Bare Timestamp Signatures with WS-Security

Paul Glezen, IBM

Abstract

This document is a member of the Bare Series of WAS topics distributed in both stand-alone and in collection form. The latest renderings and source are available on GitHub at http://pglezen.github.io/was-config.

Table of Contents

1. Concepts

WS-Policy is a very general framework for describing “non-functional” requirements for web services. WS-Policy applies to many aspects of web services such as addressing, reliability, notifications, and most commonly, security. The power of this concept lies in the potential to apply these policies to applications without requiring changes to the application code. WS-Policy addresses a wide variety of requirements in a vendor neutral way. This has obvious benefits; but also leads to its commonly cited drawback: its complexity and abstraction levels often discourage its adoption.

Since this document only addresses WS-Security, it often uses the terms WS-Policy and WS-Security interchangeably.

1.1. What versus How

WS-Security documents, in order to be general enough to apply in a platform-neutral way, are restricted to *what* is to be done. Examples of “what” include “encrypt this field” or “add a time stamp and sign it.” But when it comes to implementing these mandates, there is still the question of *how*. The most common examples of "how" are “how is an encryption key to be procured” or “how is a certificate verified to be trustworthy” or "how is a certificate to be specified in a SOAP payload". In WebSphere Application Server (WAS), this separation of concerns is implemented with policy sets and policy set bindings.

1.2. Protection versus Authentication

The WS-Security configuration panels often distinguish between protection tokens and authentication tokens. In this context, protection refers to protecting the message from eavesdropping and/or corruption (whether accidental or malicious). Eavesdropping is addressed by encryption. Message corruption is addressed by signatures. In this document these tokens are asymmetric X.509 binary security tokens.

Authentication tokens contain identity information. This identity information may be encrypted and/or signed. It conveys to the provider the identity on behalf of whom the request is executing.

1.3. Callback Handlers

The WAS security runtime uses a framework to apply the same processing pattern to similar scenarios. But while the high-level processing is the same, details about the input can vary widely. A common example is the authentication process. One can authenticate via a user ID and password, a SAML assertion, an LTPA token, an SSL certificate, or others. Each of these types of authentication have a callback handler class associated with it. When specifying which authentication to use, the callback handler class is specified for processing and the callback class is populated with the input. Each callback handler expects input in the form of its associated callback.

This concept appears often in WS-Security configuration panels. When confronted with a callback handler configuration, it's just a way to pass data into the framework process. For WS-Security panels, this is used to convey the key and trust store information.

1.4. Must Understand

WS-Security headers support a `mustUnderstand` attribute. If this attribute is set to "`1`", it means all intermediaries must be able to process the security information. If a signature is included, the intermediaries must be able to verify it. If intermediaries are not required to process the security token, then this attribute should have a value of "`0`" or the attribute must be absent (zero is assumed when absent).

By default, a WS-Security binding configuration will add `mustUnderstand="1"`. To override this default requires a custom property set on the binding. This location for this custom property in the WAS admin console is Services → General client policy set bindings → (binding) → WS-Security → Custom properties.

The mustUnderstand custom property is defined under Outbound Custom Properties and is named

```
com.ibm.wsspi.wssecurity.config.request.setMustUnderstand
```

Another useful custom property for the client bindings is defined under Inbound Custom Properties. It determines whether the "consumer" requires a timestamp on the response. Its name is

```
com.ibm.wsspi.wssecurity.consumer.timestampRequired
```

Since this property is defined on the inbound section for a client, it is referring to a response (from the provider). By default this property value is 1; meaning the provider must sign sign a return time stamp or the response will be rejected. This restriction may be relaxed by setting this value to 0.

1.5. Application Code Requirements

WS-Security policies can only be applied to Java bindings implemented with JAX-WS. They cannot be applied to JAX-RPC bindings. Services implemented with JAX-WS bindings will appear under Services → Service providers. Service provides are always considered to be managed.

