuJTNzV31sBREs8FaaCe
      ksu7Y48BmkUqw6E9
     </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>
      MIIDaDCCAlCgAwIBAgIDcB/YMA0GCSqGSIb3DQEBCwUAMGUxCzAJBgNVBAYTAlVL
      MRAwDgYDVQQIEwdCcmlzdG9sMRAwDgYDVQQHEwdCcmlzdG9sMRIwEAYDVQQKEwlG
      b3JnZVJvY2sxDzANBgNVBAsTBk9wZW5BTTENMAsGA1UEAxMEdGVzdDAeFw0xNjAz
      MTgxMTU2MjhaFw0yNjAzMTYxMTU2MjhaMGUxCzAJBgNVBAYTAlVLMRAwDgYDVQQI
      EwdCcmlzdG9sMRAwDgYDVQQHEwdCcmlzdG9sMRIwEAYDVQQKEwlGb3JnZVJvY2sx
      DzANBgNVBAsTBk9wZW5BTTENMAsGA1UEAxMEdGVzdDCCASIwDQYJKoZIhvcNAQEB
      BQADggEPADCCAQoCggEBAKNbl89eP6B8kZATNSPe3+OZ3esLx31hjX+dakHtPwXC
      AaCKqJFwjwKdxyRuPdsVG+8Dbk3PGhk26aJrSE93EpxeqmQqxNPMeD+N0/8pjkuV
      YWwPIQ/ts2iTiWOVn7wzlE4ASfvupqOR5pjuYMWNo/pd4L7QNjUCKoAt9H11HMyi
      P+6roo/EYgX4AH7OAhfUMncYsopWhkW/ze9z8wTXc8BAEgDmt8zFCez1CtqJB/Ml
      SBUGDgk8oHYDsHKmx05baBaOBQ8LRGP5SULSbRtu34eLFootBIn0FvUZSnwTiSpb
      aHHRgWrMOVm07oSLWBuO3h/bj38zBuuqqVsAK8YuyoECAwEAAaMhMB8wHQYDVR0O
      BBYEFHxfAbr6PQ5Xgc+jVx+AGTPnnpWZMA0GCSqGSIb3DQEBCwUAA4IBAQAZBMJ2
      9/2idv1ztC6ArHtB4kw/nHHwthXFwtWAN7sRPB8tLW7fD8aJ43RQr5107Bg1Lgkm
      t+FZxpafqUC/mukjIzGzbW0COMSOTcWUGss+HxK6M6Fl9aOzKJMct1uOSpPFgjIt
      cGqydGZXR2FH93vXWoAotUwtZ119IixIdxpOJwYJg0HFn+GEfpU1PmiLfq2/uwqJ
      0hGCNfNcm9puagzhQrcDFOnolxjnYPSfSkU5wxlGo99yE5eJwoHXXU7csaZVttmx
      7sPj1lUENogXUM6JMqzSyEIm1XCOCL8rZJkZ781W5CwZhuJTNzV31sBREs8FaaCe
      ksu7Y48BmkUqw6E9
     </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`.

   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.

```
<?xml version="1.0"?>
<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/>
    </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/>
    </Attribute>
    <Attribute name="basicAuthPassword">
      <Value/>
    </Attribute>
    <Attribute name="autofedEnabled">
      <Value>false</Value>
    </Attribute>
    <Attribute name="autofedAttribute">
      <Value/>
    </Attribute>
    <Attribute name="transientUser">
      <Value>anonymous</Value>
    </Attribute>
    <Attribute name="spAdapter">
      <Value/>
    </Attribute>
    <Attribute name="spAdapterEnv">
      <Value/>
    </Attribute>
    <Attribute name="fedletAdapter">
```

```
      <Value>com.sun.identity.saml2.plugins.DefaultFedletAdapter</Value>
    </Attribute>
    <Attribute name="fedletAdapterEnv">
      <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/>
    </Attribute>
    <Attribute name="localAuthURL">
      <Value/>
    </Attribute>
    <Attribute name="intermediateUrl">
      <Value/>
    </Attribute>
    <Attribute name="defaultRelayState">
      <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/>
    </Attribute>
    <Attribute name="wantArtifactResponseSigned">
      <Value/>
    </Attribute>
```

```
    <Attribute name="wantLogoutRequestSigned">
      <Value/>
    </Attribute>
    <Attribute name="wantLogoutResponseSigned">
      <Value/>
    </Attribute>
    <Attribute name="wantMNIRequestSigned">
      <Value/>
    </Attribute>
    <Attribute name="wantMNIResponseSigned">
      <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/>
    </Attribute>
    <Attribute name="saeSPLogoutUrl">
     </Attribute>
    <Attribute name="ECPRequestIDPListFinderImpl">
      <Value>com.sun.identity.saml2.plugins.ECPIDPFinder</Value>
    </Attribute>
    <Attribute name="ECPRequestIDPList">
      <Value/>
    </Attribute>
    <Attribute name="ECPRequestIDPListGetComplete">
      <Value/>
    </Attribute>
    <Attribute name="enableIDPProxy">
      <Value>false</Value>
    </Attribute>
    <Attribute name="idpProxyList">
      <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/>
    </Attribute>
    <Attribute name="basicAuthPassword">
      <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 the `sp-extended.xml` file, called `sp-extended-copy.xml`, and set `hosted="0"` in the root element of the copy.

   Use the copied file, `sp-extended-copy.xml`, when importing the Fedlet configuration into AM. AM 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 AM 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.

Deploying and Testing the Fedlet on the SP

There are two options for deploying the Fedlet, see the following for details:

- "To Install and Configure the Fedlet as a Demo Application"
- "To Embed the Java Fedlet in a Web Application"

To Install and Configure the Fedlet as a Demo Application

To deploy the Fedlet on the SP, you require the following:

- The configuration files, as created in "Creating and Configuring the Fedlet".
- The Fedlet WAR file, provided in the `Fedlet-7.1.4.zip`, within the AM distribution file; `AM-7.1.4.zip`.

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

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

2. Copy the fedlet configuration files to the `$HOME/fedlet` directory. The result may resemble the following:

