# Signing Procedures 1.6 (Handtekening procedures)

Ruimtelijke ordening

**DigitaleBouwaanvraag**

**Signing Procedures: How to implement security for the DBA web services using JAVA and .NET**

Versie 1.6

Vlaamse overheid

Vlaanderen In Actie

## Table of Contents

# Introduction

## Purpose

This document describes step-by-step procedures for implementing a web service client for accessing MAGDA web services protected with WS-S. Main focus is placed on setting and configuring options and parameters to fulfill security requirements to connect to the DBA web services as an example.
As a test example we will use CXF JAX-WS client implementation of the Java web service client and a Spring context configuration. Also for the Windows .Net environment code examples are given.

# Glossary

| | |
|---|---|
| CA | Certification Authority |
| CXF | Apache's Open-Source Services Framework (http://cxf.apache.org/) |
| WS-S | Web Services Security (http://en.wikipedia.org/wiki/WS-Security) |

# Environment Configuration

## Configuring keystores

Configuring keystore will provide measures to enable

- transport level security
  - SSL client authentication
  - SSL server trust
- web service security – ability to identify the client and sign the web service request

## Set up the client keystore

The keystore has to provide a client's identity in form of a keypair and a client's certificate. Steps to create and certify a client keypair are described in the document "Certificaten-Aanvraag". For this exercise we will assume there is a keystore test_keystore.jks with a client certificate with alias dba. The same keystore and key will be used for message signing and for ensuring SSL/TLS communication.

## Set up the client truststore

### Security consideration

By default the SSL transport layer will apply "indirect trust" to decide whether to trust another party or not. This means that if any of certificates in the certification chain is trusted, the party's certificate is trusted.
Usually indirect trust is sufficient as issuers listed in the Java default truststore are trusted. The RWO server certificate is signed by the "GlobalSign Root CA" certificate, which is globally trusted.
Globally trusted means, that the certificate is already present in the default Java truststore. If the client is satisfied with this level of trust, the next step (importing RWO certificate into a truststore) is optional.
There is a potential security issue when "chain trust" is applicable. Although the certificate used for signature validation might be "trusted" in a general sense (via a trusted issuing certificate), the client may not wish to accept every certificate issued by the trusted certificate. This is particularly relevant if any certificate in the signature truststore is (or can potentially be) used to issue a wide range of certificates. There are means described later to control set of certificates to trust.

### Trusting the server certificate

As described in the previous chapter, the server certificate is trusted by default through trusting the root GlobalSign certificate. The root certificate is located in the Java "cacerts" file. The "cacerts" file is a keystore with root certificates which are trusted by default. The cacerts ke ystore is located in the location $JAVA_HOME/lib/security/cacerts.

# Web service Client

In this chapter we will build and configure an example web service client invoking the GeefDossierStuk web service. In the example we will use generated web service client classes to invoke the web service.

## Security requirements

The security requirements for the client are:

| Requirement | Description |
|---|---|
| SSL client authentication | In order to establish SSL/TLS connection the client must provide its authentication certificate. It must be the same certificate that is used to sign the request message. |
| Authentication by a binary security token (X.509 certificate) | The client has to provide its public certificate in form of the X.509 certificate |
| Message signature | The web service request message must be signed by the client's authentication certificate. |

*Table 1: Client Security requirements*
The options for the Signature setting are:

| WS-Security option | Value |
|---|---|
| Must understand | True |
| Key identifier type | Binary security token |
| User single certificate | True |
| Signature algorithm | {+}http://www.w3.org/2000/09/xmldsig#rsa-sha1+ |
| Signature canonicalization | {+}http://www.w3.org/2001/10/xml-exc-c14n+# |
| Digest algorithm | {+}http://www.w3.org/2000/09/xmldsig#sha1+ |

*Table 2: Message signature options*
There are two levels of security in place – message signing and SSL/TLS. For each level it is technically possible to use separate certificates, keystores and configuration. **We will use the same keystore containing the client certificate**. It is enforced (checked) that the SSL client certificate is used for message signing.

### Java Web Service Framework

As a service framework we will use the Apache CXF framework and to enable security features we will use the Apache WSS4J framework. We chose the frameworks for their maturity, usability and their wide acceptance.
Apache CXF is an open source services framework. CXF helps you build and develop services using frontend programming APIs, like JAX-WS and JAX-RS.
More information can be found on {+}http://cxf.apache.org/+
The Apache WSS4J™ project provides a Java implementation of the primary security standards for Web Services, namely the OASIS Web Services Security (WS-Security) specifications from the OASIS Web Services Security TC.
More information can be found on {+}http://ws.apache.org/wss4j/+

## Generate client classes

In this chapter we will describe how to generate web service client classes.
Service definitions are publicly available at the endpoints:
{+}https://oefen.ruimtevlaanderen.be/cxf/magdaGeefDossierService?wsdl+ {+}https://oefen.ruimtevlaanderen.be/cxf/magdaZoekDossierService?wsdl+ {+}https://oefen.ruimtevlaanderen.be/cxf/magdaBewaarDossierService?wsdl+
To generate Java or .NET client we recommended to use the **wsimport** utility:
wsimport -d generated -s src -keep {+}https://oefen.ruimtevlaanderen.be/cxf/magdaGeefDossierService?wsdl+
The **wsimport** documentation can be found at :
{+}http://docs.oracle.com/javase/6/docs/technotes/tools/share/wsimport.html+

*Please note that any reasonable IDE (Integrated Development Environment) or build frameworks already contain ways to build a web service client.*

## Configuration and Invoking Web Service with Java code

For configuration and invoking web service we will use **CXF** framework with the **WSS4J** extension.