JAX-WS client bindings can be either managed or non-managed, depending on how they are packaged and initialized. A JAX-WS client is only managed if it is retrieved via a local JNDI reference. Obtaining a JAX- WS service reference through either direct instantiation or through global JNDI look-up are not managed. Non-managed JAX-WS clients cannot have WS-Security policy applied to them because they bypass the required WAS runtime hooks.

Managed JAX-WS client have to be declared through either the `@WebServiceRef` annotation or the `service-ref` entry in a deployment descriptor. The `@WebServiceRef` annotation is the more convenient option. But scanning for this annotation is restricted to EJB classes, Servlet classes, JAX-WS handler classes, and some service endpoint implementations. Moreover, it is not always practical to add `@WebServiceRef` annotations to every class in which a service client is required.

Web application projects can have a `service-ref` element to their `web.xml` deployment descriptor. The following is how such an entry would look for the CC example consumer web application.

```
<service-ref>
  <service-ref-name>service/CCService</service-ref-name>
  <service-interface>org.acme.cc.jaxws.CCService</service-interface>
</service-ref>
```

This works when the JAX-WS client binding classes are packaged within the WAR project, either directly compiled to the classes directory or included as a JAR file in `WEB-INF/lib`. But if the JAX-WS client bindings are packaged as a utility JAR included within the EAR file, an extra element is needed in the deployment descriptor.

```
 <service-ref>
   <service-ref-name>service/CCService</service-ref-name>
   <service-interface>org.acme.cc.jaxws.CCService</service-interface>
   <service-qname xmlns:pfx="urn:issw:bare:wssec:cc:query">pfx:CCService</service-
qname>
 </service-ref>
```

The `service-qname` element allows the local reference look-up to successfully determine the QName for the client binding service class. The `xmlns:pfx` attribute is a namespace declaration. The value should be the namespace declared for your service element of the relevant WSDL document.

2. Implementation

The signatures described here are considered protection tokens for the purpose of configuration within the WAS admin console. In our app-to-app scenario, the signature is doubling as an authentication mechanism of sorts, since only the possessor of the private key could have signed the message. But don't let this secondary usage misguide you when working through the policy set binding panels. The signature scenario exclusively deals with protection tokens, not authentication tokens.

2.1. Key Stores

2.1.1. Service Consumer

The service consumer requires a key store containing a private/public key pair that identifies the service consumer application. The public key will be extracted so that it may be provided to the consumer for the purpose establishing trust.

This key store, key alias, and password will be configured in the general client policy bindings as a reference to a managed key store. The scripting burden of the WS-Security configuration would be eased if the key store and key alias names could be consistent among environments.

2.1.2. Service Provider

The service provider requires a trust store containing the signer certificates that the provider is willing to accept. For this WS-Security configuration, this amounts to the CCConsumer public certificate.

As with the service consumer case, choosing a consistent name for trust store simplifies the scripting of the service provider policy set bindings.

2.2. The Policy Set

A policy set is a set of WS-Policy documents. As mentioned in Section 1.1, a policy document addresses what is to be done or enforced. Since both ends of a consumer-provider channel must agree on this, the policy document is usually shared between both parties.

In the present case, the policy document will specify the signing of a time stamp. Later sections address policy set bindings that configure role and environment specific configurations, mostly to do with key stores.