```
$ cd /Users/tomcat-user/fedlet
$ ls -A1
  FederationConfig.properties
  fedlet.cot
  idp-extended.xml
  idp.xml
  sp-extended.xml
  sp.xml
```

3. Deploy the Fedlet WAR file into your web container:

```
$ cp fedlet.war /path/to/tomcat/webapps
```

   Upon completion, you can proceed to "Testing Fedlet Single Sign-on and Single Logout".

To Embed the Java Fedlet in a Web Application

The Fedlet WAR file, `fedlet.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:

1. Unpack the Fedlet ZIP file to a working directory, remove any files you do not want to keep, such as `index.jsp` or `fedletEncode.jsp`, and merge the Fedlet files with those of your web application.

2. To integrate single sign-on into your application, modify the functionality in the `fedletSampleApp.jsp` page 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>
```

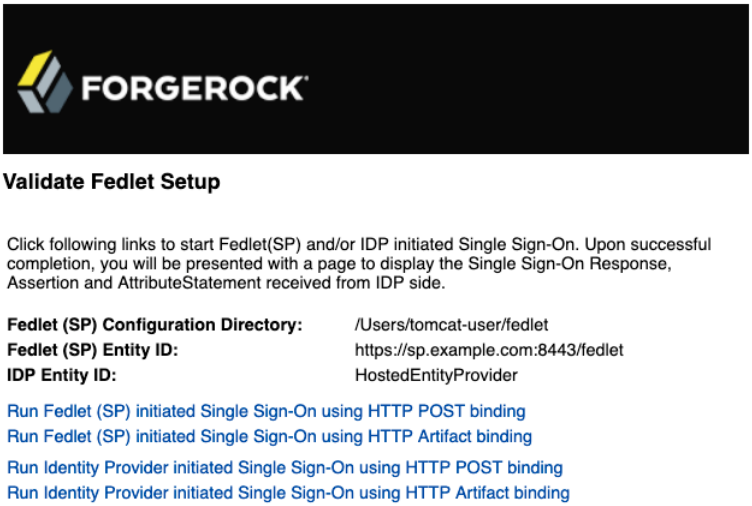
3. Build a WAR file from your web application with embedded Fedlet files.

   This is the version of the application to deploy. When you deploy your WAR file, also provide the Fedlet configuration files. For information on where to put the configuration files and how to deploy the WAR file with embedded Fedlet, see "To Install and Configure the Fedlet as a Demo Application".

Testing Fedlet Single Sign-on and Single Logout

To test single sign-on and single logout from the Fedlet, go to the Fedlet URL. For example, `https://sp.example.com:8443/fedlet`.

Try one or more examples from the Fedlet home page:



You can log in to the identity provider with username `demo` and password `Ch4ng31t`.

Integrating with the Fedlet WAR File

You can integrate your applications with the Java Fedlet to perform many of the SAML v2.0 service provider operations. The Java Fedlet offers the SAML v2.0 capabilities identified in "Fedlet Support for SAML v2.0 Features".

To Integrate Your Application

The Fedlet includes the following files that you can use when building your own service provider application:

`conf/`

Configuration files copied to the `$HOME/fedlet` directory 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 the `/export/fedlet/` directory, 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 demonstration of how to customize demo pages within the Fedlet:

1. Backup the `fedletSampleApp.jsp` file.

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

2. Edit the `fedletSampleApp.jsp` file to reduce it to a single redirection to the `myapp.jsp` page. 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. Go to the Fedlet URL, such as `https://openam.example.com:8443/fedlet/`, and try one of the login methods.

   After login, you are redirected to the `myapp.jsp` page.

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 the `saml2/jsp/spSSOInit.jsp` page in AM. Those parameters are described in spSSOInit.jsp Query Parameters.

For IDP-initiated single sign-on, call the appropriate page on the identity provider. AM's page is described in idpSSOInit.jsp Query 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`

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 the `saml2/jsp/spSingleLogoutInit.jsp` page in AM. Those parameters are described in spSingleLogoutInit.jsp Query Parameters.

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

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

Chapter 10

Reference

This reference section covers service provider, identity provider, and circle of trust configuration properties.For the global services reference, see Reference.

Hosted Identity Provider Configuration Properties

Once you have set up a hosted identity provider, you can configure it through the AM console under Realms > *Realm Name* > Applications > Federation > Entity Providers > *Provider Name*.

