Implementing a SAML sender-vouches subject confirmation scenario in WebSphere Application Server

Skill Level: Intermediate

Chunlong Liang (liangch@us.ibm.com)
WebSphere Web Services Developer
IBM

Ching-Yun (C.Y.) Chao, Ph.D. (cyc@us.ibm.com)
Senior Software Engineer
IBM

02 Nov 2011

This article describes how to configure and use SAML sender-vouches tokens in IBM® WebSphere® Application Server (V7.0 Fix Pack 9 and later). The SAML sender-vouches subject confirmation method is particularly useful when a message sender acts on behalf of a web services client to access downstream web services and must assert client identity and security attributes. This method requires the message sender and receiver to ensure integrity of SOAP messages and SAML assertions. This article explains how to setup policy set and application-specific bindings to use message level integrity protection and transport level confidential protection. A sample application is provided for reference, with fast path instructions.

Introduction

IBM WebSphere Application Server V7 Fix Pack 9 added support for sender-vouches subject confirmation usage scenarios to the Security Assertion Markup Language (SAML) features delivered in Fix Pack 7 (see Resources). The SAML sender-vouches subject confirmation method is useful when a message sender must act on behalf of the subject of the SAML token. Message senders and receivers are required by the OASIS Web Services Security: SAML Token Profile 1.1 specification (see Resources) to ensure the integrity of SOAP messages and

vouch for SAML tokens. The WebSphere Application Server V7 Fix Pack 9 sender-vouches subject confirmation implementation requires the integrity of SAML tokens and the SOAP messages the contain to be protected either at the transport level using SSL with sender certificate authentication, or at the message level using digital signature.

This article describes an approach for implementing sender-vouches subject confirmation that combines message level digital signature for integrity protection, and transport level SSL for confidentiality protection. Moreover, this article illustrates how to use the sender-vouches confirmation method with the trust model described in the SAML assertions across WebSphere Application Server security domains article, published earlier.

Prerequisites

To follow the instructions and run the sample application included with this article, you must have WebSphere Application Server V7 with Fix Pack 9 or later installed and running. Further, it is assumed that you have enabled global security and application security, and that you have setup acme.com as a trusted authentication realm; see the configuration procedures described in the earlier article.

Fast path to testing sender-vouches confirmation

Assuming you are already familiar with WebSphere Application Server and the concepts described here, you can follow the steps in this section to quickly set up the sample applications included with this article to see the sender-vouches SAML tokens in action:

1. Unzip the svsample.zip file included with this article to a temporary directory.

2. Run `keygen.cmd` from a command line and copy all the jceks files to the **<WAS_INSTALL_ROOT>/profiles/<PROFILE>/etc/ws-security/samples** directory.

3. Extract SAMLIssuerConfig.properties using wsadmin script
   ```
   $AdminConfig extract
   cells/cell_name/sts/SAMLIssuerConfig.properties
   c:/temp/SAMLIssuerConfig.properties
   ```

   and modify it using `sample_config/SAMLIssuerConfig.properties` as a reference. You might want to save the original SAMLIssuerConfig.properties file. After you are done, check it in using wsadmin script

```
$AdminConfig checkin
cells/cell_name/sts/SAMLIssuerConfig.properties
c:/temp/SAMLIssuerConfig.properties.
```

4. Import the sample_policy/SAML20SV_Sign_Not_Encrypt.zip file into your application policy sets.
5. Deploy the sample_app/EJBWSClientApp.ear and sample_app/EJBWSProviderApp.ear files.
6. Attach the SAML20SV_Sign_Not_Encrypt policy set to EJBWSClientApp and assign the SAML20SVSampleClientBinding binding. (SAML20SVSampleClientBinding is pre-configured and packaged in the included client application.)
7. Attach the SAML20SV_Sign_Not_Encrypt policy set to EJBWSProviderApp and assign the SAML20SVSampleProviderBinding binding. (SAML20SVSampleProviderBinding is pre-configured and packaged in the included provider application.)
8. Start both EJBWSClientApp and EJBWSProviderApp.
9. Use a web browser to navigate to `http://localhost:9080/testEJBWSClientWeb/testEJBWSBrowserClient.j` When the application begins, enter `https://localhost:9443/testEJB_HTTPRouter/HelloBeanService` for the EJB Service URL, and then submit the request. You might need to modify ports 9080 and 9443 based on your WebSphere Application Server installation.

