**OIOIDWS**

**Overview and Installation**

Indhold

[1 Formål 3](#_Toc276707632)

[1.1 Formålet med OIOIDWS.JAVA-pakken 3](#_Toc276707633)

[1.2 Formålet med OIOIDWS.Net-pakken 3](#_Toc276707634)

[1.3 Forudsætninger 3](#_Toc276707635)

[1.4 Java-pakkens indhold 4](#_Toc276707636)

[1.4.1 Overblik og scenarier 4](#_Toc276707637)

[1.5 .Net-pakkens indhold 10](#_Toc276707638)

[1.5.1 Delkomponenter 10](#_Toc276707639)

[1.5.2 Den rige .NET klient 11](#_Toc276707640)

[1.6 Andre scenarier 12](#_Toc276707641)

[2 Installation Guide 13](#_Toc276707642)

[2.1 Configuring the Java components 13](#_Toc276707643)

[2.2 Configuring the .NET components 19](#_Toc276707644)

[2.2.1 Webservice provider 19](#_Toc276707645)

[2.2.2 IDP using ADFS 2.0 20](#_Toc276707646)

[2.2.3 Rich client 20](#_Toc276707647)

# Formål

Denne pakke indeholder en eksempel-implementation af specifikationerne for den identitetsbaserede webservice, OIOIDWS, som sammenfatter forskellige del-elementer fra primært SAML, WS-Trust og Liberty Basic SOAP Binding.

En identitetsbaseret webservice er en service, som opererer på vegne af en bruger. Dette kunne være en service, som giver lægen adgang til journaloplysninger, en service der giver offentlige myndigheder adgang til skatteoplysninger eller en service, der gør, at banker kan udveksle oplysninger osv.

OIO identitets-baserede webservices (OIOIDWS) er en række webservice profiler, som er udviklet af IT & Telestyrelsen, og som anviser en standardiseret måde at kalde fra en webservice-consumer til en webservice-provider på vegne af slutbrugeren på med sikker videreførelse af brugerens identitet. OIOIDWS gør det muligt at tilbyde services, som udstiller personlige data på en sikker og standardiseret måde – hvilket er en forudsætning for automatisering af processer på tværs af forskellige institutioner, organisationer eller domæner.

OIOIDWS pakkerne implementerer et specifikt scenarie med identitetsbaserede web services, hvor brugeren via en browser først logger på en web applikation, der så efterfølgende har brug for at kalde en web service hos en anden organisation.

## Formålet med OIOIDWS.JAVA-pakken

Formålet med OIOIDWS.JAVA-pakken er at tilbyde de færdige komponenter, der skal til for at kunne køre et lille eksempel på et OIOIDWS-båret SOAP-kald på Java-platformen baseret på Sun’s referenceimplementation af de krævede komponenter og libraries.

## Formålet med OIOIDWS.Net-pakken

Formålet med OIOIDWS.Net-pakken er at demonstrere, hvorledes Microsoft’s værktøjer anvendes og konfigureres for at udføre et OIOIDWS-kald.

## Forudsætninger

Da OIOIDWS adresserer sikre (WS-Trust), SOAP-baserede webservice-kald med credentials båret af en SAML assertion, er hele pakken i sagens natur henvendt til den mere erfarne læser. Det forudsættes derfor, at læseren til et vist omfang er bekendt med de nævnte begreber og webservices + system-system-integration i almindelighed.

Det anbefales i øvrigt at læse såvel Java- som .Net-afsnittene for at opnå en samlet forståelse af, hvordan OIOIDWS adresseres af de to platforme.

## Java-pakkens indhold

Pakken OIOIDWS.JAVA indeholder 3 moduler, der alle nærmere beskrives i installationssektionen:

* **Web Service Consumer (WSC)**- en SAML 2.0 Service Provider som kalder STS’en og WSP’ens web service.
* **Web Service Provider (WSP)**- indeholder en “echo service” som kræver et SAML token fra STS’en.
* **Security Token Service (STS)**- en OIOSAML Demo STS som overholder WS-Trust 1.3 standarden og kan udstede et service-specifikt token til eksemplets webservice-kald.

Disse moduler demonstrerer tilsammen koncepterne fra OIOIDWS til java platformen.