Assertion Content

The following groups appear under the Assertion Content tab:

Signing and Encryption

Request/Response Signing

Specifies which parts of messages the identity provider requires the service provider to sign digitally.

Encryption

When selected, the service provider must encrypt NameID elements.

Secret ID and Algorithms

Secret ID Identifier

Specifies an identifier for the secret ID AM uses for this entity provider, when resolving secrets.

For example, when this value is set to `demo`, the entity provider will use the following secret IDs:

- `am.applications.federation.entity.providers.saml2.demo.signing`
- `am.applications.federation.entity.providers.saml2.demo.encryption`

If not specified, AM uses the entity provider role-specific, default global secret IDs. For more information, see Secret ID Mappings for SAML v2.0 Signing and Encryption in the *Security Guide*.

Signing Algorithm

The algorithms the provider can use to sign the request/response attributes selected in the Request/Response Signing group.

These algorithms are exposed in the provider's metadata extension.

This property has no default.

Digest Algorithm

The digest algorithms the provider can use when signing the requests and responses selected in the Request/Response Signing group.

These algorithms are exposed in the provider's metadata extension.

This property has no default.

Encryption Algorithm

This field specifies two types of encryption algorithms for the provider:

- Symmetric algorithms, which the provider can use to encrypt the objects selected in the Encryption group. Select one or more AES algorithms from the drop-down list.

  Default: `http://www.w3.org/2001/04/xmlenc#aes128-cbc`

- Asymmetric algorithms, advertised as the provider's transport key algorithm. When SAML v2.0 token encryption is enabled, hosted providers should use the algorithm the remote provider is advertising when encrypting symmetric encryption keys.

  Select one or more algorithms from the drop-down list:

  + *Key Transport Algorithms*

    - http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p (default).
    - http://www.w3.org/2009/xmlenc11#rsa-oaep.

      When this algorithm is configured, AM will use the Mask Generation Function Algorithm property (Configure > Global Services > Common Federation Configuration) to create the transport key.

      For a list of supported mask generation function algorithms, see "Algorithms" in the *Reference*.
    - http://www.w3.org/2001/04/xmlenc#rsa-1_5.

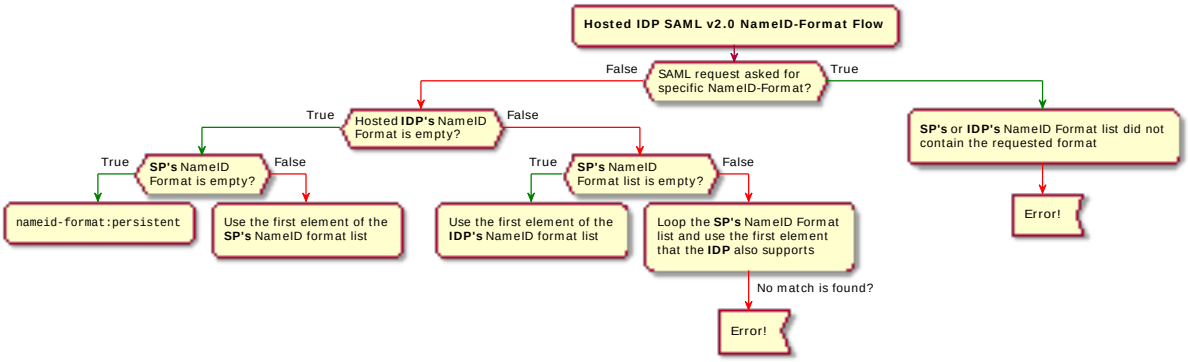
For security reasons, we strongly recommend that you do not use this option.

NameID Format

NameID Format List

Specifies the supported name identifiers for users that are shared between providers for single sign-on.

The following diagram shows how the hosted IDP decides which of the supported NameID formats is used:



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. With this flag set, AM Base64-encodes the profile attribute when adding it to the assertion.

Authentication Context

Mapper

Specifies a class that implements the `IDPAuthnContextMapper` interface and sets up the authentication context.

Default value: `com.sun.identity.saml2.plugins.DefaultIDPAuthnContextMapper`

Authentication Context

Specifies the authentication context classes the IDP supports, and any AM authentication mechanisms that are used when an SP specifies the respective class in a SAML v2.0 authentication request.

- The Predefined Reference drop-down list specifies the list of context references.
- The Key drop-down list specifies the authentication mechanism that AM uses when an SP specifies an authentication context class in a SAML v2.0 authentication request.

  + *Authentication Mechanisms Reference for the Key Drop-down List*

  Service

  AM will use the specified authentication chain or tree to authenticate the user.

  For example, in the Value field, enter *HmacOneTimePassword* to use the built-in one-time password example authentication tree, or *ldapService* to use the built-in chain that authenticates against the default identity store.

  Module

  AM will use the specified authentication module to authenticate the user.

  Authentication Level

  AM will authenticate the user with a method that has an associated authentication level equal to or higher than the specified value.

  If there is more than one suitable method, AM presents the available options by using a `ChoiceCallback`.

  For more information on using and returning callbacks during authentication, see "*Authenticating (REST)*" in the *Authentication and Single Sign-On Guide*.

  Important

  The `Role`, `User`, and `Resource URL` options are deprecated and should not be used.