## Parameters and constants

### Runtime setting to validate server certificate

In order to validate server certificate, it is possible to leverage OCSP and CRL protocol. According to the Java PKI Programming guide the CRL validation is enabled by default when building a certificate path. To enable OCSP protocol, the most straightforward way is to enable OCSP globally for whole JVM by setting ocsp security property (ocsp.enable=true).
These properties may be set either statically in the Java runtime's **$JAVA_HOME/jre/lib/security/java.security** file or dynamically using the java.security.Security. setProperty() method.
See: http://docs.oracle.com/javase/7/docs/technotes/guides/security/certpath/CertPathProgGuide.html
Conclusion:
In order to enable OCSP and CRL, following properties need to be set:
System.setProperty("com.sun.security.enableCRLDP", "true");
System.setProperty("com.sun.net.ssl.checkRevocation", "true");
java.security.Security. setProperty("ocsp.enable","true");
Note:

- According to the programming guide, if OCSP checking is enabled, CRL is used as fallback method. We recommend to enable the properties on a global (JVM) level.
- com.sun.* properties may are Sun JVM specific (may not be supported on JVM from other vendors)

### Example source code

In the example we will use several constants and parameters we can define upfront. This preparation is done solely for demonstration purposes. **A real client application should use the following settings as configurable application or environmental parameters**.
Parameter values are related to the client certificate attributes. The relations between certificate attributes and provided values are described in the document "**Certificaten-Inhoud**".

```
/**
* keystore with a client certificate
* the client certificate will be used for message signing and for SSL
* client authentication
*/
private static final String KEYSTORE = "/projects/DBA/cert/test_keystore.jks";
/**
* keystore and truststore type
*/
private static final String STORETYPE = "JKS";
/**
* truststore contains RWO server certificate
* to validate certificate we need to specify a truststore,
* we will use the default JRE keystore
* by default the file: $JAVA_HOME/lib/security/cacerts
*/
private static final String TRUSTSTORE = "/opt/java/jre7/lib/security/cacerts";
/**
* password used for keystore and trustore
* for convience we use the same password for as a key password
*/
private static final char[] PSSWD = {'a', 'p', 'o', 'g'};
/**
* default truststore password
*/
private static final char[] TRUST_PSSWD = "changeit".toCharArray();
/**
* service endpoint
*/
private static final String ENDPOINT = "https://magdaruimteenomgdienst-aip.vlaanderen.be/GeefDossierDienst-01.00/soap/WebService";
/**
* regexp filter of the service SSL/TLS certificate
*/
```

```
private static final String SERVICE_CERT_DN_REGEXP = "CN=\\*\\.vlaanderen
.be,O=Vlaamse overheid,L=Brussel,ST=Brabant,C=BE";
/**
* test dossier id
*/
public static final String DOSSIER_ID = "DBA_20120000009";
/**
* vraag context naam
*/
private static final String CONTEXT_NAAM = "Certif1";
/**
* vraag context versie
*/
private static final String CONTEXT_VERSIE = "01.00.0000";
```

Sender information needs to be configured based on certificate properties (CN) and service provider contract (authentication information):

```
/**
* afzender ID
* Afzender.Identificatie = de URI van de toepassing
* (moet hetzelfde zijn als de URI
* in de Certificaat.CN!)
* antwerpen.be/GARO of antwerpen.be/GARO-test
*/
private static final String AFZENDER_ID = "antwerpen.be/GARO";
/**
* afzender naam – not checked data, can be any string
*/
private static final String AFZENDER_NAAM = "BELVK2199336923";
/**
* Afzender.OrganisatieEenheid = de unieke identificatie van de organisatie
* (RWO-ID) zoals toegekend en beheerd door RWO (moet hetzelfde zijn als
* de RWO-ID in de Certificaat.O!)
*/
private static final String AFZENDER_ORG_EENHEID = "BELLK0207500123";
private static final String AFZENDER_REFERTE = "123-456-789";
private static final String AFZENDER_HOEDANIGHEID = "2308";
```

For signing the message we will need to prepare a property file placed in the application classpath, for this case we will create a file **META-INF/signature.properties** with content as follows:

```
#property file for WSS4J interceptor
org.apache.ws.security.crypto.provider=org.apache.ws.security.components.crypto.Merlin
org.apache.ws.security.crypto.merlin.keystore.type=jks
org.apache.ws.security.crypto.merlin.keystore.password=apog
org.apache.ws.security.crypto.merlin.keystore.alias=dba
org.apache.ws.security.crypto.merlin.file=/projects/DBA/cert/test_keystore.jks
```

To provide the key password for signing the SOAP message we will create very simple a password callback handler:

```
package dbawstest;
import java.io.IOException;
import javax.security.auth.callback.Callback;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.callback.UnsupportedCallbackException;
import org.apache.ws.security.WSPasswordCallback;
/**
* a simple password callback handler
* @author Gabriel Vince
*/
public class KeyPasswordCallback implements CallbackHandler {
private String password;
public KeyPasswordCallback(String password)
{
this.password = password;
}
@Override
public void handle(Callback[] callbacks) throws IOException, UnsupportedCallbackException {
WSPasswordCallback pc = (WSPasswordCallback) callbacks[0];
pc.setPassword(this.password);
}
}
```