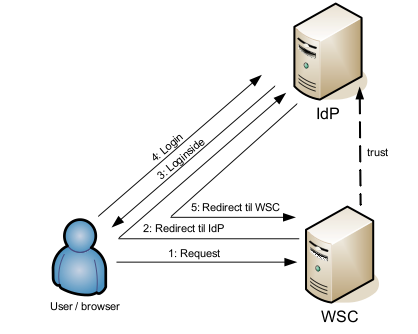
Det er dog vigtigt at tilføje, at STS’en som input kræver en IdP-udstedt SAML-assertion. IdP’en og opsætningen heraf har principielt ikke noget med OIOIDWS at gøre, men for helhedens skyld er der i OIOIDWS.java-pakken indeholdt en vejledning til opsætning af en sådan.

### Overblik og scenarier

Dette afsnit vil kort skitsere og forklare sammenhængen mellem modulerne, beskederne mellem dem, og hvor der skal etableres trust mellem komponenterne.

#### Bootstrap token

Nedenstående figur tager udgangspunkt i visse elementer fra den fællesoffentlige brugerstyring, idet hele scenariet ”bootstrap’es” fra en webapplikation tilkoblet en IdP. Webservice-klienten er således på samme tid WSC i OIOIDWS-sammenhæng og Service Provider i OIOSAML-sammenhæng:



Brugeren logger ind på WSC’ens webapplikation via den tilkoblede IdP:

1: Brugeren tilgår webapplikationen.

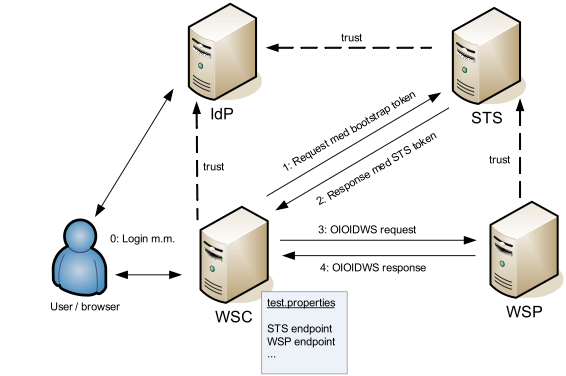
2,3,4: Brugeren re-dirigeres til IdP’en, præsenteres for en login prompt og logger ind.

5: Brugeren dirigeres tilbage til webapplikationen.

IdP’ens rolle er her udover at håndtere login (og single signon, hvis flere applikationer er tilkoblet samme IdP) også at udstede og signere et SAML-token, som gives med i det afsluttende kald til applikationen (punkt 5). Dette SAML-token er applikationens garanti for, at login er foretaget korrekt, og at brugeren er den, han/hun udgiver sig for, og derudover indeholder token’et det såkaldte ’bootstrap’ token, som sidenhen skal bruges til den videre autentifikation i forbindelse med webservice-kaldene.

#### OIOIDWS-kaldet

For at udføre webservice-kaldet mellem WSC og WSP, skal bootstrap-token’et veksles til et token, som er specifikt for den pågældende service; til dette formål anvendes en STS. Det komplette scenarie for selve webservice-kaldet ser dermed således ud:



0: Brugeren logger ind og tilvejebringer et bootstrap token.

1,2: WSC ”veksler” dette token til et servicespecifikt token ved STS’en.

3,4: Webservice-kaldet udføres.

Det servicespecifikke token er akkurat som bootstrap-token’et et SAML token, men har udover en angivelse af servicen en tidsbegrænsning og et serienummer, som skal sikre at det kun anvendes til den specificerede service en enkelt gang og umiddelbart efter udstedelse.

I OIOIDWS.java pakken bruger eksempel-WSC’en en SimpleSAMLPhP IDP til udstedelse af Bootstrap tokens. I .Net pakken er det et ADFS der udsteder tokens.

#### Request to interact

Det ovenfor skitserede eksempel er det simpleste. En lidt mere avanceret variant anvender Request to Interact (R2I), hvilket er en måde at involvere slutbrugeren på i et ekstra sikkerhedscheck eller en samtykke-erklæring om at anvende hans/hendes data, inden selve requestet gennemføres på WSP. Da ser flowet ud som skitseret i nedenstående figur:



Dette scenarie kan også testes af pakken.

#### Trust

De skitserede scenarier forudsætter, at der etableres trust mellem følgende parter:

* WSC og IdP: Populært sagt, skal WSC stole på det af IdP’en udstedte SAML token.
* WSP og STS: WSP stoler på billetten udstedt af STS’en.
* STS og IdP: STS’en trust’er det bootstrap token, IdP’en har udstedt.

#### Scenarierne i praksis

POC consumer’en er, som tidligere anført, en web applikation. Når man tilgår den færdigkonfigurerede POC consumer, mødes man med følgende skærmbillede:



Linket ”Page requiring login” redirrigerer til IdP’en, og efter login præsenteres det af IdP’en udstedte SAML token i browseren.