The remainder of this article describes the sender-vouches policy set and binding configuration in detail so that you can customize them to resemble your own usage scenarios.

Sample web services and client

The sample web services application provided in this article is similar to the one provided in the SAML assertions across WebSphere Application Server security domains article mentioned earlier. The sample Web services client generates a self-issued SAML token that represents an authenticated user. You need to modify one line of code in the com.ibm.wss.sample.ejbws.SamlTokenGenerator.java file in the sample client to specify the sender-vouches subject confirmation method. (Listing 1)

Listing 1

```
// RequesterConfig reqData = samlFactory.newBearerTokenGenerateConfig();
        RequesterConfig reqData = samlFactory.newSenderVouchesTokenGenerateConfig();
```

The IBM Rational® Application Developer project exchange file, web services provider application EAR file, and the web services client application EAR file are also included in the download material. You can deploy the Web services client and provider application EARs.

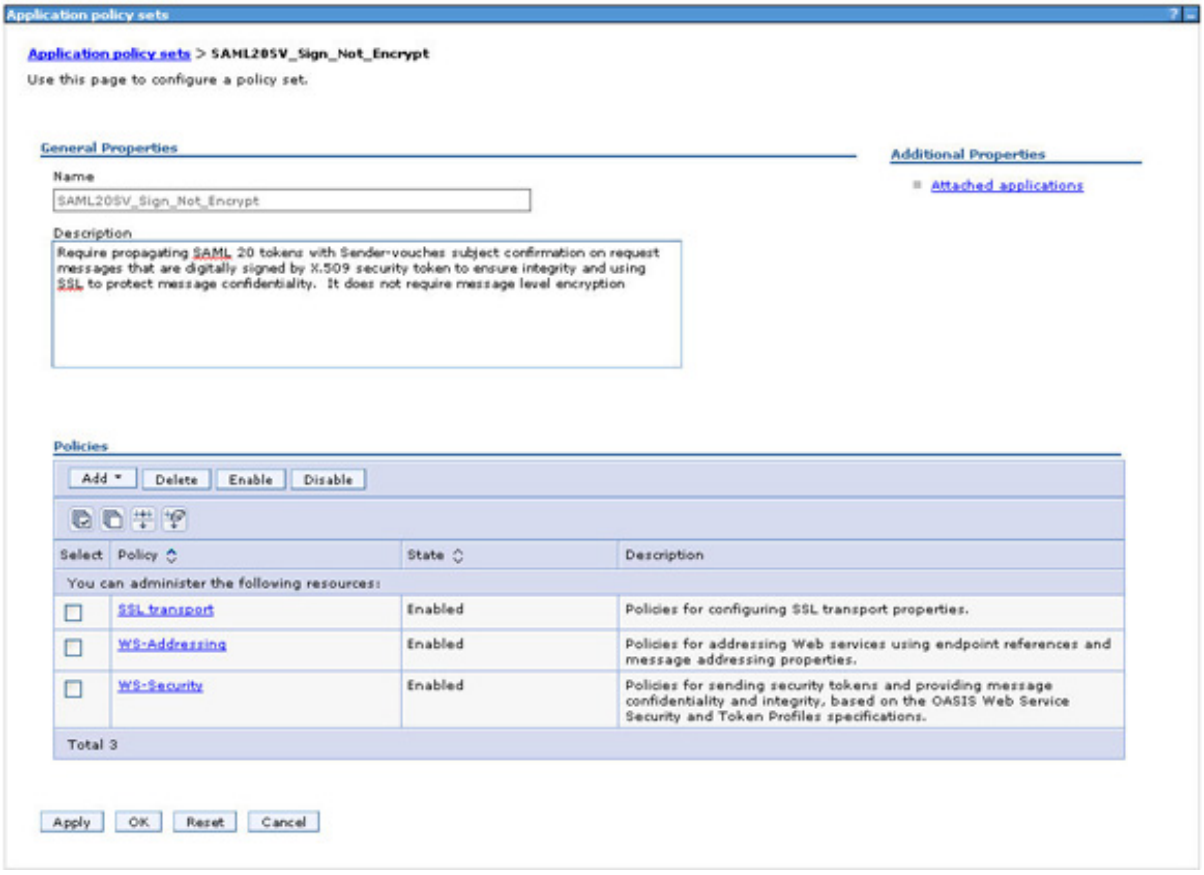
Configuring sender-vouches subject confirmation

Because WebSphere Application Server does not ship with a default policy set that requires sender-vouches subject confirmation, the steps below explain how you can setup a policy set to require a sender to use its private key to digitally sign SOAP messages and SAML tokens, and to use SSL to protect message confidentiality.

