# WS-Security unveiled

*One of the more complex examples that ship with WLS is a WS-Trust based authentication of a web service using SAML assertions. This example is quite easy to setup and run and the example's documentation gives a basic understanding of what is going on. But we want to look deeper and shed light on the complexity that is hidden behind SSL, WS-Trust and SAML authentication. In a first step we separate the scenario from the example server and integrate it into eclipse, to create an isolated laboratory environment for further investigation. We will use Wireshark to analyze the actual flow of messages on the wire. We even look inside the SSL streams to identify the WS-Trust tokens and SAML assertions as they are passed between the participants.*

## 1 Contents

1 Contents ... 1
2 Introduction ... 1
3 Modified Example Setup ... 2
3.1 Analysis of the example build process ... 3
3.2 Modification to the original example ... 4
3.3 Setting up the modified example ... 6
3.3.1 Setting up Eclipse Project on Mac OSX Lion.: ... 6
4 Network traffic analysis ... 7
4.1 Configuring bridged networking in virtual box. ... 8
4.2 Configuring the server side. ... 9
4.3 Setting up the client side. ... 9
4.4 Configuring and running Wireshark. ... 10
4.5 Analysis ... 13
4.5.1 SSL Stream 0 ... 17
4.5.2 SSL Stream 1 ... 18
4.5.3 SSL Stream 2 ... 20
5 Conclusion ... 22
6 Links ... 23

## 2 Introduction

The WS-Trust specification, which is part of the WS-* stack of specifications for web services, was approved as OASIS standard in March 2007. Meanwhile it is generally accepted as an industry standard for implementing secure, trusted, and federated message exchange between service providers and consumers. So now is a good time to have a closer look at this technology. Despite the complexity of this topic, it is fairly easy to set up running examples of WS-Trust based java implementations using the examples that ship with Weblogic Server. We want to have a closer look at one of these examples: "Using SAML 1.1 Bearer Assertion for Authentication *Case*"

In this example we use the Weblogic implementation of the WS-* standard which is based on the JAX-WS Reference Implementation (RI)[1]. In this example there is a client that requests a SAML assertion from a Secure Token Service (STS) to authenticate for using a web service. The following picture illustrates this scenario.

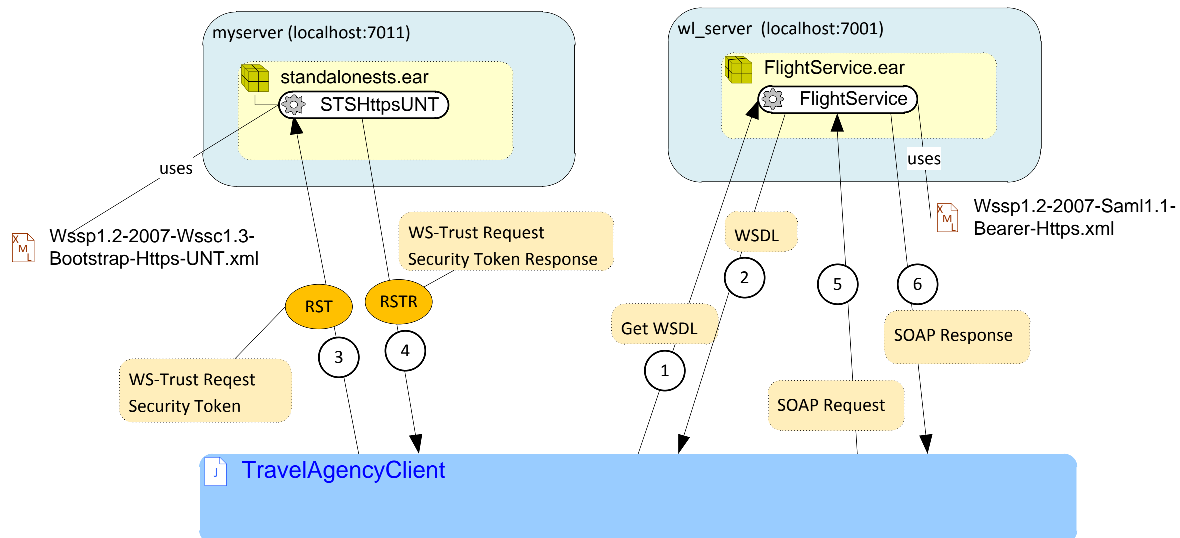


Figure 1. **Overview of the SAML 1.1 Bearer Assertion for Authentication Example.**

The WLS example server with the name wl_server is hosting the FlightService web service. This web services is protected with a WS-Security Policy. In a first step the Java standalone client connects to the wl_server and requests the WSDL of the FlightService, which is returned in the response. The client initializes a webservice port with this WSDL. The web service implementation on the client side, on parsing the WSDL, recognizes that a SAML Authentication token is required. In step 3 it sends a RST protocol message to the server, running the STS. The STS Service authenticates the client, issues a SAML assertion and returns it in a RSTR protocol message in step 4. The client uses the SAML Assertion in the SOAP request to the web service in step 5. Finally in step 6, the wl_server validates the SAML assertion and returns the response of the FlightService in a SOAP response message.