Nedenunder præsentationen af denne SAML assertion fremkommer linket ”Trigger token request”, hvilket fremskaffer et token fra STS’en:



Under dette optræder linket ”Perform token WS request”, som udfører det egentlige web service request, og resultatet heraf vises i browseren (nedenfor vises kun et del-element heraf):



Vha. linket ”Trigger request to interact” udføres et R2I-kald fra WSC til WSP. WSP’en redirigerer browseren til en side, hvor brugeren kan indtaste en ’secret’ (= en tekststreng). Når denne er indtastet, færdiggøres webservice-kaldet, og det vises, at den indtastede secret bliver båret med tilbage til responset til WSC.

## .Net-pakkens indhold

### Delkomponenter

Demoapplikationen er en lille WPF/WCF-baseret klient, som henter tokens til brug for kald til Echo webservicen, og som derefter kalder servicen med en streng som echo-servicen blot returnerer.

Projektet består af et antal Visual Studio 2008 projekter:

#### BindingsImplementerer vha. WIF (Windows Identity Foundation) et antal hjælpe-klasser, der bl.a. kan oprette WCF bindings til brug for STS kommunikation. Desuden findes der hjælpemetoder til at requeste tokens fra en STS (klassen TokenClient).

#### ClientUnittests der vha. et selvudstedt bootstraptoken tester hhv. kald til en .NET-baseret og en java-baseret webservice via en java STS, både med og uden SSL.

#### EchoService WCF snitflade til webservicen EchoService, d.v.s. MessageContracts og ServiceInterfaces. Anvendes af både service-klienter samt implementeringen af serviceprovideren.

#### EchoWebServiceProvider (WSP)WCF implementation af en Echo service. Autorisation af servicekald til provideren foretages af ServiceProviderBinding fra Bindings-assembly’en.

#### WPFClient (WSC)WPF baseret .NET GUI demo applikation, der vha. WCF og WIF får udstedt tokens, og anvender disse til, vha. Bindings-komponenten, at foretage det egentlige webservice-kald .

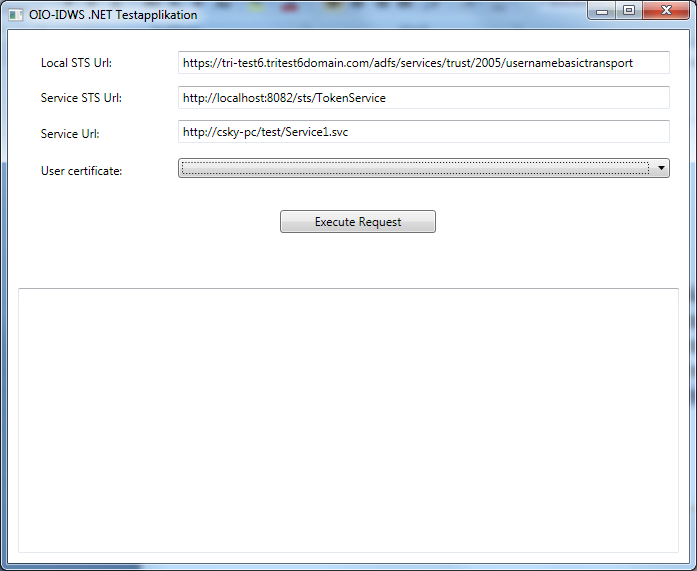
Der følger ingen .NET STS med i pakken, men man kan eksempelvis anvende OIOSAML Trust Demo STS’en fra OIO-SAML.java.

Scenariet for .NET pakken er identisk med det i afsnit [2.1.2](#Ref265849670) beskrevne. WPFClient applikationen indeholder dels kodeeksempler til selv at udstede et bootstraptoken, og dels til at få udstedt et token fra en ADFS 2.0 server. ADFS serveren foretager autorisation vha. en AD Server, der derved fungerer som IdP. Bootstrap-token’et eller det ADFS-udstedte token anvendes til at bede STS’en om et token, der kan anvendes til at autorisere det efterfølgende kald til selve Echo webservice provideren.