Create a policy set

You want to create a policy set to use an X.509 security token to digitally sign a SAML token and selected parts of the message. It might save you time to customize a copy of an existing policy set than to create a new policy set from scratch. Following that approach, there are two default policy sets you can choose from when using SAML 2.0 tokens: **SAML20 Bearer WSHTTPS default** and **SAML20 Bearer WSSecurity default** (both are default policy sets shipped in Fix Pack 7 and both require roughly same amount of customization). The policy set used in this example is SAML20 Bearer WSSecurity default. Figure 1 shows the result after a copy of this policy set is renamed to **SAML20SV_Sign_Not_Encrypt** and an **SSL Transport PolicyType** is added for using SSL. You will also need to add policy configuration to sign SAML tokens and remove message level encryption policy configuration. (For example purposes, a pre-configured policy set called **SAML20SV_Sign_Not_Encrypt** is included with the download materials that you can simply import into your WebSphere Application Server installation.)

Figure 1. Create a new policy set that requires signing SAML tokens and uses SSL



The SAML20SV_Sign_Not_Encrypt policy set you cloned above already contains the policy configuration to sign the message body, WS-Addressing headers, and the timestamp of request messages. The main change to the WS-Security PolicyType is to add policy configuration to sign SAML tokens. You can use an XPATH expression to specify the security token for signing.

When signing a SAML security token, the resulting digital signature element will reference the SAML token via a SecurityTokenReference element (see Resources). The XPATH expression will actually specify the SecurityTokenReference instead of the actual SAML security token. The WSSecurity policy already has a list of signed parts and signed elements policy configuration. However, you cannot use the existing list and will need to create a new list for adding the XPATH expressions that select the SecurityTokenReference. As you will see later, the reason you need a new list is because the binding configuration of a list of signed parts can only specify one transformation algorithm. The original list specifies the http://www.w3.org/2001/10/xml-exc-c14n# canonicalization algorithm. You need to create a new signed parts list for the SAML token to specify the Security Token Reference Transformation (STR-T) algorithm in the binding configuration. Figure 2 shows that your new list of signed parts and elements called **app_signparts_strd**.

Figure 2. Adding a new signed parts app_signparts_strd element in order to

specify

Reference transformation binding configuration

You need to add two XPATH expressions, one for the SOAP 1.1 namespace and one for the SOAP 1.2 namespace, to app_signparts_strd for selecting the SecurityTokenReference element. To do this, you can use the example XPATH expressions from the Information Center article, Signing SAML tokens at the message level in the WebSphere Application Server. You can just cut-and-paste the following two expressions into the administrative console configuration panel:

- `/*[namespace-uri()='http://schemas.xmlsoap.org/soap/envelope/' and local-name()='Envelope']/*[namespace-uri()='http://schemas.xmlsoa and local-name()='Header']/*[namespace-uri()='http://docs.oasis-open. and local-name()='Security']/*[namespace-uri()='http://docs.oasis-ope and local-name()='SecurityTokenReference']`
- `/*[namespace-uri()='http://www.w3.org/2003/05/soap-envelope' and local-name()='Envelope']/*[namespace-uri()='http://www.w3.org/200 and local-name()='Header']/*[namespace-uri()='http://docs.oasis-open. and local-name()='Security']/*[namespace-uri()='http://docs.oasis-ope and local-name()='SecurityTokenReference']`

Figure 3 shows the two XPATH expressions configuration.

Figure 3. Configure a new app_signparts_strd configuration

Remove message level encryption policy configuration

Because you will use SSL to provide message confidentiality protection, remove the encrypted configuration parts from both the request and response message part protection on the admin console panels. Uncheck the **Use the same token types that are used for integrity protection** flag on the Asymmetric signature and encryption policies panel. As a result, an X.509 security token will thus be used for the digital signature, but not for encryption purposes. Save all the changes and your have completed customizing your new policy set.

Setting up public and private keys

You need to set up a trust relationship among the three parties involved in a sender-vouches scenario:

- SAML tokens issuer (SAML issuer)
- Web services message sender (sender)
- Web services message receiver (receiver).