Documentation of this example is contained in the example server. It further explains the interactions of this scenario and describes the steps to set up this example and the files needed as java source code as well as for configurations. After starting our example server wl_server we can read the example documentation with the webbrowser.[2] For convenience we provide a pdf-capture of this description[3].

# 3 Modified Example Setup

The examples documentation contains instructions to build and run the example from a command line. While this could serve as a good proof of concept, it is not the ideal environment for further analysis. We want to integrate and run the example from within eclipse which allows for easy inspection of various files. In contrast to most of the other wls api examples the configuration here is rather complicated. As we will see in the next section,

[1] See http://docs.oracle.com/cd/E24329_01/web.1211/e24963/overview.htm

[2] Link on local the example server to the example description: http://localhost:7001/docs/server/examples/src/examples/webservices/saml/bearer11ssl/instructions.html?skipReload=true

[3] Documentation of the example as PDF: https://dl.dropbox.com/u/16989587/weblogic-corner/WS-Security_unveiled/WLS_Example_SAML_Bearer11ssl.pdf

this examples uses two separate WLS servers in separate WLS domains. One of them is the pre-configured example server wl_server, which is additionally running all the other examples. Thus, configuration changes made during the configuration of the SAML_Bearer11ssl example cannot easily be distinguished from the rest of the example server's configuration. Therefore we want to modify the example to isolate it from the wl_server.

## 3.1 Analysis of the example build process

If we follow the steps of the example instruction, we arrive at the running system after a couple of ant build commands. Since there is a lot happening in these commands we need to look deeper into this process to understand it.
The following figure illustrates the ant build process for this example.