2.2.1. Policy Set Creation

The following steps show how to create a policy set that specifies

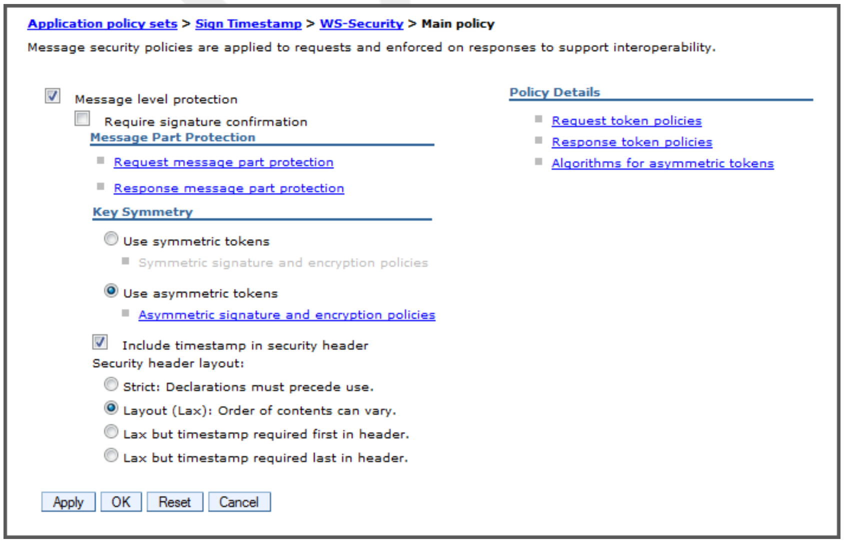
- A timestamp to be added to the WS-Security header
- the timestamp to be signed

This policy set is simple enough to create from scratch.

1. In the WAS admin console, navigate to Services → Policy sets → Application Policy sets.
2. Click the New button.
3. For Name, enter `Sign Timestamp`.
4. For Description, enter `Add a timestamp to the SOAP security header and sign it.`
5. In the Policies section, click the New button and select WS-Security. This will cause a WS-Security link to appear in the list.
6. Click the WS-Security link.
7. Click the main policy link.

   This panel holds all settings for the WS-Security policy. The Message level protection box should already be checked. In the present case, all we wish to do is add a timestamp and sign it. We will remove the other items.

Figure 1. Main Policy Panel



Application policy sets > Sign Timestamp > WS-Security > Main policy

Message security policies are applied to requests and enforced on responses to support interoperability.

☑ Message level protection

□ Require signature confirmation

Message Part Protection

- Request message part protection
- Response message part protection

Key Symmetry

○ Use symmetric tokens

- Symmetric signature and encryption policies

● Use asymmetric tokens

- Asymmetric signature and encryption policies

☑ Include timestamp in security header

Security header layout:

○ Strict: Declarations must precede use.

● Layout (Lax): Order of contents can vary.

○ Lax but timestamp required first in header.

○ Lax but timestamp required last in header.

Policy Details

- Request token policies
- Response token policies
- Algorithms for asymmetric tokens

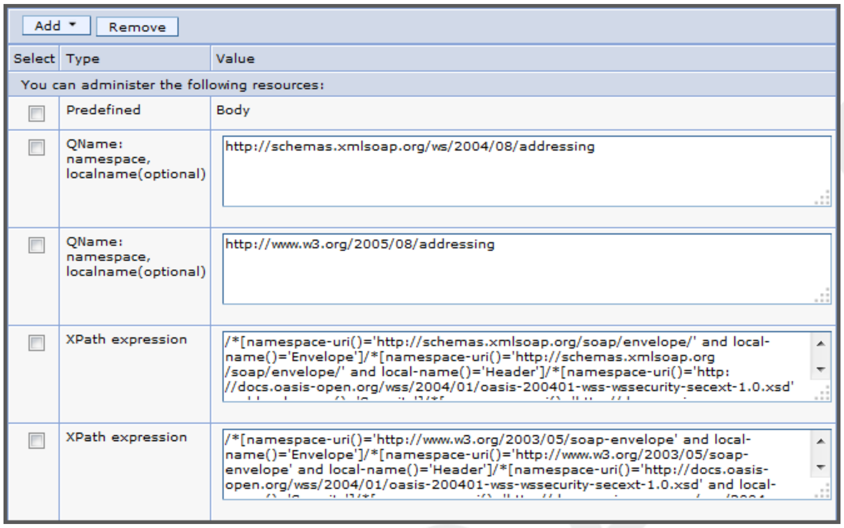
8. The checkbox Include timestamp in security header should already be checked.

9. Click the Request message part protection link.

10. Under Encrypted parts, select app_encparts and click the Delete. We remove the encrypted parts because we will not be encrypting the payload at the message level.