The receiver needs to verify the integrity of the request messages and SAML tokens, and that the sender is trusted to vouch for request messages and SAML tokens. The receiver also needs to verify the integrity of the SAML tokens and that the SAML issuer is trusted to assert the SAML subject identity and other claims. In other words, the receiver needs to verify that the SAML tokens are signed by an issuer that the receiver trusts. Finally, the sender needs to verify the integrity of the response messages.

Use the JDK keytool to generate three private and public key pairs to setup the three trust relationships (Figure 4):

- Create three key store files (**saml-issuer.jceks**, **sender.jceks**, and **receiver.jceks**) and a trust store file (**validator.jceks**).
- All private keys in key store files should have the same key password: `keypass`.
- All three key store files and the trust store file should use the same key store password: `storepass`.

Figure 4. JDK keytool commands for creating key files and public/private key pairs

```
keytool -genkey -alias samlissuer -keystore saml-issuer.jceks -dname "CN=SAMLIssuer,
O=Example"  -storepass storepass -keypass keypass -storetype jceks -validity 5000 -keyalg
RSA  -keysize 2048

keytool -genkey -alias sender    -keystore sender.jceks     -dname "CN=Purchasing,
O=Example"  -storepass storepass -keypass keypass -storetype jceks -validity 5000 -keyalg
RSA  -keysize 2048

keytool -genkey -alias receiver  -keystore receiver.jceks   -dname "CN=Marketplace,
O=BigCorp"  -storepass storepass -keypass keypass -storetype jceks -validity 5000 -keyalg RSA
-keysize 2048

keytool -export -alias samlissuer -file issuerpub.cer  -keystore saml-issuer.jceks -storepass
storepass -storetype jceks

keytool -export -alias sender    -file clientpub.cer  -keystore sender.jceks     -storepass storepass
-storetype jceks

keytool -export -alias receiver  -file servicepub.cer -keystore receiver.jceks   -storepass
storepass -storetype jceks

keytool -import -alias samlissuer -file issuerpub.cer  -keystore validator.jceks  -storepass
storepass -storetype jceks -keypass keypass -noprompt

keytool -import -alias receiver  -file servicepub.cer -keystore sender.jceks     -storepass
storepass -storetype jceks -keypass keypass -noprompt

keytool -import -alias sender    -file clientpub.cer  -keystore receiver.jceks   -storepass storepass
-storetype jceks -keypass keypass -noprompt
```

The contents of the key and trust files and their purpose are listed in Table 1.

Table 1. Key store, trust store, and certificate contents

Key and trust store file name	Key or certificate alias	Key or certificate type	Purpose
saml-issuer.jceks	samlissuer	key pair and certificate	SAML issuer uses the private key to sign

			SAML tokens.
validator.jceks	samlissuer	certificate	Receiver uses the certificate to validate SAML token integrity.
sender.jceks	sender	key pair and certificate	Sender uses the private key to sign SAML tokens with important SOAP request message parts.
sender.jceks	receiver	certificate	Sender uses the certificate to verify integrity of SOAP response messages.
receiver.jceks	receiver	key pair and certificate	Receiver uses the private key to sign important SOAP response message parts.
receiver.jceks	sender	certificate	Receiver uses the certificate to verify integrity of SAML tokens and SOAP request messages to meet sender-vouches subject confirmation requirements.

You need to modify the SAMLIssuerConfig.properties configuration file to use saml-issuer.jceks to sign SAML tokens. A sample SAMLIssuerConfig.properties file is provided in the download materials for your reference. You can extract and replace it with the sample configuration file using wsadmin scripts.

Attach policy set and create binding configuration

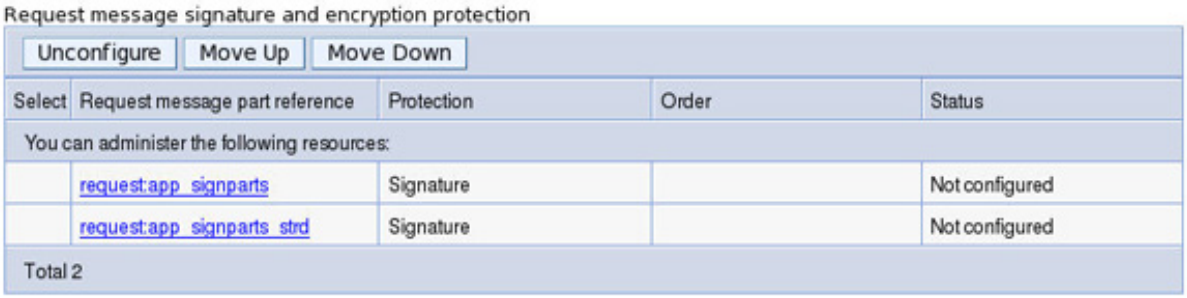
You must use an application-specific binding configuration. You cannot use any general binding configuration because you added an additional list of signed parts (app_signparts_strd) in the policy in order to sign SAML tokens via SecurityTokenReference transformation. When there are two lists of signed parts, the web services runtime and admin utility cannot apply default rules to establish a linkage from the signed part policy elements to the signing keys and other binding configuration elements. Although, in this particular example, the two signed parts (app_signparts and app_signparts_strd) share a common signing key, they use different transformation algorithms: exclusive C14n and SecurityTokenReference transformation, respectively. Hence, you have to use an application-specific binding configuration to explicitly assign each list of a different transformation algorithm.