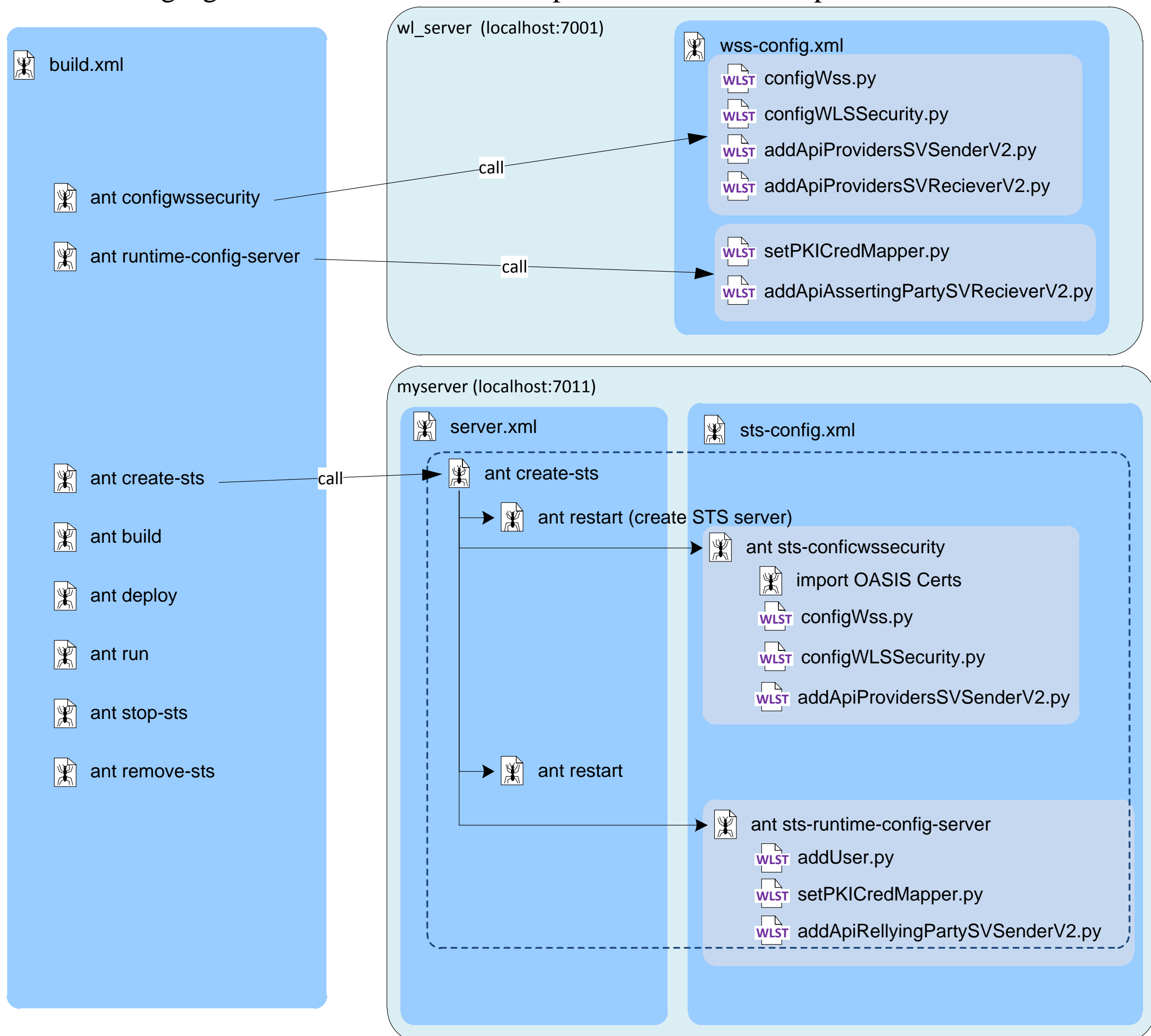


Figure 2. **Build Structure of the SAML 1.1 Bearer Assertion example.**

The deep blue box on the right shows the general build.xml and the ant targets that have to be called on this file to build and run everything. These targets, in turn, call targets imported from additional ant build files. In some of these targets we use the Weblogic Server Scripting Tool (WLST) to connect to running server instances and do configurations or deployments. In the light blue shaded boxes we depict the Python scripts with which WLST is called.
Looking at the whole process, we recognize that the targets *configswssecurity* and *runtime-config-server* operate on the pre-existing example server wl_server and the target create-sts

and the targets of that call chain operate on the STS server “myserver”. The domain of myserver is created in the course of this configuration. Further investigation reveals that activities on both servers are quite similar and reuse a subset of the same ant files.
In order to separate the example from the wl_server we want to modify the configuration process to also create the WSS server that is running the flight service, thus replacing the wl_server.
Let’s have a look at these modifications in the next section.

## 3.2 Modification to the original example

In this section we point out the modifications necessary to separate this example from wl_server. Prior to these activities we integrate the whole example into eclipse put we don’t provide the details here, because we will offer the resulting eclipse project for download. It can be imported into eclipse as described in 3.3 unterhalb.
The general idea of the modifications is to create and configure the WSS server in the same manner as the STS server
We have to implement the following steps:

- change name and ports for wl_server to wss_server, localhost:8011, ssl-port:8012 (in examples.properties)
- implement ant target create-wss in server.xml
- implement ant target wss-configwssecurity in wss-config.xml
- implement ant target wss-runtime-config-server in wss-config.xml

The domains and servers are created by starting them with the flag createconfig=true.
No domain names and server names are supplied to this process. Thus they default to mydomain and myserver for both domains. We distinguish them by generating them into different domain roots: wss_stage and sts_stage
The following table summarizes the relevant URLs and Logins for these servers.

| | | |
|---|---|---|
| sts-server | http://localhost:7011/console<br>https://localhost:7012/console | user=system<br>password=gumby1234 |
| wss-server | http://localhost:8011/console<br>https://localhost:8012/console | user=weblogic<br>passowrd=welcome1 |

**Table 1. URLs and Logins fort he sts-server and wss-server.**

The resulting build and configuration process structure is depicted in the following figure.