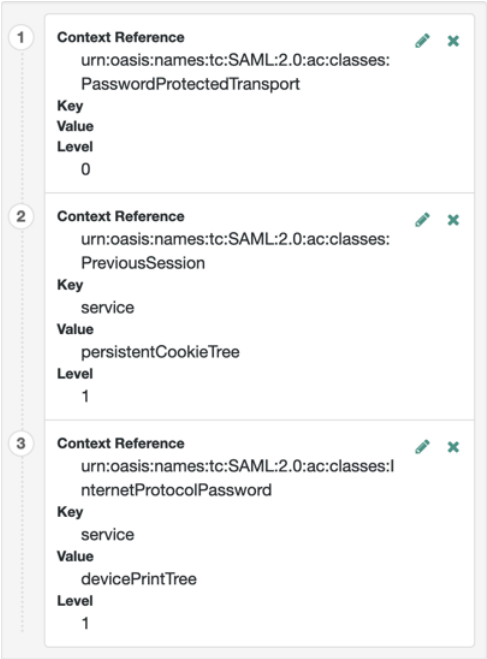
- The Value field specifies the name relative to the authentication mechanism you chose in the previous step. For example, if you chose the `Service` authentication mechanism, add the name of an authentication tree or chain in the Value field.
- The Level field specifies the numeric value of precedence of the supported context reference classes.

  Classes with higher numbers are considered stronger than lower numbered classes. The values determine which authentication classes can be used when the SP makes an authentication request that uses a comparison attribute; for example, `minimum` or `better`.

  Note that the value in the Level field should match the auth level of the specified chain, module, or tree. For example, if the specified authentication chain sets an auth level of 10, set the same

value in the Level field in the Authentication Context table. As AM compares the current auth level against the level specified in Authentication Context table, if the two values do not match, AM may require a logged in user to re-authenticate unnecessarily.

+ *Example*



For more information on authentication context classes, see Authentication Context for the OASIS Security Assertion Markup Language (SAML) V2.0 in the *SAML V2.0 Standard*.

Default value: `urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport`

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 endpoints.

Assertion Cache

Enabled

When enabled, cache assertions.

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`

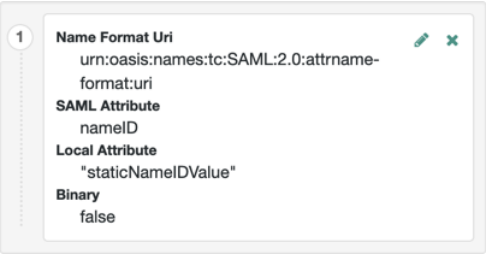
Attribute Map

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 "Adding User Profile Attributes" in the *Setup Guide*, for instructions on allowing additional attributes in user profiles.

To see the profile attributes available for an LDAP identity repository, log in to the AM console, and go to Realms > *Realm Name* > Identity Stores > User Configuration. Check the LDAP User Attributes list.

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 example:



Account Mapper

Account Mapper

Specifies a class that implements `AccountMapper` to map remote users to local user profiles.

Disable NameID Persistence

Disables the storage of the NameID values in the user data store for all NameIDs issued by the IDP instance as long as the NameID format is anything but the persistent NameID format: `urn:oasis:names:tc:SAML:2.0:nameid-format:persistent`. That is, you can disable the storage of NameID values with persistent NameID-Format if and only if there is a NameID value mapping set up for the NameID-Format.

> **Note**
>
> By preventing the storage of the NameID values, the `ManageNameID` and the `NameIDMapping` SAML profiles will no longer work when using any persistent NameID formats. Existing account links that have been established and stored are not removed when disabling NameID persistence.

Default value: `false`

Local Configuration

Auth URL

If present, overrides the default UI login URL used to authenticate users during federation.

Use this property to specify an alternative URL for authenticating users, for example, if you have created a custom user interface other than the UI.

The specified authentication URL is responsible for authenticating the federated user and must establish a session in AM, and return the SSO token value in the configured cookie name, usually `iPlanetDirectoryPro`.

The domain of the authentication URL must be configured in AM so that the cookie is accepted, and if host cookies are configured in AM, then the fully qualified domain name of the authentication URL must be identical to that of the AM instance.

For more information on configuring the domains AM accepts in the SSO cookies, see "Configuring the Cookie Domain" in the *Security Guide*.

> **Important**
>
> AM redirects users to the URL specified, and appends a `goto` parameter. The parameter contains the URL the user must be redirected to after authentication. The specified authentication URL must not override the `goto` parameter, as that would redirect the user elsewhere and federation will fail.
>
> For more information, see "Configuring Success and Failure Redirection URLs" in the *Authentication and Single Sign-On Guide*.

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.

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`.

Ensure the MetaAlias is unique for each provider configured in a CoT and in the realm.

IDP Service Attributes

Artifact Resolution Service

Specifies the endpoint to manage artifact resolution. The Index is a unique number identifier for the endpoint.

Single Logout Service

Specifies the endpoints to manage single logout, depending on the SAML binding selected.

Manage NameID Service

Specifies the endpoints to manage name identifiers, depending on the SAML binding selected.

Single SignOn Service

Specifies the endpoints to manage single sign-on.