11. Under Signed parts, select app_signparts and click Edit.

Figure 2. Message parts to sign



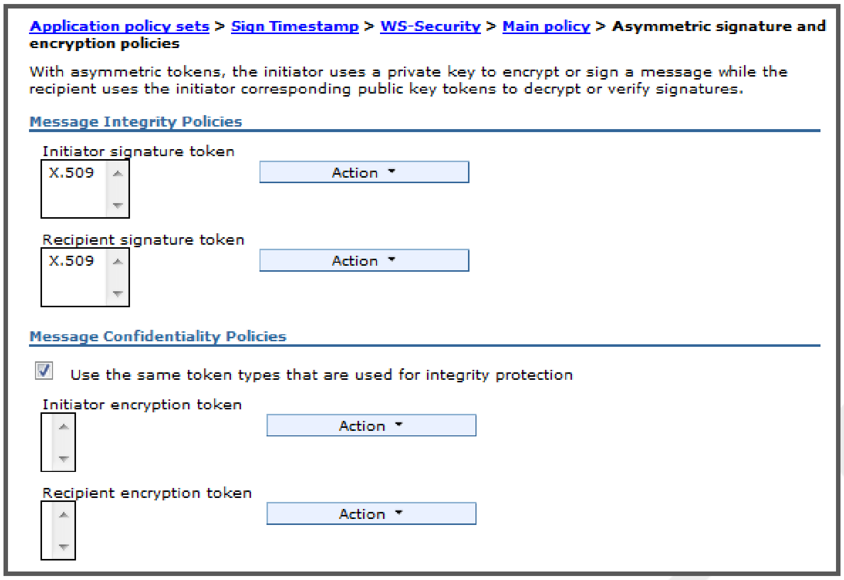
Select	Type	Value
You can administer the following resources:		
□	Predefined	Body
□	QName: namespace, localname(optional)	http://schemas.xmlsoap.org/ws/2004/08/addressing
□	QName: namespace, localname(optional)	http://www.w3.org/2005/08/addressing
□	XPath expression	/*[namespace-uri()='http://schemas.xmlsoap.org/soap/envelope/' and local-name()='Envelope']/*[namespace-uri()='http://schemas.xmlsoap.org/soap/envelope/' and local-name()='Header']/*[namespace-uri()='http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd'
□	XPath expression	/*[namespace-uri()='http://www.w3.org/2003/05/soap-envelope' and local-name()='Envelope']/*[namespace-uri()='http://www.w3.org/2003/05/soap-envelope' and local-name()='Header']/*[namespace-uri()='http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd' and local-

12. By default there are five signed parts specified: three general parts and two timestamp parts at the bottom. Remove the top three parts compresed of the Body and two WS-Addressing QName parts. You should be left with two XPath expression parts for the timestamp: one for SOAP 1.1 and one for SOAP 1.2.

13. Click OK for the signed parts and Done for the request message part protection.

14. Click Asymmetric signature and encryption policies. Verify that X.509 is chosen for the Message Integrity Policy section as shown in Figure 3.

Figure 3. Asymmetric signature and encryption policies



Application policy sets > Sign Timestamp > WS-Security > Main policy > **Asymmetric signature and encryption policies**

With asymmetric tokens, the initiator uses a private key to encrypt or sign a message while the recipient uses the initiator corresponding public key tokens to decrypt or verify signatures.

Message Integrity Policies

Initiator signature token: X.509

Recipient signature token: X.509

Message Confidentiality Policies

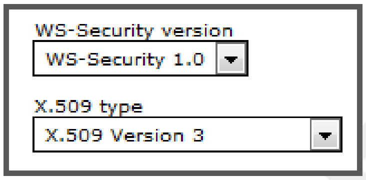
☑ Use the same token types that are used for integrity protection

Initiator encryption token

Recipient encryption token

If they are not present, click the Action drop-down menu and select Add X.509 Type. Choose the version shown in Figure 4.

Figure 4. Add X.509 Type



WS-Security version: WS-Security 1.0

X.509 type: X.509 Version 3

This completes the specification of the request message protection.