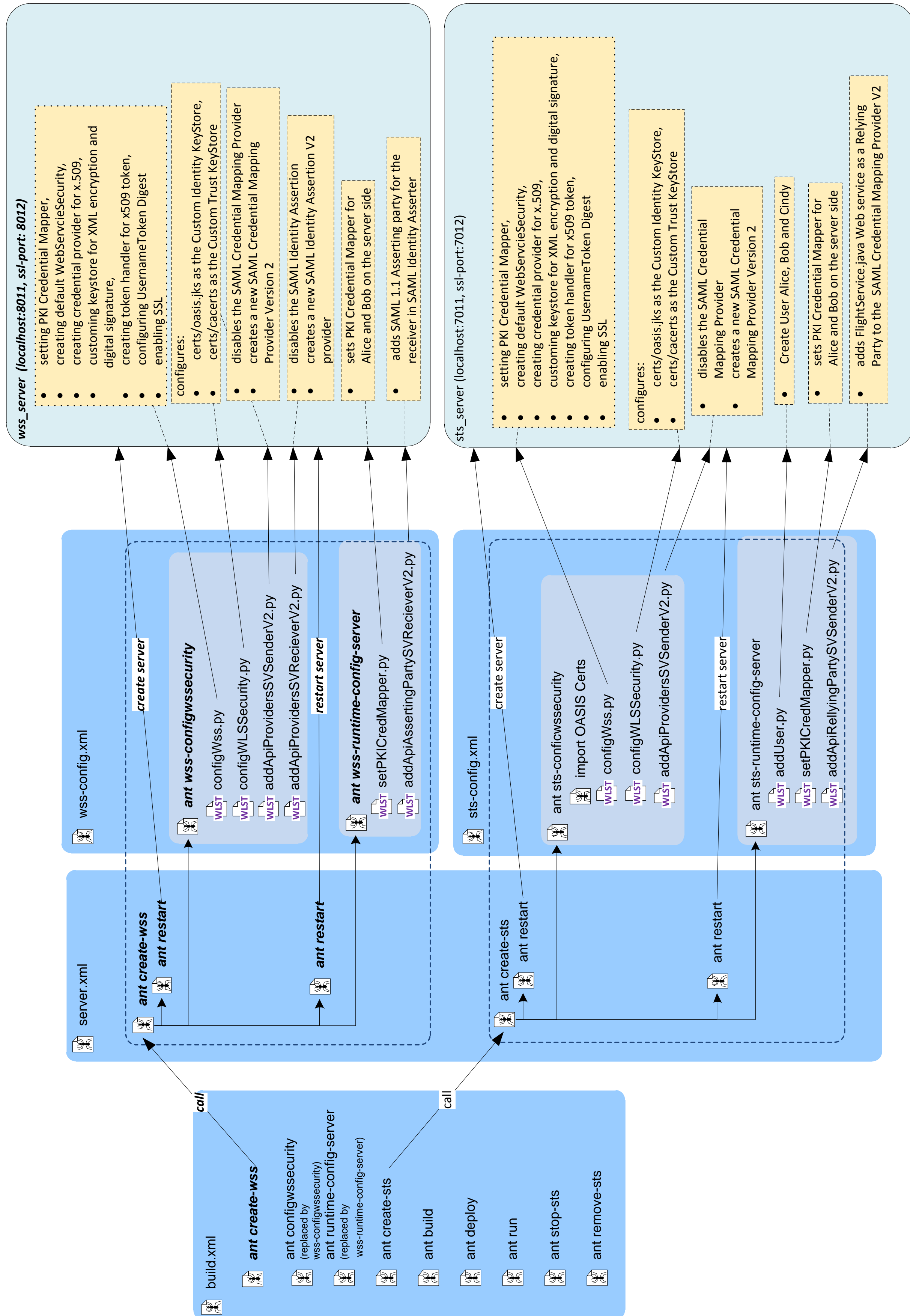


Figure 3. **Structure of the modified ant build.**

In the right box symbolizing the file build.xml we introduced the new ant target create_wss which is actually an import from the file server.xml. We create the WSS server in the same manner as the STS server by using the ant task wl_server with the create_config option set. In the first step of this process we restart the Weblogic server, i.e. we stop it first to prevent any conflicts from previous runs. In the newly introduced file wss-config.xml we first call the ant target wss-configssecurity which replaces the target configwssecurity from build.xml. The figure shows which Python scripts are called here and what configurations result in the server. The process restarts the WSS server, which is necessary for some changes to take effect. The following ant task wss-runtime-config-server replaces the ant target runtime-config-server from build.xml. It calls two Python scripts to complete the server's configuration. After execution of this target the server remains running and since its process is spawned from within ant, the ant target does not terminate.
The create-sts target follows the same execution pattern, however with different configuration settings in the file properties.txt and some calls to different Python files.
After both servers are configured and running, we use the ant targets build and deploy, which will build all Java artefacts and deploy them to the servers. The client is built as well.
The run target simply starts a Java process and runs the Java client, which executes the whole example.

## 3.3 Setting up the modified example

We offer the modified example as exported eclipse project as *saml_bearer11ssl.zip*[4].
To set up the example we download the zip archive and import it as an eclipse project. Depending on the platform we are using, we have to make some adjustments. After that, we run the example completely from within eclipse.
Here are the instructions to set up the saml_bearer11ssl.zip on MacOSX Lion. Alternatively the project has been tested on Windows 7, 64-bit and Oracle Enterprise Linus version 5.5.

### 3.3.1 Setting up Eclipse Project on Mac OSX Lion.:

From within eclipse we execute the following steps.

1. Import->General->Existing Project into Workspace
   Select archive file: saml_bearer11ssl.zip
   Projects (Select All)
   Finish
   We can ignore the Python Interpreter Configuration Questions.