Assertion ID Request Service

Specifies the endpoints to request for an specific assertion by referring to its assertion ID.

NameID Mapping

URL

Specifies the endpoint to manage name identifier mapping.

Advanced Settings

The following properties appear under the Advanced tab:

SAE Configuration

IDP URL

Specifies the endpoint to manage 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. For single logout (SLO) operations, AM validates the redirection URL in the `RelayState` parameter against this list. If the `RelayState` parameter's value is in the list, AM allows redirection to the `RelayState` URL. If it is not in the list, a browser error occurs.

Use the pattern matching rules described in "Configuring Success and Failure Redirection URLs" in the *Authentication and Single Sign-On Guide* to specify URLs in the list.

If you **DO NOT** specify any URLs in this property, AM only allows redirection to `RelayState` URLs that match the domain of the instance. Any other URL will cause a browser error.

> **Note**
>
> This property does not apply to IDP-initiated single sign-on, where the validation of the `RelayState>` parameter should be performed on the service provider.

IDP Adapter

IDP Adapter Class

Specifies a class to invoke immediately before sending a SAML v2.0 response.

Remote Identity Provider Configuration Properties

Once you have set up a remote identity provider, you can configure it through the AM console under Realms > *Realm Name* > Applications > Federation > Entity Providers > *Provider Name*.

Assertion Content

The following properties appear under the Assertion Content tab:

Signing and Encryption

Request/Response Signing

Specifies which parts of messages the identity provider requires the service provider to sign digitally.

Encryption

When selected, the service provider must encrypt NameID elements.

NameID Format

NameID Format List

Specifies the supported name identifiers for users that are shared between providers for single sign-on.

The information on how a hosted SP determines the NameID format to use, see NameID Format.

Basic Authentication

Enabled, User Name, Password

When enabled, authenticate with the specified user name and password at SOAP endpoints.

Services

The following properties appear under the Services tab:

IDP Service Attributes

Artifact Resolution Service

Specifies the endpoint to manage artifact resolution. The Index is a unique identifier for the endpoint.

Single Logout Service

Specifies the endpoints to manage single logout, depending on the SAML binding selected.

Manage NameID Service

Specifies the endpoints to manage name identifiers, depending on the SAML binding selected.

Single SignOn Service

Specifies the endpoints to manage single sign-on.

NameID Mapping

URL

Specifies the endpoint to manage name identifier mapping.

Hosted Service Provider Configuration Properties

Once you have set up a hosted service provider, you can configure it through the AM console by navigating to > *Realm Name* > Applications > Federation > Entity Providers > *Provider Name*.

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.

Secret ID and Algorithms

Secret ID Identifier

Specifies an identifier for the secret ID AM uses for this entity provider, when resolving secrets.

For example, when this value is set to `demo`, the entity provider will use the following secret IDs:

- `am.applications.federation.entity.providers.saml2.demo.signing`
- `am.applications.federation.entity.providers.saml2.demo.encryption`

If not specified, AM uses the entity provider role-specific, default global secret IDs. For more information, see Secret ID Mappings for SAML v2.0 Signing and Encryption in the *Security Guide*.

Signing Algorithm

The algorithms the provider can use to sign the request/response attributes selected in the Request/Response Signing group.

These algorithms are exposed in the provider's metadata extension.

This property has no default.

Digest Algorithm

The digest algorithms the provider can use when signing the requests and responses selected in the Request/Response Signing group.

These algorithms are exposed in the provider's metadata extension.

This property has no default.

Encryption Algorithm

This field specifies two types of encryption algorithms for the provider:

- Symmetric algorithms, which the provider can use to encrypt the objects selected in the Encryption group. Select one or more AES algorithms from the drop-down list.

  Default: `http://www.w3.org/2001/04/xmlenc#aes128-cbc`

- Asymmetric algorithms, advertised as the provider's transport key algorithm. When SAML v2.0 token encryption is enabled, hosted providers should use the algorithm the remote provider is advertising when encrypting symmetric encryption keys.

  Select one or more algorithms from the drop-down list:

  + *Key Transport Algorithms*

    - http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p (default).
    - http://www.w3.org/2009/xmlenc11#rsa-oaep.

      When this algorithm is configured, AM will use the Mask Generation Function Algorithm property (Configure > Global Services > Common Federation Configuration) to create the transport key.

      For a list of supported mask generation function algorithms, see "Algorithms" in the *Reference*.

    - http://www.w3.org/2001/04/xmlenc#rsa-1_5.

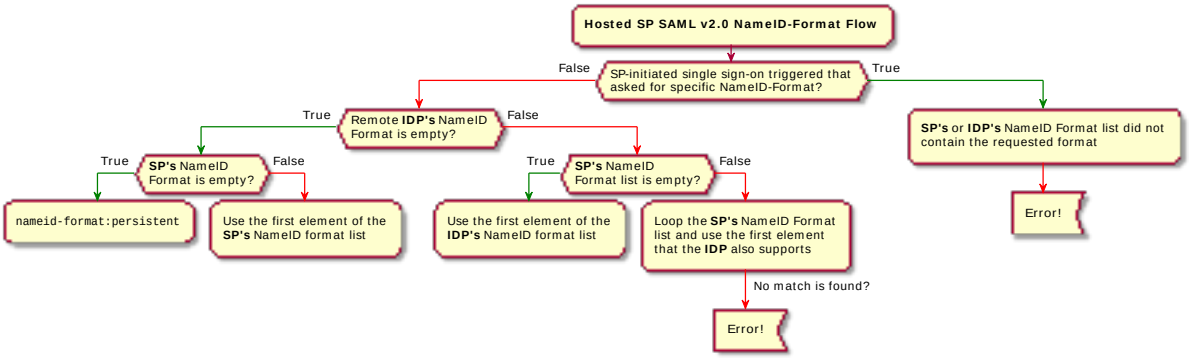
For security reasons, we strongly recommend that you do not use this option.