Imported classes:

```
import be.vlaanderen.rwo.dba.dossiervraag.v1.GeefDossierVraagType;
import be.vlaanderen.vip.geefdossierdienst_01_00.webservice.GeefDossierPortType;
import be.vlaanderen.vip.geefdossierdienst_01_00.webservice.RepliekType;
import be.vlaanderen.vip.geefdossierdienst_01_00.webservice.VerzoekType;
import be.vlaanderen.vip.geefdossierdienst_01_00.webservice.VraagType;
import be.vlaanderen.vip.geefdossierdienst_01_00.webservice.VragenType;
import be.vlaanderen.vip.generiek_02_00.AfzenderAdresType;
import be.vlaanderen.vip.generiek_02_00.AnnotatieType;
import be.vlaanderen.vip.generiek_02_00.AnnotatiesType;
import be.vlaanderen.vip.generiek_02_00.BerichtType;
```

```
import be.vlaanderen.vip.generiek_02_00.BerichtTypeType;
import be.vlaanderen.vip.generiek_02_00.ContextType;
import be.vlaanderen.vip.generiek_02_00.OntvangerAdresType;
import be.vlaanderen.vip.generiek_02_00.TijdstipType;
import java.io.FileInputStream;
import java.io.InputStream;
import java.security.KeyStore;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.net.ssl.KeyManagerFactory;
import javax.net.ssl.TrustManagerFactory;
import javax.xml.ws.WebServiceException;
import org.apache.cxf.configuration.jsse.TLSClientParameters;
import org.apache.cxf.configuration.security.CertificateConstraintsType;
import org.apache.cxf.configuration.security.CombinatorType;
import org.apache.cxf.configuration.security.DNConstraintsType;
import org.apache.cxf.configuration.security.FiltersType;
import org.apache.cxf.endpoint.Client;
import org.apache.cxf.frontend.ClientProxy;
import org.apache.cxf.interceptor.LoggingInInterceptor;
import org.apache.cxf.interceptor.LoggingOutInterceptor;
import org.apache.cxf.jaxws.JaxWsProxyFactoryBean;
import org.apache.cxf.transport.http.HTTPConduit;
import org.apache.cxf.ws.addressing.EndpointReferenceType;
import org.apache.cxf.ws.security.wss4j.WSS4JOutInterceptor;
import org.apache.ws.security.WSConstants;
import org.apache.ws.security.handler.WSHandlerConstants;
```

## Service invocation

We will step-by-step describe the client configuration and service invocation. We will use the generated service classes to build request message and invoke the web service.

```
GeefDossierPortType geefDossierStukClient = this.getGeefDossierStuckService();
VerzoekType request = Dbawstest.buildRequest(DOSSIER_ID, "random_reference_id");
RepliekType reply = geefDossierStukClient.geefDossier(request);
```

### Building a service client

To instantiate and configure web service client we will use the CXF framework. This cope snipped shows how to instantiate a web service client and how to set WS-Security and TLS options.

```
/**
* @return a service interface port
* @throws Exception
*/
private GeefDossierPortType getGeefDossierStuckService() throws Exception {
// we will use CXF bean factory to instantiate a client
JaxWsProxyFactoryBean proxyFactory = new JaxWsProxyFactoryBean();
proxyFactory.setAddress(ENDPOINT);
proxyFactory.setServiceClass(GeefDossierPortType.class);
// add security feature parameters
this.setSecurityFeatures(proxyFactory);
// set logging (optionally)
LoggingOutInterceptor logOut = new LoggingOutInterceptor();
logOut.setPrettyLogging(true); // ident the output
LoggingInInterceptor logIn = new LoggingInInterceptor();
logIn.setPrettyLogging(true); // ident the output
proxyFactory.getOutInterceptors().add(logOut);
proxyFactory.getInInterceptors().add(logIn);
GeefDossierPortType geefDossierService = (GeefDossierPortType)proxyFactory.create();
Client client = ClientProxy.getClient(geefDossierService);
// set up TLS
HTTPConduit http = (HTTPConduit) client.getConduit();
TLSClientParameters tlsParams = new TLSClientParameters();
this.setupTlsParams(tlsParams);
http.setTlsClientParameters(tlsParams);
return geefDossierService;
}
```

### Configure security features

This configuration ensures message signing and X509 service client authentication. To enable WS-Security options we will use the WSS4J outbound interceptor and set required options.

```
private void setSecurityFeatures(JaxWsProxyFactoryBean proxyFactory) {
Map<String, Object> secMap = new HashMap<String, Object>();
secMap.put(WSHandlerConstants.ACTION,
WSHandlerConstants.SIGNATURE);
// keystore alias
secMap.put(WSHandlerConstants.USER, "dba");
// signature file
secMap.put(WSHandlerConstants.SIG_PROP_FILE, "/META-INF/signature.properties");
// use client certificate only, not complete chain to construct Binary Security Token
secMap.put(WSHandlerConstants.USE_SINGLE_CERTIFICATE, "true");
// set key password handler, provide the KEY password
KeyPasswordCallback pwdCallback = new KeyPasswordCallback(new String(PSSWD));
secMap.put(WSHandlerConstants.PW_CALLBACK_REF, pwdCallback);
// Key identifier type Binary security token
// this is even default value, not necessary to set for WSS4J
secMap.put(WSHandlerConstants.SIG_KEY_ID, "IssuerSerial");
//Signature algorithm http://www.w3.org/2000/09/xmldsig#rsa-sha1
secMap.put(WSHandlerConstants.SIG_ALGO, WSConstants.RSA_SHA1);
//Signature canonicalization http://www.w3.org/2001/10/xml-exc-c14n#
secMap.put("CanonicalizationMethod", WSConstants.C14N_EXCL_OMIT_COMMENTS);
//Digest algorithm http://www.w3.org/2000/09/xmldsig#sha1
secMap.put(WSHandlerConstants.SIG_DIGEST_ALGO, WSConstants.SHA1);
WSS4JOutInterceptor securityInterceptor = new WSS4JOutInterceptor(secMap);
// Binary Key Security Token
securityInterceptor.setProperty("signatureKeyIdentifier", "DirectReference");
proxyFactory.getOutInterceptors().add(securityInterceptor);
}
```