2. Open example.properties and adjust:

```
bea.home=/Users/uAries/Oracle/Middleware
wl.home=/Users/uAries/Oracle/Middleware/wlserver_12.1
examples.classpath=/Users/uAries/Oracle/Middleware/wlserver_12.1/server/lib/weblogic.jar
# The endorsed directory is need on MacOS (only needed on MacOS)
extra.jvm.options=-Djava.endorsed.dirs=${wl.home}/endorsed
```

3. Modify server.xml to account for the endorsed directory (only needed on MacOS). We need to add the property extra.jvm.options to the start of the jvm which happens two times:

a. in target debug-server-macro (only needed for debugging)

```
<jvmarg line=-ms${wls-min-stack} -mx${wls-max-stack} -XX:MaxPermSize=128m ${extra.jvm.options}
```

[4] Zip archive of the eclipse project: https://dl.dropbox.com/u/16989587/weblogic-corner/WS-Security_unveiled/saml_bearer11ssl.zip

b. in target start-server-macro (only needed on MacOS):

```
<wlserver dir="${dir}" host="${host}" port="${port}" generateConfig="true"
username="${username}" password="${passwd}" action="start">
<jvmarg line="-classpath ${examples.classpath} ${extra.jvm.options} -ms${wls-min-stack} -
mx${wls-max-stack} -XX:MaxPermSize=256m -Dweblogic.wsee.wls.home=${env.WL_HOME} -
DCommonSecurityEnabled=true -Dweblogic.wsee.verbose=${weblogic.wsee.verbose} ${extra-server-
verbose}
```

4. We modify server.xml in target start-server-macro. We may need to increase the value of MaxPermSize (only needed on MacOS)

```
<wlserver dir="${dir}" host="${host}" port="${port}" generateConfig="true"
username="${username}" password="${passwd}" action="start">
<jvmarg line="-classpath ${examples.classpath} ${extra.jvm.options} -ms${wls-min-stack} -
mx${wls-max-stack} -XX:MaxPermSize=256m -Dweblogic.wsee.wls.home=${env.WL_HOME} -
DCommonSecurityEnabled=true -Dweblogic.wsee.verbose=${weblogic.wsee.verbose} ${extra-server-
verbose}
```

5. In Project Properties->Java Build Path->Libraries (only needed for auto completion)
   Adjust your path to include weblogic.jar
6. Now we can run the following ant targets directly from within eclipse from the file build.xml. (Outline view->right mouse click on chosen target->run as->ant build)
   - create-sts (leave the server running)
   - create-wss (leave the server running)
   - build
   - deploy
   - run

For reference we provide the output of all five successful executions in the file saml_bearer11ssl.ouput.txt[5].

# 4 Network traffic analysis

Now that we have prepared our laboratory environment, we want to use it and look at the actual traffic on the wire using Wireshark. Our Hardware equipment consists of single laptop, running Windows 7 Professional 64-Bit. It has an Intel i7 2,4 GHz Quadcore CPU and has installed memory of 20 GByte, so it is strong enough to drive an Oracle Enterprise Linux Server version 5.5 inside Virtual Box. We use the appliance VDD_*WLS_labs_2012.ova* which can be freely downloaded from Oracle. Details about this appliance can be found on the Oracle website[6]. We use this appliance because there is WLS12c and Eclipse already installed, so we only need to import the saml_bearer11ssl eclipse project and run it. We also use the same eclipse project on the server side but we change the listen address of the servers from localhost to the actual ip-address of the machine. We then modify the client running in the Linux virtual box to communicate with the WLS servers on Windows. Using these two different machines makes it easier to capture the traffic with Wireshark, because we can apply a capture filter that only captures traffic between the ip-addresses of these two machines. The following picture illustrates this test setup.

[5] Console output text file: https://dl.dropbox.com/u/16989587/weblogic-corner/WS-Security_unveiled/saml_bearer11ssl.output.txt
[6] Details about the appliance VDD_*WLS_labs_2012.ova:*
http://www.oracle.com/technetwork/middleware/weblogic/downloads/weblogic-developer-vm-303434.html