NameID Format

NameID Format List

Specifies the supported name identifiers for users that are shared between providers for single sign-on.

The following diagram shows how the hosted SP decides which of the supported NameID formats is used:



Disable NameID Persistence

Disables the storage of NameIDs in the user data store, even if the `NameID` format is `urn:oasis:names:tc:SAML:2.0:nameid-format:persistent` in the received assertion and the account mapper has identified the local user.

You may want to disable storage of NameID values if you are using a read-only data store, or an external identity store that does not have the AM identity schemas applied.

> **Note**
>
> When local authentication is utilized for account linking purposes, disabling federation persistence requires end users to authenticate locally for each SAML-based login.

Default value: `false`

Authentication Context

Mapper

Specifies a class that implements the `SPAuthnContextMapper` interface and sets up the authentication context.

Default: `com.sun.identity.saml2.plugins.DefaultSPAuthnContextMapper`

Authentication Context

Specifies the authentication context classes the SP supports, and any AM authentication mechanisms that are used when an IDP specifies the respective class in a SAML v2.0 authentication request.

- The Predefined Reference drop-down list specifies the list of context references.
- The Key drop-down list specifies the authentication mechanism that AM uses when an SP specifies an authentication context class in a SAML v2.0 authentication request.

  + *Authentication Mechanisms Reference for the Key Drop-down List*

  Service

  AM will use the specified authentication chain or tree to authenticate the user.

  For example, in the Value field, enter *HmacOneTimePassword* to use the built-in one-time password example authentication tree, or *ldapService* to use the built-in chain that authenticates against the default identity store.

  Module

  AM will use the specified authentication module to authenticate the user.

  Authentication Level

  AM will authenticate the user with a method that has an associated authentication level equal to or higher than the specified value.

  If there is more than one suitable method, AM presents the available options by using a `ChoiceCallback`.

  For more information on using and returning callbacks during authentication, see "*Authenticating (REST)*" in the *Authentication and Single Sign-On Guide*.

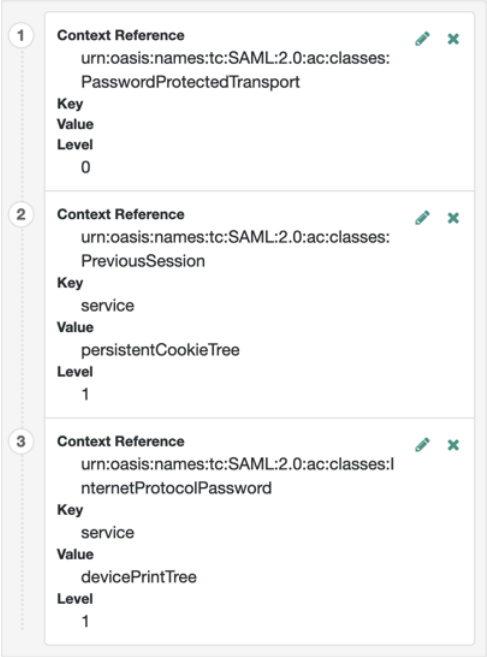
  > **Important**
  >
  > The `Role`, `User`, and `Resource URL` options are deprecated and should not be used.

- The Value field specifies the name relative to the authentication mechanism you chose in the previous step. For example, if you chose the `Service` authentication mechanism, add the name of an authentication tree or chain in the Value field.
- The Level field specifies the numeric value of precedence of the supported context reference classes.

Classes with higher numbers are considered stronger than lower numbered classes. The values determine which authentication classes can be used when the SP makes an authentication request that uses a comparison attribute; for example, `minimum` or `better`.

Note that the value in the Level field should match the auth level of the specified chain, module, or tree. For example, if the specified authentication chain sets an auth level of 10, set the same value in the Level field in the Authentication Context table. As AM compares the current auth level against the level specified in Authentication Context table, if the two values do not match, AM may require a logged in user to re-authenticate unnecessarily.

+ *Example*



For more information on authentication context classes, see Authentication Context for the OASIS Security Assertion Markup Language (SAML) V2.0 in the *SAML V2.0 Standard*.

Default value: `urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport`

Comparison Type

Used in conjunction with the Default Authentication Context to specify the possible range of authentication mechanisms the IDP can choose from.

For example, if the Comparison Type field is set to `better`, and the `PasswordProtectedTransport` authentication context class is selected in the Default Authentication Context field, the IDP must select an authentication mechanism with a higher level assigned.

For information on how AM maps the authentication mechanism to authentication context classes, and how to set the precedence, see Authentication Context.