### 

### 

### Den rige .NET klient



Demoapplikationen er et eksempel på, hvordan en rig .NET klient kan anvende OIOIDWS. Den giver mulighed for at vælge en række parametre, der anvendes til kaldet af Echo-servicen:

* Local STS Url: Url til den IdP der skal udstede token. Efterlades feltet blankt anvendes et selvudstedt bootstraptoken
* Service STS Url: Url til STS’en der kan udstede tokens til brug for webservice-kaldet.
* Service Url: Url til selve Echo webservicen.
* User certificate: her vælges et certifikat blandt alle lokalt installerede certifikater; det valgte certifikat anvendes ved autentificering mod den lokale STS/IdP.
* Ved et klik på ”Execute request” hentes et token ved IdP’en, dette veksles til et service-token ved Service STS’en, og til sidst kaldes webservicen. Resultatet af hver af de 3 kald vises derefter i tekstfeltet under knappen.

## Andre scenarier

De ovenfor angivne scenarier er blot eksempler på, hvordan OIOIDWS kan anvendes. Det vil formentlig være lige så almindeligt, at klientapplikationen kører på en maskine, der allerede er autentificeret op mod eksempelvis et AD. I en sådan situation vil STS’en muligvis anvende et Kerberos-token som bootstrap-token, dvs. ’bootstrap token’ anvendes som et generelt begreb for det token, der veksles ved STS’en. Ligeledes vil man i et mere udbygget setup have flere STS’er til flere forskellige WSP’er, og evt. situationer, hvor en STS veksler en billet udstedt af en anden STS.

Fælles for scenarierne er princippet om, at et system kalder en webservice på vegne af en bruger, og at brugerens credentials er båret af en STS-udstedt billet.

Flere scenarier kan findes her: <http://digitaliser.dk/resource/416476/artefact/OIO+IDWS+Scenarios+1.0.pdf>

# Installation Guide

This section describes how to install and configure the OIOIDWS components and run a few manual tests based on them.The test setup consists of the following components:

1. Secure Token Service (STS), running in Java using custom code
2. WebService Provider (WSP), running either Java (custom code) or .NET (WIF)
3. WebService consumer (WSC), in this case implemented as a Java-based web application or a rich .Net client (WPF/WIF/WCF)
4. An Identity Provider (IdP), in this case the freely available SimpleSAMLphp for the java platform and an ADFS for the .Net platform

The manual tests show that

* the WSC is protected by an IdP login
* the WSC can retrieve tokens from the STS
* the WSC client can invoke the WSP service
* the WSC and WSP support Request to Interact (R2I)

There is also a suite of automated integration tests that provide more detailed - though not complete - testing of a number of details in this setup. However, the detailed setup for running those tests is not described here.

## Configuring the Java components

The following must be installed: JDK 6, Subversion (this guide assumes the command line version, substitute as appropriate if using something like TortoiseSVN).

#### Install an Application Server

The application server used is GlassFish v2. Download from <https://glassfish.dev.java.net/> and install and start it.This package is tested on Glassfish 2.2.1.

#### Install the STS

The STS consists of a regular WAR file which can be installed into GlassFish/Tomcat or any other servlet container. Download the most recent STS from http://digitaliser.dk/group/42063

**Limitations:**

* Only supports WS-Trust 1.3
* Supports both SOAP 1.1 and 1.2
* Only supports SAML 2.0 TokenType
* All SAML Assertion attributes are copied from the request assertion
* Requests must contain a SAML Assertion in OnBehalfOf
* Requests must be signed
* There is no real error handling
* The AppliesTo value will be copied without further checking

These limitations means that the STS is in no possible way production ready, but this not important for this project where the goal is to show how to establish WSC and WSP functionality. Thus, the STS is just needed for testing of the scenarios.

Before deploying the STS, it must be configured with an EntityID and a certificate. Do this by creating a new file in ~/.oiosaml (C:\Documents and Settings\<username>\.oiosaml on Windows XP or C:\Users\<username>\.oiosaml on Windows 7) called sts.properties containing the following:

sts.entityId=http://sts.oiosaml.netsts.certificate.location=TestVOCES1.pkcs12sts.certificate.password=Test1234

If this file is missing, the deployment will fail with an error saying "IllegalStateException: System not configured".

Modify the values according to your local settings. The certificate location is relative to the sts.properties file itself and can be either a JKS keystore or a PKCS12 file.

For test certificates, go to <https://www.certifikat.dk/export/sites/dk.certifikat.oc/da/developer/eksempler/>.Unzip the archive and deploy the war file. For deployment to GlassFish, do as follows:

* Open <http://localhost:4848> in a browser
* Login (default administrator login is admin/adminadmin)
* Click on Applications in the menu, then on Web Applications
* Click Deploy...
* Choose the sts.war file and click OK

Restart GlassFish. Check the GlassFish logs at domains/domain1/logs/server.log to check that everything is running - one of the last lines should contain "Configured OIOSAML to .../.oiosaml/sts.properties".