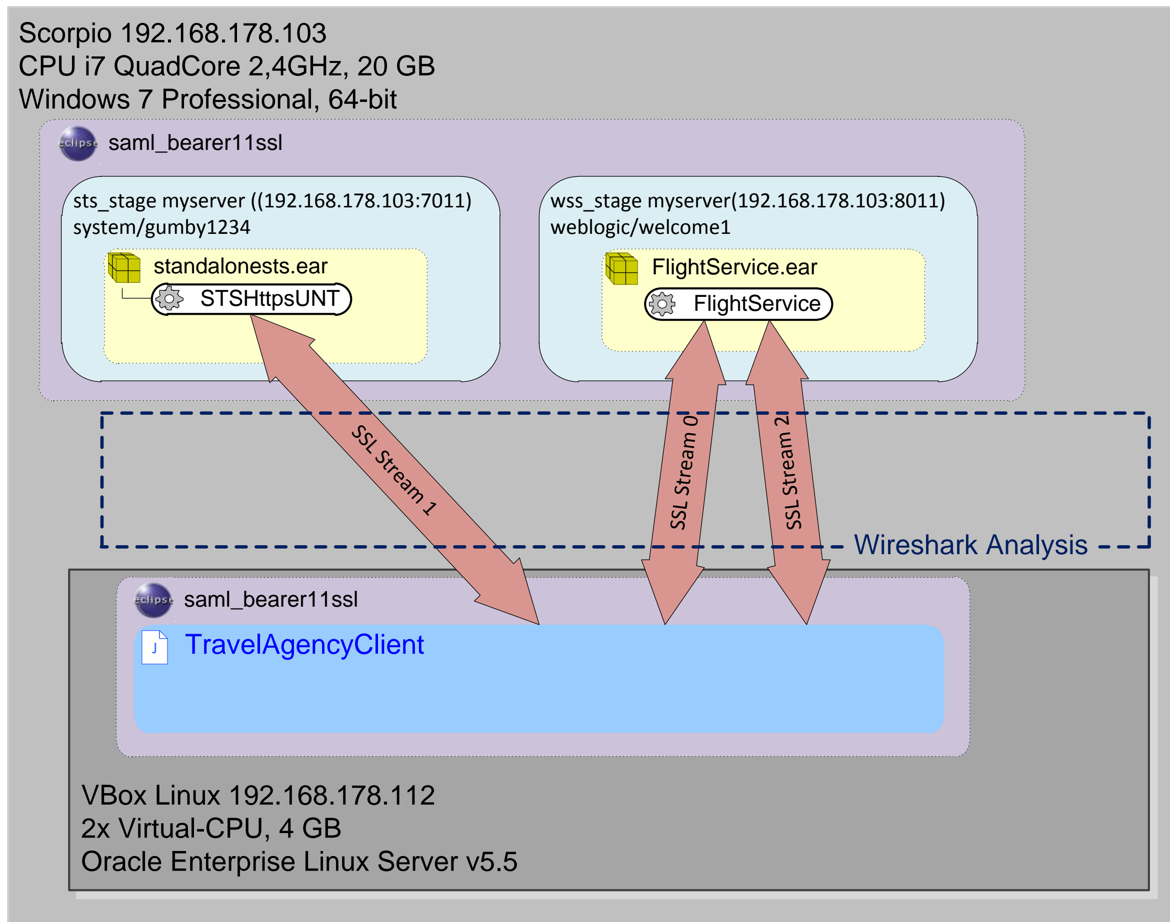


Figure 4. **Test Setup for Wireshark Analysis**

As we will see later, there are three SSL data streams that are revealed by the network traffic analysis. Stream 0 contains the initial communication between the client and the WSS server with a request to retrieve the WSDL of the Flight Service. Stream 1 contains the communication between the client and the STS server to retrieve the SAML assertions. And in stream 2 the client sends a SOAP request containing the SAML assertion and receiving the web service response.
Before diving into the details of this analysis, let’s have a look at the configurations of our test environment.

## 4.1 Configuring bridged networking in virtual box.

The Linux guest system in the virtual box needs to communicate to the windows host system via the network. Among the different network options offered by virtual box “bridged networking” satisfies this requirement. The following picture shows this network configuration.