The key step to configuring the SecurityTokenReference transformation algorithm is

described in this section. (The remaining configuration steps are similar to those outlined in the SAML assertions across WebSphere Application Server security domains article and the Configuring client and provider bindings for the SAML sender-vouches token Information Center article and will not be repeated here.) A completed set of client and provider application specific bindings (SAML20SVSampleClientBinding and SAML20SVSampleProviderBinding) are included in the sample application for your reference; you can simply assign the sample bindings to the sample web services client and provider.

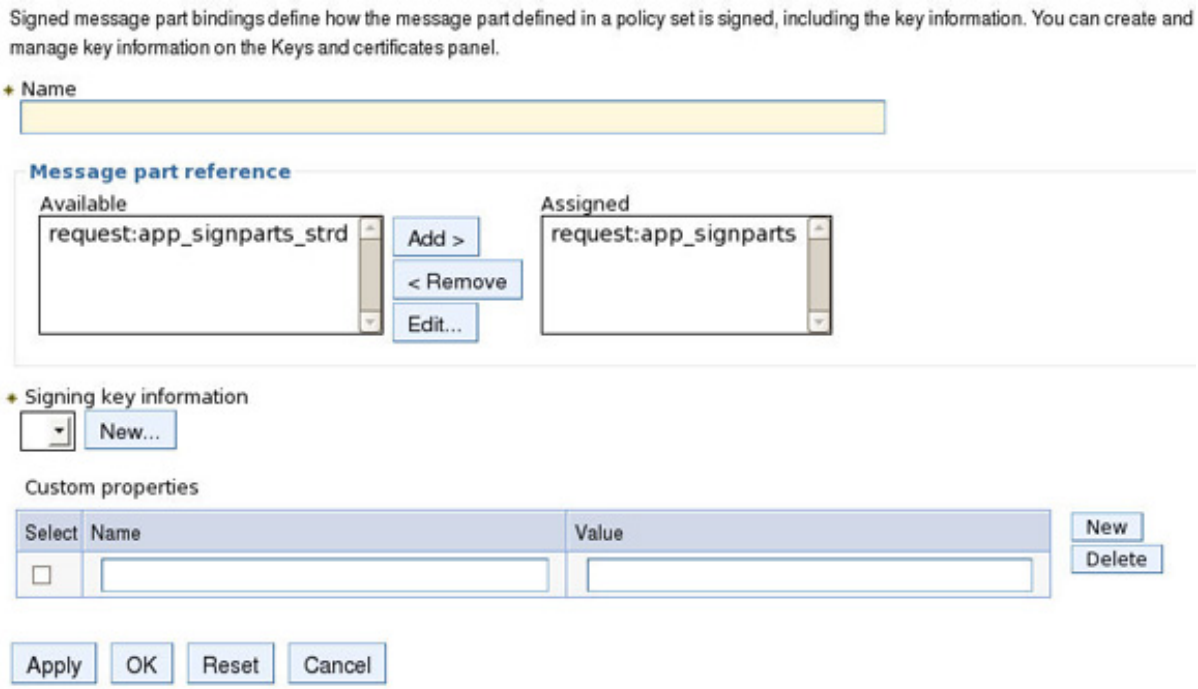
To create an application-specific client binding configuration from scratch, we'll assume that the SAML20SV_Sign_Not_Encrypt policy set is already attached to the sample web services client. When creating a new application-specific binding configuration, you will see a panel showing references to both the signparts and the signparts_strd elements (Figure 5).