#### Install the WebService Provider

The WebService Provider is a standard JEE web application. It has been tested with GlassFish only and it requires Metro/WSIT version 2.0 (not 2.0.1 – there is a problem with signature validation with this version). Download the most recent version from http://digitaliser.dk/group/705156 and unzip the file. Deploy the poc-provider.war file to GlassFish by following the same instructions as for the STS. This can be done into either the same GlassFish instance, or with an instance on another host.The only required configuration for the WSP is that a number of certificates and keys must be installed:

* The STS certificate must be present in the truststore
* The root CA certificate for the WSC certificates must be present in the truststore
* The private key and certificate must be present in the server's keystore

**The STS certificate:**

The STS certificate must be present in the truststore. First, list the contents of the STS keystore file in order to find the alias of the key.

keytool -list -keystore TestVOCES1.pkcs12 -storetype pkcs12

The output should look similar to this:

Keystore type: PKCS12

Keystore provider: SunJSSE

Your keystore contains 1 entry

2, Jul 5, 2010, PrivateKeyEntry,

Certificate fingerprint (MD5): 75:D0:2A:21:A0:68:BB:1E:6A:E9:F2:AE:33:16:31:16

The alias is first part of the second last line (in this case the "2" that starts the line "2, Jul 5, 2010, PrivateKeyEntry"). Note that this is a "PrivateKeyEntry". What we need is a certificate, which can be generated and imported with the following commands.

keytool -export -rfc -keystore TestVOCES1.pkcs12 -storetype pkcs12 -alias "<alias read from the output of the previous command>" > sts.crt(if necessary copy sts.crt to the host of the WebService Provider)cd <glassfish\_home>/domains/domain1/configkeytool -import -trustcacerts -keystore cacerts.jks -file sts.crt -alias sts

The first command requires the password for the keystore (e.g. "Test1234"). The last command requires the password for the truststore of the Glassfish domain, which is "changeit" by default. The command also prompts for confirmation that the certificate should be trusted - make sure to change the default answer from "no" to "yes".

**The root certificate**

The root CA certificate for the WSC certificates must be present in the truststore. For OCES test certificates, the CA certificate can be found at <https://www.certifikat.dk/export/sites/dk.certifikat.oc/da/developer/eksempler/>.

(get the ca certificate and save it to cacert.crt)cd <glassfish\_home>/domains/domain1/configkeytool -import -trustcacerts -keystore cacerts.jks -file cacert.crt -alias cacert

Again, the last command requires the password for the truststore of the Glassfish domain which is "changeit" by default. The command also prompts for confirmation that the certificate should be trusted - make sure to change the default answer from "no" to "yes".

**The private key**

The private key must be copied to the server's keystore and must have a "voces" alias. First, get the alias of the key in the existing file:

keytool -list -keystore TestVOCES1.pkcs12 -storetype pkcs12

As in the section describing the STS certificate import, the alias is the first part on the 2nd last line.Then, import the key into the server keystore:

cd <glassfish\_home>/domains/domain1/configkeytool -importkeystore -srckeystore TestVOCES1.pkcs12 -destkeystore keystore.jks -srcstoretype pkcs12 -destalias voces -srcalias "<alias>"

NOTE 1: The keytool command can behave a bit strange. Even though the alias found in the keystore listing above was for example "2" it may not accept this as srcalias. Given that the source keystore contains only the relevant key, simply try using "1" instead, if you experience this problem - it has been seen to work several times.

The import can be verified by listing the contents of the keystore.jks and verifying that it contains an entry with the alias "voces":

$ keytool -list -keystore keystore.jks

Enter keystore password:

Keystore type: JKS

Keystore provider: SUN

Your keystore contains 2 entries

s1as, Jun 8, 2010, PrivateKeyEntry,

Certificate fingerprint (MD5): D5:45:32:04:5A:1B:13:84:BA:76:85:AF:4C:06:AA:A7

voces, Jul 5, 2010, PrivateKeyEntry,

Certificate fingerprint (MD5): 75:D0:2A:21:A0:68:BB:1E:6A:E9:F2:AE:33:16:31:16

NOTE 2: Glassfish requires that the keys in the keystore all have the same password as the master password of the keystore itself - otherwise the server will fail during startup of the domain with a log entry saying "Cannot recover key" (which just means that it was not possible to find/use a key with the given alias and password). The password can be changed using the following command which will prompt for the keystore password, the (old) key password and the new key password. If the password is changed multiple times it can be a little confusing, because keytool does not prompt for the old key password if the keystore password is also valid as key password.

keytool -keypasswd -keystore keystore.jks -alias voces

It might be necessary to restart GlassFish after installing the keys and certificates.