## Configure SSL/TLS properties

This code snippet enables SSL/TLS authentication. Providing a keystore enables client SSL authentication. Providing a truststore the client can be sure it connects to the legitimate service endpoint.

```
private void setupTlsParams(TLSClientParameters tlsClientParameters) throws Exception
{
KeyStore clientKeyStore = loadKeystore(KEYSTORE, STORETYPE, PSSWD);
KeyStore trustStore = loadKeystore(TRUSTSTORE, STORETYPE, TRUST_PSSWD);
KeyManagerFactory kmf = KeyManagerFactory.getInstance(KeyManagerFactory.getDefaultAlgorithm());
// provide TLS client key password
kmf.init(clientKeyStore, PSSWD);
// ensure we will use PKIX Trust Manager which validates
// the server certificate if validation is enables
TrustManagerFactory tmf = TrustManagerFactory.getInstance("PKIX");
PKIXBuilderParameters pkixParams = new PKIXBuilderParameters(trustStore, new X509CertSelector());
pkixParams.setRevocationEnabled(true); //should be true by default
ManagerFactoryParameters tmfParams = new CertPathTrustManagerParameters(pkixParams);
tmf.init(tmfParams);
tlsClientParameters.setKeyManagers(kmf.getKeyManagers());
tlsClientParameters.setTrustManagers(tmf.getTrustManagers());
// these filters ensure that a ciphersuite with
// export-suitable or null encryption is used,
// but exclude anonymous Diffie-Hellman key exchange as
// this is vulnerable to man-in-the-middle attacks
FiltersType tlsFilter = new FiltersType();
tlsFilter.getInclude().add(".EXPORT.");
tlsFilter.getInclude().add(".EXPORT1024.");
tlsFilter.getInclude().add(".WITH_DES.");
tlsFilter.getInclude().add(".WITH_AES.");
tlsFilter.getInclude().add(".WITH_NULL.");
tlsFilter.getExclude().add(".DH_anon.");
tlsClientParameters.setCipherSuitesFilter(tlsFilter);

// following settings are optional
// assuming server certificate is CN=*.vlaanderen.be,O=Vlaamse overheid,L=Brussel,ST=Brabant,C=BE
// we will ensure if the certificate is from the "Vlaamse overheid"
// establishing direct trust – trust only to the
// CN=*.vlaanderen.be,O=Vlaamse overheid,L=Brussel,S=Brabant,C=BE
CertificateConstraintsType certConstraint = new CertificateConstraintsType();
DNConstraintsType dnConstraint = new DNConstraintsType();
dnConstraint.getRegularExpression().add(SERVICE_CERT_DN_REGEXP);
certConstraint.setSubjectDNConstraints(dnConstraint);
tlsClientParameters.setCertConstraints(certConstraint);
}
```

```
Example method to load a keystore
private KeyStore loadKeystore(String file, String storeType, char[] psswd) throws Exception
{
KeyStore keyStore = KeyStore.getInstance(storeType);
InputStream in = new FileInputStream(file);
// provide the keystore psswd
keyStore.load(in, psswd);
in.close();
return keyStore;
}
```

## Building a sample request object

To successfully invoke a web service, a correctly formatted and filled request is necessary. This method shows how to build a basic request structure and to fill mandatory fields. We will use generated web service client classes.
Note: Afzender / Naam must be equal to the CN attribute of the client certificate.

```
/**
* build a sample GeefDossierStuk request
**/
public static VerzoekType buildRequest(String dossierId, String reference)
{
java.text.DateFormat dateFormat = new java.text.SimpleDateFormat("yyyy-MM-dd");
java.text.DateFormat timeFormat = new java.text.SimpleDateFormat("HH:mm");
VerzoekType request = new VerzoekType();
//context
ContextType context = new ContextType();
request.setContext(context);
context.setNaam(CONTEXT_NAAM);
context.setVersie(CONTEXT_VERSIE);
//bericht
BerichtType bericht = new BerichtType();
context.setBericht(bericht);
bericht.setType(BerichtTypeType.VRAAG);
TijdstipType tijdstip = new TijdstipType();
bericht.setTijdstip(tijdstip);
Date currentDate = new Date();
tijdstip.setDatum(dateFormat.format(currentDate));
tijdstip.setTijd(timeFormat.format(currentDate));
//Afzender / onvanger
// must be equal to certificate CN
afzender.setIdentificatie(AFZENDER_ID);
//Afzender naam is not validated, can be any string
afzender.setNaam(AFZENDER_NAAM);
// O attribute of the client certificate
afzender.setOrganisatieEenheid(AFZENDER_ORG_EENHEID);
afzender.setReferte(AFZENDER_REFERTE);
afzender.setHoedanigheid(AFZENDER_HOEDANIGHEID);
afzender.setOrganisatieEenheid(AFZENDER_ORG_EENHEID);
OntvangerAdresType ontvanger = new OntvangerAdresType();
bericht.setOntvanger(ontvanger);
ontvanger.setIdentificatie("dba.vlaanderen.be");
ontvanger.setNaam("dba");
// end afzender / ontvanger
//end bericht
AnnotatiesType annotaties = new AnnotatiesType();
context.setAnnotaties(annotaties);
AnnotatieType annotatie = new AnnotatieType();
annotaties.getAnnotatie().add(annotatie);
annotatie.setNaam("taal");
annotatie.setWaarde("nl");
// end context
VragenType vragen = new VragenType();
request.setVragen(vragen);
VraagType vraag = new VraagType();
vragen.setVraag(vraag);
// any value helping the client to correlate a response
vraag.setReferte("dossier:"+DOSSIER_ID);
// the request body
GeefDossierVraagType inbound = new GeefDossierVraagType();
vraag.setInhoud(inbound);
inbound.setDossierId(DOSSIER_ID);
inbound.setInclusiefDossierstukken(false);
inbound.setInclusiefDossierverloop(false);
inbound.setInclusiefDuimspijkers(false);
```

```
inbound.setDossierfase("samenstelling");
return request;
}
```