Figure 5. Request message signature and encryption protection reference configuration



Clicking either link (**request:app_signparts** in this example) will bring you to the panel shown in Figure 6.

Figure 6. Message signed parts configuration panel



Select **request_app_signparts_strd** and add it to the **Assigned** reference list on the right (Figure 7). Select **request:app_signedparts_strd** and click the **Edit...** button to configure the SecurityTokenReference transformation algorithm (Figure 8).

Figure 7. Add request:app_signparts_strd reference to the list of "Assigned" references

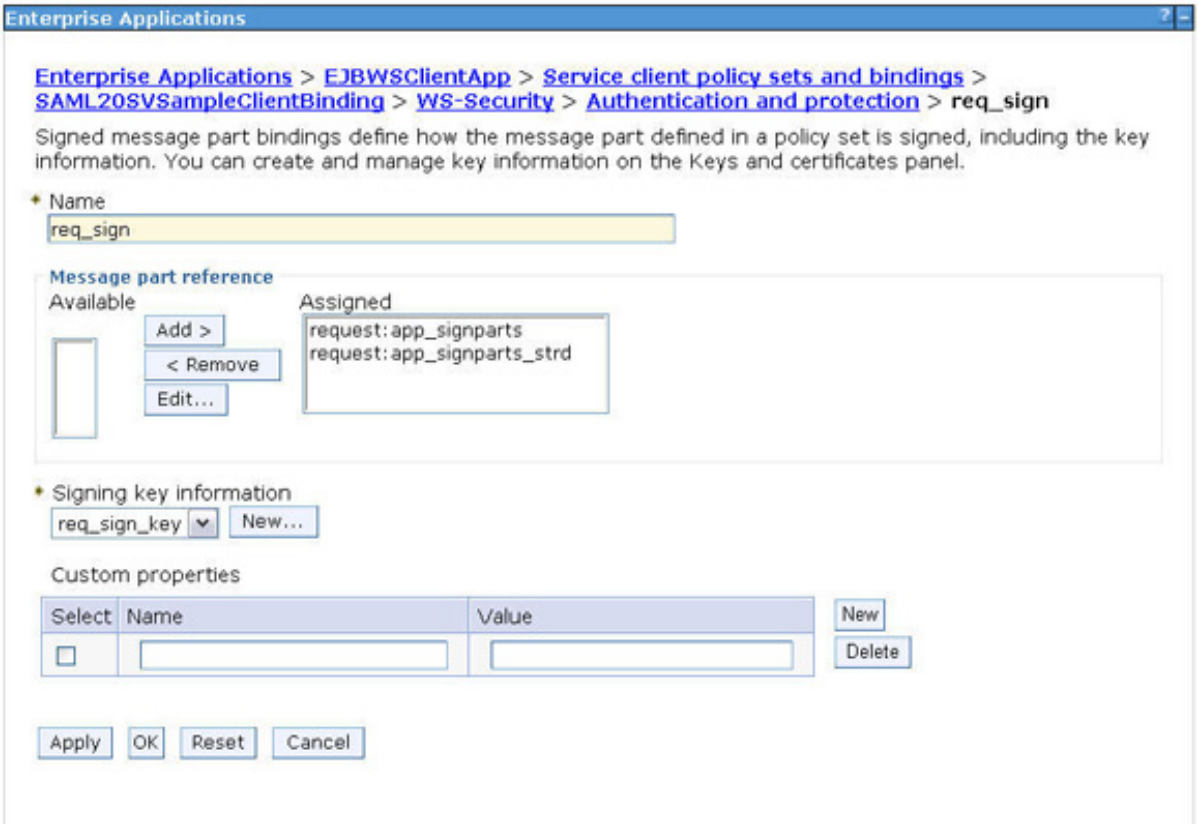


Figure 8. Configure the SecurityTokenReference transformation algorithm

Complete the remaining configuration steps and repeat the steps above to create the Web services provider application-specific binding configuration. Follow the instructions in SAML assertions across WebSphere Application Server security domains to test the sample application.

Next, you will need to specify a client binding configuration to create a SecurityTokenReference element to reference a SAML token. From the Service client policy set and bindings panel, navigate to **WS-Security > Authentication and protection > Authentication tokens**. Select and open the SAML token you want to sign for editing and add the custom property `com.ibm.ws.wssecurity.createSTR =true` or `signToken =true`.