Default: `exact`

Include Request Authentication Context

Specifies whether to include an authentication context class as the Requested Authentication Context in the SAML v2.0 Authentication Request.

Default: Enabled

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 endpoints.

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 session properties, or user profile attributes.

The value of Key is a SAML attribute sent in an assertion, and the value of Value is a property in the user's session, or an attribute of the user's profile.

By default, the SP maps the SAML attributes it receives to equivalent-named session properties. However, when the SP is configured to create identities during autofederation and the identity

does not exist yet, the SP maps the SAML attributes to their equivalents in the newly-created user profile.

The special mapping `Key: *, Value: *` means that the SP maps each attribute it receives in the assertion to equivalent-named properties or attributes. For example, if the SP receives `mail` and `firstname` in the assertion, it maps them to `mail` and `firstname` respectively.

Remove the special mapping and add key pairs to the map if:

- (During autofederation) The attributes in the IdP's and the SP's identity stores do not match.
- You need control over the names of the session properties.
- You need control over which attributes the SP should map, because the IdP adds too many to the assertion.

For example, if the the SAML attribute is `firstname` and you want the SP to map it to a session property/user profile attribute called `cn`, create a mapping similar to `Key: firstname, Value: cn`.

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.

Transient User

Specifies the user profile to map all identity provider users when sending transient name identifiers.

Artifact Message Encoding

Artifact Message Encoding

Specifies the message encoding format for artifacts.

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.

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`.

Ensure the MetaAlias is unique for each provider configured in a CoT and in the realm.

SP Service Attributes

Single Logout Service

Specifies the endpoints to manage single logout, depending on the SAML binding selected.

Manage NameID Service

Specifies the endpoints to manage name identifiers, depending on the SAML binding selected.

Assertion Consumer Service

Specifies the endpoints to consume assertions, with Index corresponding to the index of the URL in the standard metadata.

The scheme, FQDN, and port configured must exactly match those of the service provider as they appear in its metadata.

To determine the service provider's endpoint URL, AM uses the Base URL service, if configured.

If the URL does not match, the SAML v2.0 flow will fail and AM will log `Invalid Assertion Consumer Location specified` in the audit log file.

Advanced Settings

The following properties appear under the Advanced tab:

SAE Configuration

SP URL

Specifies the endpoint to manage Secure Attribute Exchange requests.

SP Logout URL

Specifies the endpoint 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, AM includes a `Scoping` element in the authentication request to enable the request to be proxied.

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. AM validates the redirection URL in the `RelayState` parameter against this list. If the `RelayState` parameter's value is in the list, AM allows redirection to the `RelayState` URL. If it is not in the list, a browser error occurs.

Use the pattern matching rules described in "Configuring Success and Failure Redirection URLs" in the *Authentication and Single Sign-On Guide* to specify URLs in the list.

If you **DO NOT** specify any URLs in this property, AM only allows redirection to `RelayState` URLs that match the domain of the instance. Any other URL will cause a browser error.

Remote Service Provider Configuration Properties

Once you have set up a remote service provider, you can configure it through the AM console by going to Realms > *Realm Name* > Applications > Federation > Entity Providers > *Provider Name*.

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.

NameID Format

NameID Format List

Specifies the supported name identifiers for users that are shared between providers for single sign-on.

The information on how a hosted SP determines the NameID format to use, see NameID Format.

Disable NameID Persistence

Disables the storage of NameID values at the IDP when generating an assertion for this remote SP.

Default value: `false`

Basic Authentication

Enabled, User Name, Password

When enabled, require authentication with the specified user name and password at SOAP endpoints.

Assertion Processing

The following properties appear under the Assertion Processing tab:

Attribute Mapper

Attribute Map

Override any mappings of attributes to user profile attributes at the IDP.

Artifact Message Encoding

Encoding

Specifies the message encoding format for artifacts.

Services

The following properties appear under the Services tab:

SP Service Attributes

Single Logout Service

Specifies the endpoints to manage single logout, depending on the SAML binding selected.

Manage NameID Service

Specifies the endpoints to manage name identifiers, depending on the SAML binding selected.

Assertion Consumer Service

Specifies the endpoints to consume assertions, with Index corresponding to the index of the URL in the standard metadata.

Advanced

The following properties appear under the Advanced tab:

Request Processing

Skip Endpoint Validation For Signed Requests

When enabled, AM does not verify Assertion Consumer Service URL values provided in SAML authentication requests. This allows the Assertion Consumer Service URL to contain dynamic query parameters, for example.

As assertion consumer service URL verification is part of the SAML v2.0 spec, you can only skip validation if the authentication request is digitally signed by the SP. To digitally sign authentication requests, in the remote SP configuration navigate to Assertion Content > Signing and Encryption > Request/Response Signing, and select Authentication Requests Signed.

> **Important**
>
> You must configure the remote SP to sign the authentication request.
>
> AM returns an error if it receives an unsigned authentication request and this option is enabled.

SAE Configuration

SP URL

Specifies the endpoint to manage Secure Attribute Exchange requests.

SP Logout URL

Specifies the endpoint of the service provider that can handle global logout requests.

IDP Proxy

IDP Proxy enabled

When enabled, the authentication requests from this service provider can be proxied.

Proxy all requests

When enabled, AM proxies every authentication request from the SP, whether it contains a `Scoping` element or not.