<glassfish\_home>/bin/asadmin stop-domain<glassfish\_home>/bin/asadmin start-domain

Test that the application is installed correctly by retrieving the WSDL file. To do this, open <http://localhost:8080/poc-provider/ProviderService?wsdl> in a browser (edit hostname and port number according to the local installation if the defaults have been changed).

**Install the WebService Consumer**

The WebService Provider is a standard JEE web application. Download the most recent version from http://digitaliser.dk/group/705156and unzip the file. Deploy the poc-consumer.war file to GlassFish or to another Servlet container by following the same instructions as for the STS. This can be done in the same GlassFish instance, or on an instance on another host.

Configure the application for an Identity Provider (IDP)

It is a prerequisite for this step that an IDP is installed and available - the IDP must support SAML 2.0 Assertions (urn:oasis:names:tc:SAML:2.0:assertion version 2.0), but it is out of scope for this document to describe how to install and configure one. This scenario has been tested with SimpleSAMLphp which is freely available from [http://simplesamlphp.org](http://simplesamlphp.org/).The IdP should be configured to map from a given STS Entity ID to a TokenService URL of an STS, and to provide it as an EndPointReference in the User Assertion. This is the value which will be inserted into the Issuer field of the generated assertion, because the STS is a very simple implementation that just uses the IDP to issue a SAML token. In production mode the IDP will not have any associations to the STS - the STS must however know which IDP’s it trusts.The OIOSAML.java filter is configured for the application by performing the following steps:

* Download or acquire the metadata for the IdP and save it in a file (e.g. idp-metadata.xml)
* Enter the URL <http://localhost:8080/poc-consumer/saml/configure>, which - for an unconfigured application - will yield a page for providing the configuration. If the page claims that the application is already configured, but you wish to reconfigure, then remove (i.e. either delete or move) the directory ~/.oiosaml-poc-consumer, restart the application and try again.
* Fill in the configuration page like this:Entity-ID: http://saml.poc-consumer.localhostIdentity provider metadata: <path to IDP metadata file>Create new self-signed keystore? YesKeystore password: Test1234Organization Name: <Organization name>Organization URL: <Organization URL>Technical email contact address: <some email address>Enable artifact consumer? YesEnable Redirect consumer? YesEnable SOAP Single Logout? YesEnable OCES Attribute Profile? Yes
* Click the “Configure system” button
* Now the folder ~/.oiosaml-poc-consumer has been created and populated with a number of files and directories.
* Configure the IdP with knowledge about the poc-consumer metadata. This is to be found in ~/.oiosaml-poc-consumer/metadata/SP/SPMetadata.xml.
* For the SimpleSAMLphp IdP this requires converting the XML to PHP code (with a conversion service that is available in SimpleSAMLphp) and afterwards inserting this code into metadata/saml20-sp-remote.php file along with these extra attributes:  'attributemap' => 'oiosaml', 'NameIDFormat' => 'urn:oasis:names:tc:SAML:1.1:nameid-format:X509SubjectName',   'SPNameQualifier' => '',   'simplesaml.nameidattribute' => 'urn:oid:0.9.2342.19200300.100.1.1',
* Edit the file ~/.oiosaml-poc-consumer/oiosaml-sp.properties and add these properties:oiosaml-trust.bootstrap.base64=falsepoc.provider=http://localhost:8080/poc-provider/ProviderService oiosaml-trust.certificate.location=sts.jks oiosaml-trust.certificate.password=Test1234
* Export a certificate from key private key of the STS (which was placed in the keystore ~/.oiosaml/TestVOCES1.pkcs12) and import the certificate into a new keystore called ~/.oiosaml-poc-consumer/sts.jks. The filename and password of the new keystore should match the value of the properties oiosaml-trust.certificate.location and oiosaml-trust.certificate.password in ~/.oiosaml-poc-consumer/oiosaml-sp.properties.

keytool -exportcert -keystore ~/.oiosaml/TestVOCES1.pkcs12 -storetype pkcs12 -alias “<alias>” -file sts.crtkeytool -importcert -keystore ~/.oiosaml-poc-consumer/sts.jks -file sts.crt -alias sts -storepass Test1234