## Configuration and Invoking Web Service with Spring configuration

The Spring Framework is a very popular open source application framework and Inversion of Control container for the Java platform.
{+}http://www.springsource.org/+
In this chapter we will show an example to build a Spring context used for invoking the RWO / DRA web services. The Spring context configuration is analogical to the previous Java code.

### Spring configuration with WSS4J

The Spring context configuration is defined to :

- configure WSS4J interceptor enabling WS-Security features
- configure a web service client using the CXF framework
- configure the HTTPS transport protocol to enable SSL client authentication and server certificate trust
- execute a test method

### Limitation of the Spring based example

In this example we won't set CertPathTrustManagerParameters (used for server certificate validation), as initializing the trust manager requires passing parameters with "non-bean" approach. To achieve this in the Spring configuration, bean factories can be created. This is out of scope of this document.

### Spring context test bean

In this chapter we will define context basic context structure and namespaces. A test bean is defined to execute a test method.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:jaxws="http://cxf.apache.org/jaxws"
xmlns:http="http://cxf.apache.org/transports/http/configuration"
xmlns:sec="http://cxf.apache.org/configuration/security"
xmlns:cxf="http://cxf.apache.org/core"
xsi:schemaLocation="
http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
http://cxf.apache.org/jaxws http://cxf.apache.org/schemas/jaxws.xsd
http://cxf.apache.org/transports/http/configuration http://cxf.apache.org/schemas/configuration/http-conf.xsd
http://cxf.apache.org/core http://cxf.apache.org/schemas/core.xsd">

<!-- main method - service invocation -->
<bean id="main" class="dbawstest.SpringDbaTest" init-method="executeTest">
<property name="proxy" ref="geefDossierStukClient"></property>
</bean>
</beans>
```

### Spring WSS4J configuration

In this chapter we will define a WSS4J outbound interceptor used for a web service client. Please note, that a property file described in the Java code example is used for signing.

```
<!-- WSS4J outbound interceptor enabling WS-Security features -->
<bean id="securityInterceptor" class="org.apache.cxf.ws.security.wss4j.WSS4JOutInterceptor">
<constructor-arg>
<map>
<entry key="action" value="Signature"/>
<entry key="user" value="dba"/>
<entry key="signaturePropFile" value="/META-INF/signature.properties"/>
<entry key="useSingleCertificate" value="true"/>
<entry key="passwordCallbackRef">
<!-- password callback handler to provide key password -->
<bean id="passwordHandler" class="dbawstest.KeyPasswordCallback">
<constructor-arg>
<value>apogado</value>
```

```
</constructor-arg>
</bean>
</entry>
<entry key="signatureKeyIdentifier" value="IssuerSerial"/>
<entry key="signatureAlgorithm" value="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
<entry key="CanonicalizationMethod" value="http://www.w3.org/2001/10/xml-exc-c14n#"/>
<entry key="signatureDigestAlgorithm" value="http://www.w3.org/2000/09/xmldsig#sha1"/>
<entry key="signatureKeyIdentifier" value="DirectReference"/>
</map>
</constructor-arg>
</bean>
```

## Spring WS Client configuration

The following definition instantiates a web service client.

```
<!-- web service client -->
<jaxws:client
id="geefDossierStukClient"
address="https://magdaruimteenomgdienst-aip.vlaanderen.be/GeefDossierDienst-01.00/soap/WebService"
serviceClass="be.vlaanderen.vip.geefdossierdienst_01_00.webservice.GeefDossierPortType" >
<jaxws:inInterceptors>
<bean class="org.apache.cxf.interceptor.LoggingInInterceptor">
<property name="prettyLogging" value="true" />
</bean>
</jaxws:inInterceptors>
<jaxws:outInterceptors>
<ref bean="securityInterceptor" />
<bean class="org.apache.cxf.interceptor.LoggingOutInterceptor">
<property name="prettyLogging" value="true" />
</bean>
</jaxws:outInterceptors>
</jaxws:client>
```

## Sprint TLS configuration

Following configuration enables TLS security calling the web service.