- The default algorithms are fine; but may be adjusted. If you choose this section, only adjust the Algorithm suite. Do not change the Canonicalization algorithm or the XPath version unless you know what you're doing.
- There are no request or response tokens for this configuration.

2.2.2. Policy Set Export and Import

A policy set is usually shared between consumer and provider instances as well as among different environments. It is usually created once and exported; then imported wherever else it is needed.

To export the `Sign Timestamp` policy, navigate to Services → Policy sets → Application policy sets. Check the box next to Sign Timestamp and click the Export button at the top. This will reveal a Sign Timestamp.zip link. Click this link to download the policy set export.

Tip

Because "Sign Timestamp" contains a space in the name for readability, the admin console will supply a default file name of `Sign Timestamp.zip`. Scripting will be simplified if this space is removed from the file name. This space will still be preserved for the policy name after import into other WAS cells.

Here are the steps to import a policy set.

1. In the admin console, navigate to Services → Policy sets → Application policy sets.
2. Click the Import button and select From Selected location.
3. Click the Browse button and select the policy set archive.
4. Click OK.

The imported policy set should now appear in the list of application policy sets.

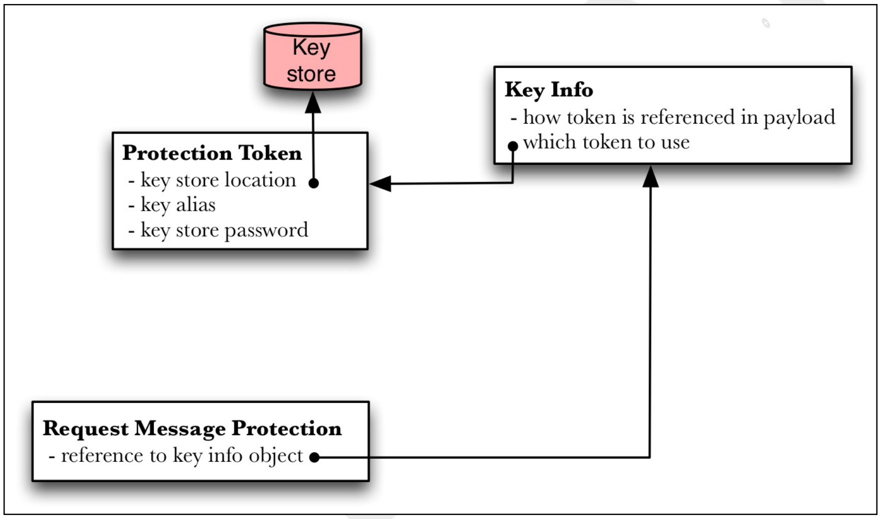
2.3. Client Policy Set Bindings

The client policy set bindings for the CCConsumerApp application specifies how the consumer application will sign the SOAP payload elements required by the policy set. This amounts to specifying

- a key store along with the alias of the key used to sign the request,
- how the corresponding certificate is to be identified.

These concepts are illustrated below in Figure 5 as objects in a WAS configuration.