External security token services consideration

The application-specific Web services client and provider bindings provided in the sample application are configured for using self-issued SAML tokens. You can modify the Web services client binding configuration to retrieve SAML tokens from an external Security Token Services (STS).

Next, you will specify the STS URL, web services policy set, and binding configuration for sending WS-Trust request messages.

Another configuration parameter is the keyType property, which specifies the desired subject confirmation method for requested SAML tokens. Intuitively, you might set keyType to **sender-vouches**. It is interesting to note that the three KeyType values that are defined by the WS-Trust 1.3 OASIS Standard 19 March 2007 specification -- PublicKey, SymmetricKey, and Bearer -- are different from the SAML token subject confirmation methods. WS-Trust 1.3 PublicKey and SymmetricKey KeyTypes indicate the type of key required in requested security tokens. The WS-Trust 1.3 Bearer KeyType basically means that requested tokens do not require keys, or, in other words, security token holders are not required to prove they have knowledge of the keys. When requesting SAML tokens with the sender-vouches subject confirmation method, whether it is required to specify the WS-Trust 1.3 Bearer KeyType is determined by the particular STS you have. You will likely need to configure requiring the sender-vouches subject confirmation method on the STS service side.

If you are using IBM TivoliM® Federated Identity Manager V6.2.1, you can select sender-vouches from a drop down menu at the bottom of the SAML issue configuration panel. If you are using Tivoli Federated Identity Manager V6.2.0, you will need to set the STSUU attribute shown in Listing 2 to your mapping rule.

Listing 2

```
<stsuuser:Attribute name="SamlSubjectConfirmationMethod"
type="urn:oasis:names:tc:SAML:2.0:assertion">
        <stsuuser:Value>urn:oasis:names:tc:SAML:2.0:cm:sender-vouches</stsuuser:Value>
</stsuuser:Attribute>
```

Conclusion

This article discussed how you can configure and use SAML tokens in web services in IBM WebSphere Application Server (V7.0 Fix Pack 9 and later), demonstrating how to construct a policy that requires a sender to vouch for a SAML token using a digital signature at the message level for integrity, and to use SSL for confidentiality. The article also explained why you must use application specific bindings and not general bindings. The sample application code and application specific bindings that are pre-configured in the included EARs use self-issued SAML tokens. Using the pre-configured policy set, the sample application with pre-configured bindings, a script to generate sample keys and trust files, and the sample SAMLIssuerConfig.properties file, you can deploy and run the examples in a few simple steps. The sample policy and application-specific binding configuration also serve as useful references for customizing for your particular use casea. You can modify the web services client code and binding to request SAML tokens from an external STS.

Downloads

Description	Name	Size	Download method
Code sample	svsample.zip	46 KB	HTTP

Information about download methods

Resources

Learn

- WebSphere Application Server V7 Information Center
- SAML assertions across WebSphere Application Server security domains
- Configuring client and provider bindings for the SAML bearer token
- Signing SAML tokens at the message level
- Tivoli Federated Identity Manager Information Center
- Configuring policy sets and bindings to communicate with STS
- WS-Trust 1.3 OASIS Standard 19 March 2007
- Tivoli Federated Identity Manager Information Center
- Introduction to Security Assertion Markup Language (SAML) and SAML Support in IBM WebSphere Application Server Version 7.0 Fix Pack 7 (PDF)
- SAML concepts
- Setting up the SAML configuration
- IBM developerWorks WebSphere

Get products and technologies

- Trial download: Get IBM WebSphere Application Server V8 (trial version)
- No-charge offering: Download WebSphere Application Server for Developers

About the authors

Chunlong Liang
Chunlong Liang is a developer of Web services security on the WebSphere platform. Chunlong has been in WebSphere development for over 9 years and has developed many security features for the WebSphere platform. Prior to joining WebSphere team, Chunlong was an actuarial analyst and programmer for 6 years, and was a statistician for 3 years.

Ching-Yun (C.Y.) Chao, Ph.D.
Ching-Yun (C.Y.) Chao was a WebSphere Web services security architect while

writing this article. C.Y. is currently a security architect on the IBM Workload Deployer product. C.Y. has been in WebSphere Application Server development for over 10 years and has designed and worked on many WebSphere security features. Prior to joining WebSphere development, C.Y. was lead architect and developer of the IBM PC server highly available system clustering project. He was awarded IBM Master Inventor in 2006 and has many patents in the area of security and high availability system clustering. C.Y. received his Ph.D. in Electrical Engineering from Northwestern University.