```
<!-- TLS configuration -->
<http:conduit name="{http://webservice.GeefDossierDienst-01_00.vip.vlaanderen.be\}geefDossierPortTypePort.http-conduit"
>
<http:tlsClientParameters>
<sec:keyManagers keyPassword="apog">
<sec:keyStore type="JKS" password="apog"
file="/projects/DBA/cert/test_keystore.jks"/>
</sec:keyManagers>
<!--
default trust manager is used. By default PKIX Trust Manager is instantiated which verifies server certificate if revocation check is enabled -->
<sec:trustManagers>
<sec:keyStore type="JKS" password="changeit"
file=""[JAVA_HOME]/lib/security/cacerts"/>
</sec:trustManagers>
<sec:cipherSuitesFilter>
<!-- these filters ensure that a ciphersuite with
export-suitable or null encryption is used,
but exclude anonymous Diffie-Hellman key exchange as
this is vulnerable to man-in-the-middle attacks -->
<sec:include>.EXPORT.</sec:include>
<sec:include>.EXPORT1024.</sec:include>
<sec:include>.WITH_DES.</sec:include>
<sec:include>.WITH_AES.</sec:include>
<sec:include>.WITH_NULL.</sec:include>
<sec:exclude>.DH_anon.</sec:exclude>
</sec:cipherSuitesFilter>
<!--
optional settings to ensure client side security
that the client connects to the right web service endpoint
-->
<sec:certConstraints>
<sec:SubjectDNConstraints>
<sec:RegularExpression>CN=*\.vlaanderen\.be,O=Vlaamse overheid,L=Brussel,ST=Brabant,C=BE</sec:RegularExpression>
</sec:SubjectDNConstraints>
</sec:certConstraints>
</http:tlsClientParameters>
<http:client AutoRedirect="true" Connection="Keep-Alive"/>
```

</http:conduit>

### Service Invocation

According to the Spring configuration the test is executed from class dbawstest.SpringDbaTest method executeTest:
The method Dbawstest.buildRequest is used from the previous Java code.

```
/**
* IoC / Spring defined service client
*/
private GeefDossierPortType proxy;
public void executeTest()
{
try
{
VerzoekType request = Dbawstest.buildRequest(Dbawstest.DOSSIER_ID, dbawstest.Dbawstest.DOSSIER_ID);
RepliekType reply = proxy.geefDossier(request);
}
catch(Exception ex)
{
logger.log(Level.SEVERE,"test",ex);
}
}
```

## Configuration and Invoking Web Service with .NET code

## Framework

In this example we use the C# programming language for leveraging the .NET 4 framework.

## Assumptions

In the code snippets we assume that the following configurations are in place:

- server side certificate is trusted - signed by a trusted certification authority
- client certificate is placed in a personal system key store
- Web Service Proxy code is generated. For this example the Web Service Proxy is created by C# MS Visual Studio as a Service Reference.

## C# code snippets

This code snippet assumes that the following requirements are met:

- client key and certificate are stored in the personal system certificate store.
- Server certificate (vlaanderen.be) is signed by a trusted certificate authority (GlobalSign Root CA). which is trusted and stored in the system certificate store as a Trusted Authority. GlobalSign certificate is configured as a trusted root certificate by default, so no action is needed.

### WS-Security configuration

```
//configure binding security features
var asymmetricBindingElement = new AsymmetricSecurityBindingElement
{
MessageSecurityVersion =
MessageSecurityVersion.WSSecurity11WSTrust13WSSecureConversation13WSSecurityPolicy12BasicSecurityProfile10,
InitiatorTokenParameters = new X509SecurityTokenParameters(),
RecipientTokenParameters = new X509SecurityTokenParameters(),
MessageProtectionOrder = MessageProtectionOrder.SignBeforeEncrypt,
SecurityHeaderLayout = SecurityHeaderLayout.Strict,
EnableUnsecuredResponse = true,
IncludeTimestamp = false,
DefaultAlgorithmSuite = SecurityAlgorithmSuite.Basic128Rsa15
```

```
};
asymmetricBindingElement.SetKeyDerivation(false);
asymmetricBindingElement.EndpointSupportingTokenParameters.Signed.Add(new X509SecurityTokenParameters());
var binding = new CustomBinding();
binding.Elements.Add(asymmetricBindingElement);
binding.Elements.Add(new TextMessageEncodingBindingElement(MessageVersion.Soap11, Encoding.UTF8));
binding.SendTimeout = new TimeSpan(0, 3, 0);
```

## TLS client configuration

```
//SSL
//We will set trust using a root certificate "GlobalSign Root CA"
var httpsBindingElement = new HttpsTransportBindingElement {RequireClientCertificate = true};
binding.Elements.Add(httpsBindingElement);
var ws = new GeefOndernemingDienst.geefDossierPortTypeClient(
binding,
new EndpointAddress(
new Uri("https://magdaruimteenomgdienst-aip.vlaanderen.be/GeefDossierDienst-01.00/soap/WebService"),
new DnsEndpointIdentity("GlobalSign Root CA"),
new AddressHeaderCollection()) );
ws.Endpoint.Contract.ProtectionLevel = System.Net.Security.ProtectionLevel.Sign;
ws.ClientCredentials.ServiceCertificate.Authentication.CertificateValidationMode = X509CertificateValidationMode.ChainTrust;
ws.ClientCredentials.ClientCertificate.SetCertificate(
StoreLocation.CurrentUser,
StoreName.My,
X509FindType.FindBySubjectName,
"[client certificate CN name]");
ws.ClientCredentials.ServiceCertificate.SetDefaultCertificate(
StoreLocation.CurrentUser,
StoreName.Root,
X509FindType.FindBySubjectName,
"GlobalSign Root CA");
// if a client certificate is stored in a file, following code can be used
// ws.ClientCredentials.ClientCertificate.Certificate = new X509Certificate2("[identity keystore].p12", "[keystore password]");
// from a functional aspect this next step is optional,
// but allows custom certificate validation (CRL)
ServicePointManager.ServerCertificateValidationCallback = new RemoteCertificateValidationCallback(ValidateRemoteCertificate);
```

---

## Server certificate verification

It is possible to use a callback to validate the certificate in an SSL certificates. This is an example callback
Note that the default chaining engine can be overridden using the CryptoConfig class. On Microsoft Windows Server 2003, the default engine conforms to the specification described in RFC3280, "Certificate and Certificate Revocation List (CRL) Profile."