**Testing the integration of the applications**

**Test 1 - basic flow**

* Enter the URL http://localhost:8080/poc-consumer/⇒ The start page of the WSC shows
* Click the link “Page requiring login”⇒ The browser redirects to the login of the IdP (unless a login has already been performed)
* Perform login(In the case of using SimpleSAMLphp with the opensign module as authsource, the OpenSign applet shows and the user can browse for a private key file, e.g. TestMOCES1.pkcs12 and login using that) ⇒ Upon successful login, the browser gets a SAML User Assertion in a cookie and is redirected back to the poc-consumer which shows the assertion and a few more details
* Click the link in the bottom of the page “Call Service Provider with token”⇒ A new page shows the EndPointReference and the Token (a new SAML assertion) from a request to the STS
* Click the link in the bottom of the page “Perform token WS request”⇒ The consumer shows a page with data from the WS request to and response from the WSP. By default, the response is “null” but you may add “?length=3”  to the URL of the page, and a real response object is generated (although the only thing visible is the default output of the toString() method). Note that a larger length (e.g. 6) generates an exponentially larger result object, so larger number can easily result in a very long response time, and possibly an OutOfMemoryError. (The length request parameter defines the depth of a dummy object graph).

**Test 2 - request to interact**

* Enter the URL http://localhost:8080/poc-consumer/⇒ The start page of the WSC shows
* Click the link “Trigger Request to Interact”⇒ The browser redirects to the login of the IdP (unless a login has already been performed)
* Perform login (as in Test 1)⇒ Upon successful login, a page is shown that explains that extra information from the user is required to complete the request, and a more extensive link is provided
* Click the long link (which includes a ReturnToURL parameter)⇒ The browser is redirected to a page of the WSP prompting the user for a 'secret'
* Enter some information in the field (anything, e.g. “1234”) and click the “Send” button⇒The browser is now redirected back to a page of the WSC which shows the information that the user provided to the WSP.

In a real scenario, the WSP would probably not give away the secret info from the user to the WSC, but this is a simple way of simulating that the user approved the operation; ie. password or one-time-key has been entered by the user to the WSP, and thereby authorized the WSC to complete a certain webservice call at the WSP.

## Configuring the .NET components

Please note, that there is no .NET STS included in the package. The Java STS could be used instead.

### DLL Setup

Due to missing support for sp:ProtectTokens in WCF, Microsoft has released a few hotfixes for this particular issue. System.ServiceModel and System.IdentityModel have been patched to include this feature. Please note, though that these fixes might be considered non-standard with regards to MS-support. The hotfix includes support for UseStrTransform, which has been included in later WCF releases though, so please note, that you might have a potential DLL version conflict in the future on this particular issue!

The hotfixes cannot be downloaded from Microsoft, but are available on the OIOIDWS group on digitaliser.dk: http://digitaliser.dk/group/705156

### Webservice provider

In this example, the webservice provider is hosted by an IIS on Windows Vista/Windows 7, but it should be possible to host in WAS too. It requires that both .NET 3.5 framework and Windows Identity Foundation 1.0 (WIF) are installed.

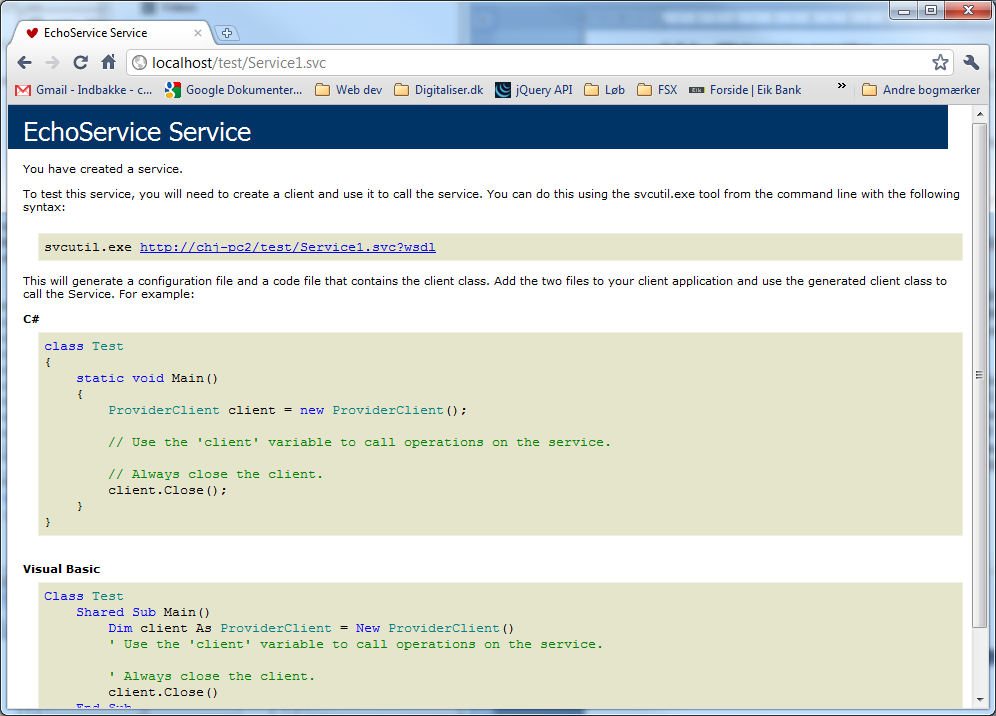
The steps described in this section explains what needs to be done on an IIS 7 running on a Windows 7. First, the Echo webservice itself should be configured:

* Open the IIS Manager
* Under Sites, “Default Web Site”, right click and select “Add Application”
* In the Alias field type “test”
* In the physical field, select the path to the EchoServiceProvider folder.
* Leave the application pool as Default

Now the user executing the webservice should install the STS certificate and grant the Echo service access to the STS certificate. This is done inside the MMC (Microsoft Management Console), File, Add/Remove Snap-in. In the list of available Snap-ins, select “Certificates”, click “Add >”, select “Computer account” and then finally click the Ok button.

Under the Console Root folder, a folder named “Certificates (Local Computer)” should be visible. Here, open the folders Personal, right click on the Certificates folder, select “All tasks” and “Import...”. Point to the STS.pfx file, “click Next >”, use Test1234 as password and enable “Mark this key as exportable”. Then click “Next >”, “Next >” and “Finish”. Then the certificate is displayed in the list of personal certificates. Right click on the certificate, and select “All task > Manage Private Keys”. Inside the permissions dialog, click “Add”, type IIS\_USRS and click “Ok” twice, and the service is able to access the certificate.

Check that the webservice is alive by entering <http://localhost/test/Service1.svc> in a browser. If the service and certificate are configured correctly, the browser shows a page like this:



### Rich client

The rich client requires .NET 3.5 framework and Windows Identity Foundation 1.0 (WIF). The STS certificate should be installed in the Local Machine certificate store as described in the web service provider section above, in order to be able to select it in the dropdown-box. A few settings in WPFClient.exe.config can be configured:

* Configures the default values for the 3 textboxes in the application:  
  <applicationSettings>

<WPFClient.Properties.Settings>

<setting name="LocalSTSUrl" serializeAs="String">

<value>https://tri-test6.tritest6domain.com/adfs/services/trust/2005/usernamebasictransport</value>

</setting>

<setting name="STSUrl" serializeAs="String">

<value>http://localhost:8082/sts/TokenService</value>

</setting>

<setting name="WSUrl" serializeAs="String">

<value>http://localhost/test/Service1.svc</value>

</setting>

</WPFClient.Properties.Settings>

</applicationSettings>

* Configuration of WIF logging, using normal .NET trace listener configuration:

<source name="System.ServiceModel"

switchValue="Verbose, ActivityTracing"

propagateActivity="true">

<listeners>

<add name="sdt"

type="System.Diagnostics.XmlWriterTraceListener"

initializeData= "c:\temp\wiftrace.e2e" />

</listeners>

</source>

### IDP using AD FS 2.0

Configuring an ADFS is beyond the scope of this document, just a few overall notes on how to use it in our scenario though.

Instead of creating our own bootstrap token, it is possible to use AD FS to issue these tokens. The AD FS in that case takes the role as an IDP that authenticates users by means of username/password, Windows Authentication token, X509-certificate etc.

In case you want to test the rich .net client with an ADFS 2.0 server, it requires a Windows Server 2008 R2 with ADFS 2.0 Server installed as add on to an Active Directory installation. There are several ways in which you can authenticate against the ADFS, in the sample code two examples of authentication are implemented: Windows authentication and username/password. Depending on which method should be used, you should modify the Window1.xaml.cs file and thereby use the appropriate GetTokenFromSTSUsingXXX method (in the buttonWS\_Click() method).

In the example code, an ADFS server is installed on the URL <http://tri-test6.tritest6domain.com>. Depending on the binding used for obtaining the bootstrap token from the ADFS, the url should be modified according to the binding.

On the ADFS, a signing certificate must be installed, and the public key of this signing certificate should be available for the rich client in the Windows certificate store. In the sample source code, the signing certificate is the one with the subject CN=ADFS Signing - tri-test6.tritest6domain.com.