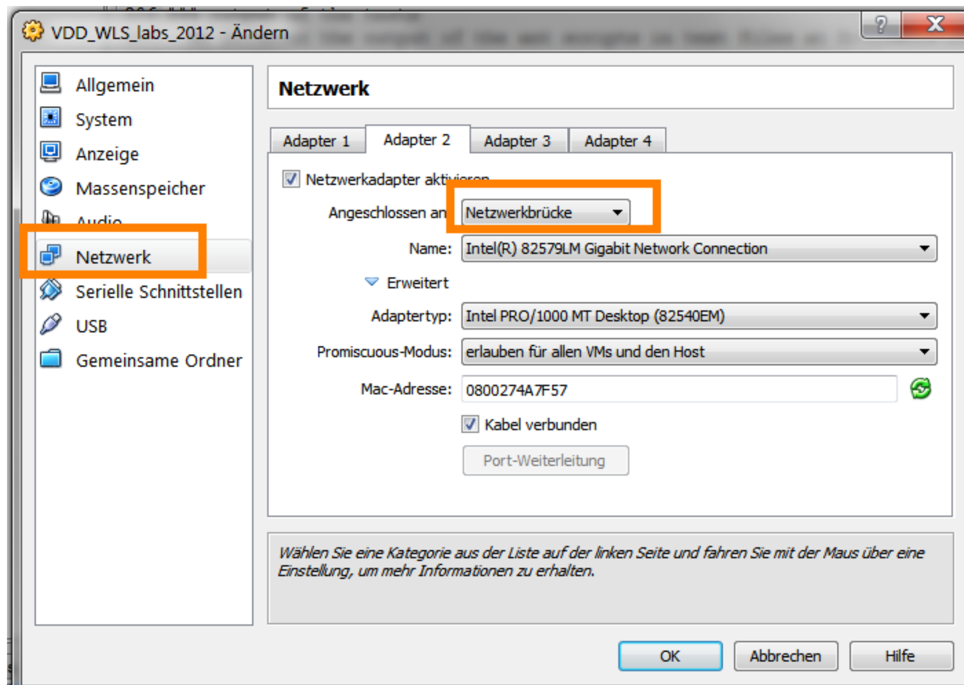


Figure 5. **Configuring bridged networking in Virtual Box.**

In this setting the network interface will get an IP address from the DHCP server of the local network.

## 4.2 Configuring the server side.

Both Weblogic servers will be running on the Windows system. We have to change the host address from local host to the current IP address. Therefore we edit the file properties.txt in eclipse. We change the settings of the following properties. The number indicates the line number.

```
11: wls-host=192.168.178.103
 18: sts-wls-host=192.168.178.103
 66: remote.host=192.168.178.103
```

After that we rebuild the whole example by executing the ant targets: create-sts, create-wss, build, deploy and run.

## 4.3 Setting up the client side.

The Java Client is running on Linux in the guest system. The modified example saml_bearer11ssl is installed as eclipse project. Now we need to redirect the java client to talk to the Weblogic servers on the Windows machine.
In the file properties.txt we make the following modifications:

```
#flightServiceURL=https://${wls-ssl-server}${flightServicePath}
flightServiceURL=https://192.168.178.103:8012${flightServicePath}

#samlStsURL=https://${sts-wls-host}:${sts-sport}/standalonests/SamlSTS
samlStsURL=https://192.168.178.103:7012/standalonests/SamlSTS
```

We have to run the ant targets build to rebuild the client.

## 4.4 Configuring and running Wireshark.

For the analysis of the network traffic we use Wireshark. We download and install the file Wireshark-win64-1.8.0.exe from the website http://www.wireshark.org/download.html. Wireshark requires loading the WinPcap drivers called NPF on windows systems. There are various options for loading these drivers[7]. We start a windows command shell with the "run as administrator" options. Then we use the following commands for starting and stopping the drivers.

```
net start npf
net stop npf
```

We have to start the driver before starting Wireshark.
In Wireshark we select the capture interface and configure a capture filter.
Menu->Capture->Options->Double-Click on Device with ip 192.168.178.103
Dialog: Edit Interface Settings->Capture Filter
Dialog: Wireshark: Capture Filter - Profile: Default->Properties:
Filter name: Between VDD_WLS_labs_2012 and Scorpio
Filter string: (src net 192.168.178.103 and dst net 192.168.178.112) or (src net 192.168.178.112 and dst net 192.168.178.103)
This will restrict the captured packages to the traffic between the Windows Host and the Linux system. The following picture shows these settings.