Figure 5. Client binding concepts



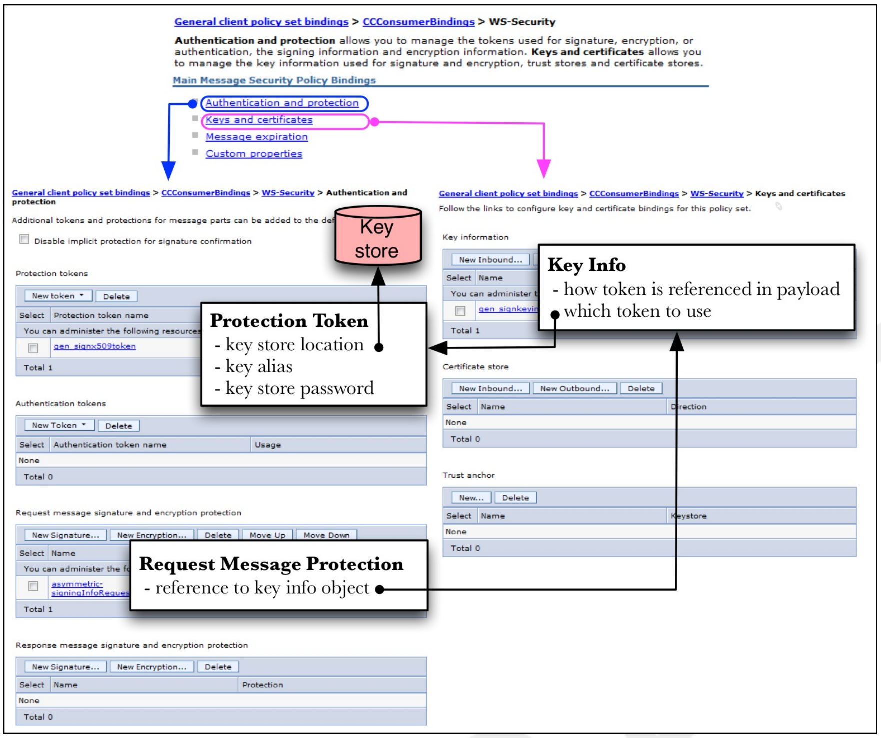
The *protection token* object references a key store and contains properties for specifying the alias of the relevant key in the key store along with a key store password. It represents a private key in the configuration.

The *key info* object determines how information about the key will be added to the payload. Examples include referring to the certificate's serial number, the certificate's SHA1 thumbprint, or just including the entire certificate as base64-encoded text. In addition to specifying how the key will be referenced in the payload, it also includes a reference to the key itself through through a reference to the associated protection token object.

The *request message protection* object binds the protection token and key info into a single configuration. Only when key info objects are referenced by a request message protection object are they "activated" by the binding.

The objects in Figure 5 are configured in the WAS admin console by navigating to Services → Policy sets → General client policy set bindings. After selecting from the particular binding, click the WS-Security link. unfortunately not all objects are available from the same panel. Figure 6 gathers together the three screens needed for this configuration with the conceptual objects from Figure 5 superimosed on them for reference.

Figure 6. Client binding concepts overlayed on screen shot



General client policy set bindings > CCConsumerBindings > WS-Security

Authentication and protection allows you to manage the tokens used for signature, encryption, or authentication, the signing information and encryption information. **Keys and certificates** allows you to manage the key information used for signature and encryption, trust stores and certificate stores.

Main Message Security Policy Bindings

- Authentication and protection
- Keys and certificates
- Message expiration
- Custom properties

General client policy set bindings > CCConsumerBindings > WS-Security > Authentication and protection

Additional tokens and protections for message parts can be added to the def

□ Disable implicit protection for signature confirmation

Protection tokens

Select	Protection token name
You can administer the following resources	
□	gen_signx509token
Total 1	

Authentication tokens

Select	Authentication token name	Usage
None		
Total 0		

Request message signature and encryption protection

Select	Name
You can administer the fo	
□	asymmetric-signingInfoReques
Total 1	

Response message signature and encryption protection

Select	Name	Protection
None		
Total 0		

General client policy set bindings > CCConsumerBindings > WS-Security > Keys and certificates

Follow the links to configure key and certificate bindings for this policy set.

Key information

Select	Name
You can administer t	
□	gen_signkeyin
Total 1	

Certificate store

Select	Name	Direction
None		
Total 0		

Trust anchor

Select	Name	Keystore
None		
Total 0		

The WS-Security panel is shown at the top of Figure 6. The Authentication and protection link navigates to the panel on the left. The Keys and certificates link navigates to the panel on the right. The direction of the arrows in Figure 6 convey the direction of references. So the objects must be configured in the reverse order of the arrows so that the references may be resolved.

1. Configure a managed key store as described in Section 2.1.1.
2. Create a protection token object referencing the key store.
3. Create a key info object referencing the protection token.
4. Create a request message protection object referencing the key info object.
5. Add custom properties.

The panel for the last item is not shown in Figure 6. But the Custom properties link to its panel is shown near the top.