**<http://static.springsource.org/spring-ws/site/reference/html/security.html#security-wss4j-security-interceptor>**

[**http://www.w3.org/TR/ws-policy/**](http://www.w3.org/TR/ws-policy/)

**7.1. Introduction**

This chapter explains how to add WS-Security aspects to your Web services. We will focus on the three different areas of WS-Security, namely:

**Authentication.** This is the process of determining whether a *principal* is who they claim to be. In this context, a "principal" generally means a user, device or some other system which can perform an action in your application.

**Digital signatures.** The digital signature of a message is a piece of information based on both the document and the signer's private key. It is created through the use of a hash function and a private signing function (encrypting with the signer's private key).

**Encryption and Decryption.** *Encryption* is the process of transforming data into a form that is impossible to read without the appropriate key. It is mainly used to keep information hidden from anyone for whom it is not intended. *Decryption* is the reverse of encryption; it is the process of transforming of encrypted data back into an readable form.

All of these three areas are implemented using the XwsSecurityInterceptor or Wss4jSecurityInterceptor, which we will describe in [Section 7.2, “ XwsSecurityInterceptor ”](http://static.springsource.org/spring-ws/site/reference/html/security.html#security-xws-security-interceptor) and [Section 7.3, “ Wss4jSecurityInterceptor ”](http://static.springsource.org/spring-ws/site/reference/html/security.html#security-wss4j-security-interceptor), respectively

## 7.3.  Wss4jSecurityInterceptor

The Wss4jSecurityInterceptor is an EndpointInterceptor (see[Section 5.5.2, “Intercepting requests - the EndpointInterceptor interface”](http://static.springsource.org/spring-ws/site/reference/html/server.html#server-endpoint-interceptor)) that is based on [Apache's WSS4J](http://ws.apache.org/wss4j/).

WSS4J implements the following standards:

* OASIS Web Serives Security: SOAP Message Security 1.0 Standard 200401, March 2004
* Username Token profile V1.0
* X.509 Token Profile V1.0

This inteceptor supports messages created by the AxiomSoapMessageFactory and the SaajSoapMessageFactory.

### 7.3.1. Configuring Wss4jSecurityInterceptor

WSS4J uses no external configuration file; the interceptor is entirely configured by properties. The validation and securement actions executed by this interceptor are specified via validationActions and securementActions properties, respectively. Actions are passed as a space-separated strings. Here is an example configuration:

<bean class="org.springframework.ws.soap.security.wss4j.Wss4jSecurityInterceptor">

<property name="validationActions" value="UsernameToken Encrypt"/>

...

<property name="securementActions" value="Encrypt"/>

...

</bean>

Validation actions are:

| **Validation action** | **Description** |
| --- | --- |
| UsernameToken | Validates username token |
| Timestamp | Validates the timestamp |
| Encrypt | Decrypts the message |
| Signature | Validates the signature |
| NoSecurity | No action performed |

Securement actions are:

| **Securement action** | **Description** |
| --- | --- |
| UsernameToken | Adds a username token |
| UsernameTokenSignature | Adds a username token and a signature username token secret key |
| Timestamp | Adds a timestamp |
| Encrypt | Encrypts the response |
| Signature | Signs the response |
| NoSecurity | No action performed |

The order of the actions is significant and is enforced by the interceptor. The interceptor will reject an incoming SOAP message if its security actions were performed in a different order than the one specified byvalidationActions.

### 7.3.3. Authentication

#### 7.3.3.1. Validating Username Token

Spring-WS provides a set of callback handlers to integrate with Spring Security. Additionally, a simple callback handler SimplePasswordValidationCallbackHandler is provided to configure users and passwords with an in-memory Properties object.

Callback handlers are configured via Wss4jSecurityInterceptor's validationCallbackHandler property.

##### 7.3.3.1.1. SimplePasswordValidationCallbackHandler

SimplePasswordValidationCallbackHandler validates plain text and digest username tokens against an in-memory Properties object. It is configured as follows:

<bean id="callbackHandler"

class="org.springframework.ws.soap.security.wss4j.callback.SimplePasswordValidationCallbackHandler">

<property name="users">

<props>

<prop key="Bert">Ernie</prop>

</props>

</property>

</bean>

##### 7.3.3.1.2. SpringSecurityPasswordValidationCallbackHandler

The SpringSecurityPasswordValidationCallbackHandler validates plain text and digest passwords using a Spring Security UserDetailService to operate. It uses this service to retrieve the (digest of ) the password of the user specified in the token. The (digest of) the password contained in this details object is then compared with the digest in the message. If they are equal, the user has successfully authenticated, and a UsernamePasswordAuthenticationToken is stored in theSecurityContextHolder. You can set the service using the userDetailsService. Additionally, you can set a userCache property, to cache loaded user details.

<beans>

<bean class="org.springframework.ws.soap.security.wss4j.callback.SpringDigestPasswordValidationCallbackHandler">

<property name="userDetailsService" ref="userDetailsService"/>

</bean>

<bean id="userDetailsService" class="com.mycompany.app.dao.UserDetailService" />

...

</beans>

#### 7.3.3.2. Adding Username Token

Adding a username token to an outgoing message is as simple as adding UsernameToken to the securementActions property of the Wss4jSecurityInterceptor and specifying securementUsername andsecurementPassword.

The password type can be set via the securementPasswordType property. Possible values are PasswordText for plain text passwords or PasswordDigest for digest passwords, which is the default.

The following example generates a username token with a digest password:

<bean class="org.springframework.ws.soap.security.wss4j.Wss4jSecurityInterceptor">

<property name="securementActions" value="UsernameToken"/>

<property name="securementUsername" value="Ernie"/>

<property name="securementPassword" value="Bert"/>

</bean>

If plain text password type is chosen, it is possible to instruct the interceptor to add Nonce and/or Created elements using the securementUsernameTokenElements property. The value must be a list containing the desired elements' names separated by spaces (case sensitive).

The next example generates a username token with a plain text password, a Nonce and a Created element:

<bean class="org.springframework.ws.soap.security.wss4j.Wss4jSecurityInterceptor">

<property name="securementActions" value="UsernameToken"/>

<property name="securementUsername" value="Ernie"/>

<property name="securementPassword" value="Bert"/>

<property name="securementPasswordType" value="PasswordText"/>

<property name="securementUsernameTokenElements" value="Nonce Created"/>

</bean>

the client sets a SOAP header in every request that contains a username and password. The password is sent in the header in plain text. To be securing all HTTP requests and responses should be SSL encrypted.

Unfortunately there is no WS-Policy support in Spring-WS. You have to write your own WSDL if you want to add WS-Policy.

My original question was how to add WS-Security to the wsdl file generated by CXF. I thought it was possible by using the interceptors, but the interceptors are only for handling requests/responses.