The IDP Proxy option must be enabled for this option to take effect.

Introduction enabled

When enabled, use introduction cookies to find the proxy identity provider.

> **Note**
>
> This property only works with a non-default *SAML2IDPProxyFRImpl* implementation, and will be deprecated in a future release.

Use IDP Finder

When enabled, use the IDP finder service to determine the IDP to which to proxy authentication requests.

Proxy Count

Specifies the maximum number of proxy identity providers. AM sets the specified value in the `Scoping` element of authentication request it proxies for this SP.

The Proxy all requests option must be enabled for this option to take effect.

IDP Proxy List

Specifies a list of URIs identifying preferred proxy identity providers.

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 > Federation > Circle of Trust > *Circle of Trust Name*.

Name

String to refer to the circle of trust.

Once you have set up a circle of trust, the Name cannot be configured.

Description

Short description of the circle of trust.

Status

Whether this circle of trust is operational.

Entity Providers

Known hosted and remote identity and service providers participating in this circle of trust.

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: `https://discovery.example.com:8443/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: `https://discovery.example.com:8443/openam/saml2reader`.

SAML v2.0 Advanced Properties

To configure SAML v2.0 advanced properties, in the AM console, go to Configure > Server Defaults > Advanced.

`openam.saml.decryption.debug.mode`

When enabled, AM decrypts SAML v2.0 messages that are sent and received, and writes the content to the debug logs.

As these messages may contain user information, you should not enable this property in production environments.

Default: `False`

`org.forgerock.openam.saml2.authenticatorlookup.skewAllowance`

Specifies the allowable time difference, in seconds, between any existing session the user may have, and the current time when an authentication request specifies `ForceAuthn`.

If the authenticated user's session was created outside of the allowable time range, AM rejects the assertion, and re-authentication is required.

Default: `60`

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 identities 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 agent profile configuration information, typically created to delegate agent profile creation to the user installing a web or Java 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 AM 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	AM 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. AM 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. AM can play this role in the OAuth 2.0 authorization framework.
Client-based OAuth 2.0 tokens	After a successful OAuth 2.0 grant flow, AM returns a token to the client. This differs from CTS-based OAuth 2.0 tokens, where AM returns a *reference* to token to the client.
Client-based sessions	AM sessions for which AM returns session state to the client after each request, and require it to be passed in with the subsequent

	request. For browser-based clients, AM sets a cookie in the browser that contains the session information. For browser-based clients, AM sets a cookie in the browser that contains the session state. When the browser transmits the cookie back to AM, AM decodes the session state from the cookie.
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 AM configuration data.
Cross-domain single sign-on (CDSSO)	AM capability allowing single sign-on across different DNS domains.
CTS-based OAuth 2.0 tokens	After a successful OAuth 2.0 grant flow, AM returns a *reference* to the token to the client, rather than the token itself. This differs from client-based OAuth 2.0 tokens, where AM returns the entire token to the client.
CTS-based sessions	AM sessions that reside in the Core Token Service's token store. CTS-based sessions might also be cached in memory on one or more AM servers. AM tracks these 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.
Delegation	Granting users administrative privileges with AM.
Entitlement	Decision that defines which resource names can and cannot be accessed for a given identity 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 AM.
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 AM on the service provider side; AM lets you create Java Fedlets.
Hot swappable	Refers to configuration properties for which changes can take effect without restarting the container where AM 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 agent	Java web application installed in a web container that acts as a policy enforcement point, 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	Java, web, or custom agent that intercepts requests for resources, directs principals to AM for authentication, and enforces policy decisions from AM.
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, AM associates the Subject with the Principal.
Privilege	In the context of delegated administration, a set of administrative tasks that can be performed by specified identities in a given realm.
Provider federation	Agreement among providers to participate in a circle of trust.
Realm	AM unit for organizing configuration and identity information. Realms can be used for example when different parts of an organization have different applications and identity stores, and when different organizations use the same AM 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 AM 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).
Authentication Session	The interval while the user or entity is authenticating to AM.
Session	The interval that starts after the user has authenticated and ends when the user logs out, or when their session is terminated. For browser-based clients, AM manages user sessions across one or more applications by setting a session cookie. See also CTS-based sessions and Client-based sessions.

Session high availability	Capability that lets any AM 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 AM after successful authentication. For a CTS-based sessions, 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 AM servers configured the same way, accessed through a load balancer layer. The load balancer handles failover to provide service-level availability. The load balancer can also be used to protect AM services.
Standard metadata	Standard federation configuration information that you can share with other access management software.
Stateless Service	Stateless services do not store any data locally to the service. When the service requires data to perform any action, it requests it from a data store. For example, a stateless authentication service stores session state for logged-in users in a database. This way, any server in the deployment can recover the session from the database and service requests for any user. All AM services are stateless unless otherwise specified. See also Client-based sessions and CTS-based sessions.
Subject	Entity that requests access to a resource When an identity successfully authenticates, AM associates the identity with the Principal that distinguishes it from other identities. An identity can be associated with multiple principals.
Identity store	Data storage service holding principals' profiles; underlying storage can be an LDAP directory service or a custom `IdRepo` implementation.
Web Agent	Native library installed in a web server that acts as a policy enforcement point with policies based on web page URLs.