```
private static bool ValidateRemoteCertificate(object sender, X509Certificate cert, X509Chain chain, SslPolicyErrors policyErrors)
{
bool result;
if(policyErrors!= SslPolicyErrors.NONE)
{
result = false;
}
else
{
X509Certificate2 certx = (X509Certificate2)cert;
result = certx.Verify();
}
return result;
}
```

## Web Service invocation

```
ws.Open();
GeefOndernemingDienst.RepliekType repliek = ws.GeefDossier(SampleRequest.Verzoek);
ws.Close();
```

## Example of building a request payload

```
GeefOndernemingDienst.GeefDossier req = new GeefOndernemingDienst.GeefDossier()
{
Verzoek = new GeefOndernemingDienst.VerzoekType()
{
Context = new GeefOndernemingDienst.ContextType()
{
Naam = "Certif1",
Bericht = new GeefOndernemingDienst.BerichtType()
{
Type = GeefOndernemingDienst.BerichtTypeType.VRAAG,
Tijdstip = new GeefOndernemingDienst.TijdstipType()
{
Datum = "2012-12-21", // current date
Tijd = "15:01" // current time
},
Afzender = new GeefOndernemingDienst.AfzenderAdresType()
{
Identificatie = "...", //certificate CN
Naam = "BEL...", //certificate CN
Referte = "123-456",
OrganisatieEenheid = "BELL",
Hoedanigheid = "2308"
},
Ontvanger = new GeefOndernemingDienst.OntvangerAdresType()
{
Identificatie = "dba...",
Naam = "dba..."
}
},
Annotaties = new GeefOndernemingDienst.AnnotatieType[] {
new GeefOndernemingDienst.AnnotatieType() { Naam = "taa1", Waarde="n1"}
}
},
Vragen = new GeefOndernemingDienst.VragenType()
{
Vraag = new GeefOndernemingDienst.VraagType()
{
Referte = "doss",
Inhoud = new GeefOndernemingDienst.GeefDossierVraagType()
{
DossierId = "DBA...",
InclusiefDossierstukken = false,
InclusiefDuimspijkers = false,
InclusiefDossierverloop = false,
Dossierfase = "sampleasdas"
}
}
}
}
};
return req;
```

## Annex: Server Certificate and Sample Request

## Sample Request

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Header>
<wsse:Security soap:mustUnderstand="1"
xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
<wsse:BinarySecurityToken
EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary"
ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3"
wsu:Id="CertId-5A30CADA45FC26967813567128475701"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">MIIE6DCCA9CgAwIBAgIOAQAAAAABO
m5GlX1C1V4wDQYJKoZIhvcNAQEFBQAwNDELMAkGA1UEBhMCQkUxFjAUBgNVBAMTDUdvdmVybm1lbnQgQ0ExDTALBgNVBAUTBDI
wMTAwHhcNMTIxMDE3MTAzMDUwWhcNMTQwMTE3MTAzMDUwWjCBwjEZMBcGA1UEAwwQVkJFTFZLMjE5OTMzNjkyMzELMAkGA1
UEBhMCQkUxNTAzBgkqhkiG9w0BCQEWJmRpZ2l0YWxlYm91d2FhbnZyYWFnQHJ3by52bGFhbmRlcmVuLmJlMREwDwYDVQQFEwgyM
DEyMTAwMzERMA8GA1UEBwwIQnJ1c3NlbHMxETAPBgNVBAgMCEJydXNzZWxzMQ0wCwYDVQQLDARURVNUMRkwFwYDVQQKDBB
WbGFhbXNlIE92ZXJoZWlkMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAk7eX7HPsC+rTrgFF3GBRz048kFWUFJ3DhDsRjTM
EoJ1aSD+EoK0pILAl+ygosYrBueSLyHlVcd6BZbz8Kjz/Ufg6XlVyHho9VwagI2E9w6OmJ6t96oat6DFF+iy6yWn+v1Y8oUKanLRj4KalpAlXeG6
TZNaoZZgHh9ddDb+weBYaPIfMhkKEMx9PF7di0JYq7iU218qlGvc0bUOlClQdFpSNiaeMPYxbpLX8s9O2AzORkO23a6XFfN88FDazQQ4NU
CfWHIrNmwi/xbSamyQDjY4cQVHo+jMpPT0IcJoaPFy49C9bZBV5Pekxu+hNFrT4Qd5Vxg29nhjejYXjLq734QIDAQABo4IBZzCCAWMwHwY
DVR0jBBgwFoAUQZbOhaflXugWWT0K8YTd8/K7TokwbgYIKwYBBQUHAQEEYjBgMDYGCCsGAQUFBzAChipodHRwOi8vY2VydHMucGtp
```

LmJlbGdpdW0uYmUvYmVsZ2l1bXJzMi5jcnQwJgYIKwYBBQUHMAGGGmh0dHA6Ly9vY3NwLnBraS5iZWxnaXVtLmJlMAkGA1UdEwQCM
AAwRAYDVR0gBD0wOzA5BgdgOAkBAQMDMC4wLAYIKwYBBQUHAgEWIGh0dHA6Ly9yZXBvc2l0b3J5LnBraS5iZWxnaXVtLmJlMD0GA
1UdHwQ2MDQwMqAwoC6GLGh0dHA6Ly9jcmwucGtpLmJlbGdpdW0uYmUvZ292ZXJubWVudDIwMTAuY3JsMA4GA1UdDwEB/wQEAwIE
8DARBglghkgBhvhCAQEEBAMCBLAwHQYDVR0OBBYEFAUiXduhIcSRwLcXAJY/G/AoNXCHMA0GCSqGSIb3DQEBBQUAA4IBAQBNOC
d//02xmx7mub6iY0F/14SIjOmWIqb2+vSuJdP+A0XkB9G9KKW0kCG4UYyp49IgcCEsTOmwLSAKwJS1Jzu/AEVqitMAx+oyyet0oe63EpwmS
+CY807JuUDCeuMmVJ2x7QPGHLnzMN5vrtvffEHJdlJzhwAR/b73e9elPyEzuW/ssydsnmjCtsl69yyFHhSfu2QG+p6XhYtOzG6dLt67nCypsR
NFhR5fbrvj+OU3N9yNqr5DwP9BQhs1Ar/hx1KA6BBl7fM66escZEGCKHtsciSDEnXnTXUEB6t4QIfxbebjQoJem34eoUzZHEHCBeAXw/Km4
QXvGckG4vgBHzzj
<ds:Signature Id="Signature-1" xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
<ds:SignedInfo>
<ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
<ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
<ds:Reference URI="#id-2">
<ds:Transforms>
<ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
</ds:Transforms>
<ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
<ds:DigestValue>B01rGiLBK5/+/Tnzy+EjFOaf8go=</ds:DigestValue>
</ds:Reference>
</ds:SignedInfo>
<ds:SignatureValue>eBXzttauIK/NNKlutT4Grh7Y+AKklq8q+aB2yxKzNghK5/ASRtCLQco1ImrrKzEvSm3/tT3c5Yq7
KVxDi2uZ1umk/McehB6WahHc+J3dTo5SG0mGzSGlw67oXZmfwux8oLSdv0AV4dx/7J7gAPEWg2HO
T/BqMKsKLR3V9mC/n+kBp2rSDUBM6ZV3vLxRnOanWZCsAh8QGGRvMwst4G7USMLB55BnpdIrOIB9
mEj1mRpzMtA590Y48bqFJEOtecCDtyvb57wG4KH15RGVAa9VG0S0fzHfvyB5y+yJlKZb3Hnl8NAK
UdAfW7Rcq9LMNxsMvmfdrytNHom4cSbUH5NKmQ==</ds:SignatureValue>
<ds:KeyInfo Id="KeyId-5A30CADA45FC26967813567128476172">
<wsse:SecurityTokenReference wsu:Id="STRId-5A30CADA45FC26967813567128476173"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
<wsse:Reference URI="#CertId-5A30CADA45FC26967813567128475701"
ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3"/>
</wsse:SecurityTokenReference>
</ds:KeyInfo>
</ds:Signature>
</wsse:Security>
</soap:Header>
<soap:Body wsu:Id="id-2" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
<ns2:GeefDossier xmlns:ns2="http://webservice.GeefDossierDienst-01_00.vip.vlaanderen.be"
xmlns:ns3="urn:be:vlaanderen:ruimtelijkeordening:dossier:v1">
<Verzoek>
<Context>
<Naam>Certif1</Naam>
<Versie>01.00.0000</Versie>
<Bericht>
<Type>VRAAG</Type>
<Tijdstip>
<Datum>2012-12-21</Datum>
<Tijd>15:01</Tijd>
</Tijdstip>
<Afzender>
<Identificatie>antwerpen.be/GARO</Identificatie>
<Naam>VBELVK2199336923</Naam>
<Referte>123-456-789</Referte>
<OrganisatieEenheid>BELLK0207500123</OrganisatieEenheid>
<Hoedanigheid>2308</Hoedanigheid>
</Afzender>
<Ontvanger>
<Identificatie>dba.vlaanderen.be</Identificatie>
<Naam>dba</Naam>
</Ontvanger>
</Bericht>
<Annotaties>
<Annotatie>
<Naam>taal</Naam>
<Waarde>nl</Waarde>
</Annotatie>
</Annotaties>
</Context>
<Vragen>
<Vraag>
<Referte>dossier:DBA_20120000009</Referte>
<Inhoud>
<DossierId>DBA_20120000009</DossierId>
<InclusiefDossierstukken>false</InclusiefDossierstukken>
<InclusiefDuimspijkers>false</InclusiefDuimspijkers>
<InclusiefDossierverloop>false</InclusiefDossierverloop>
<Dossierfase>samenstelling</Dossierfase>
</Inhoud>

</Vraag>
</Vragen>
</Verzoek>
</ns2:GeefDossier>
</soap:Body>
</soap:Envelope>

# Document properties

**OPGESTELD DOOR:**

| VERSIE | DATUM | WIE | COMMENTAAR |
|---|---|---|---|
| 1 | 25-01-2013 | Lieven Heuninck - Apogado | Eerste versie |
| 1.1 | 27-02-2013 | Lieven Heuninck - Apogado | Nieuwe template |
| 1.2 | 19-04-2013 | Gabriel Vince - Apogado | .NET code snippets |
| 1.4 | 14-05-2013 | Gabriel Vince - Apogado | Added OCSP checking, cfr remarks GVDE |
| 1.5 | 15-05-2013 | Hans Arents (CORVE) | Kleine tekstverbeteringen |
| 1.6 | 03-06-2013 | Gabriel Vince - Apogado | Kleine .NET aanpassingen, cfr opmerking DL. |

**GERELATEERDE DOCUMENTEN:**

| VERSIE | DATUM | NAAM DOCUMENT | OMSCHRIJVING |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |

**GOEDGEKEURD DOOR:**

| VERSIE | DATUM | WIE |
|---|---|---|
| 1.5 | 15-05-2013 | Hans Arents (CORVE) |
| 1.6 | 05-06-2013 | Hans Arents (CORVE) |