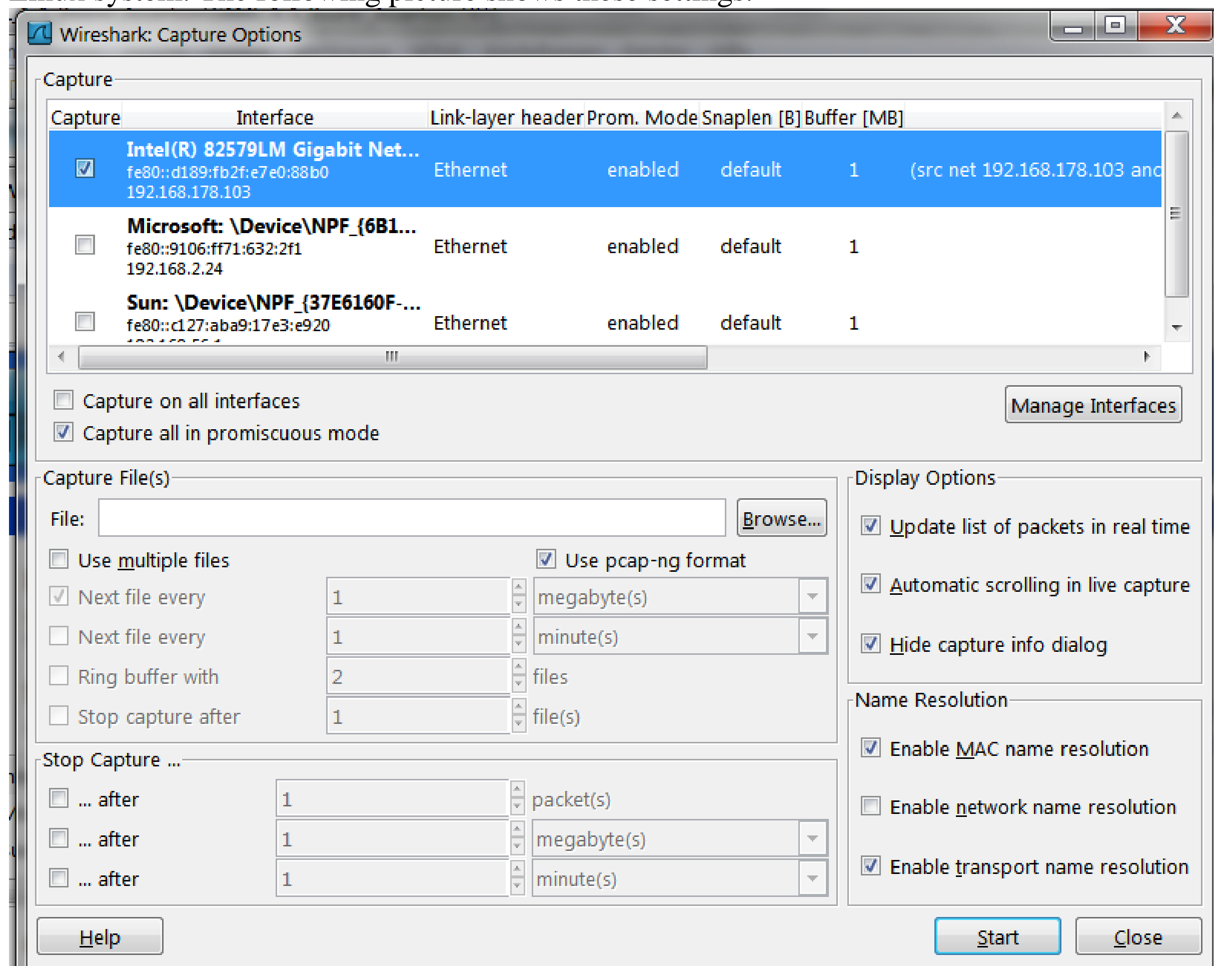


Figure 6. **Capture Options Dialog in Wireshark**

Since we also want to look into the SSL Streams we have to provide the keys to decrypt the SSL stream. There are two SSL connections that we need to analyse. One is the connection

[7] Information on WinPcap drivers: http://wiki.wireshark.org/WinPcap

between then WSS-server and the client and the other is the connection between the STS-server and the client. In both cases we use one-way SSL, i.e. the server is presenting its identity to the client. The keys to encrypt the SSL communication are the identity keys of the servers.
We can list the contents of the keys using the java keytool.

```
##Set the environement to access the keytool:
cd D:\11Eclipse\workspace01\saml_bearer11ssl\bearer11ssl\certs
D:\10Oracle\01Middleware\wlserver_12.1\common\bin\commEnv.cmd


------------------oasis.jks
D:\11Eclipse\workspace01\saml_bearer11ssl\bearer11ssl\certs>keytool -list -keystore oasis.jks
-storepass password

Keystore-Typ: JKS
Keystore-Provider: SUN

Ihr Keystore enthõlt 6 Eintrõge.

alice, 19.03.2005, PrivateKeyEntry,
Zertifikatsfingerabdruck (MD5): 57:CE:81:F1:03:C4:2C:F7:5B:1A:DE:AC:43:64:0A:84
root, 19.03.2005, PrivateKeyEntry,
Zertifikatsfingerabdruck (MD5): 0C:0D:00:27:BF:4B:32:63:40:A8:B2:03:96:4B:58:14
ca, 19.03.2005, PrivateKeyEntry,
Zertifikatsfingerabdruck (MD5): CA:0A:6D:E3:A4:9F:E8:55:98:0A:F8:10:66:35:40:C6
bob1, 17.12.2008, trustedCertEntry,
Zertifikatsfingerabdruck (MD5): 89:3E:86:D2:4F:9C:E7:39:B6:71:8A:EF:00:C5:89:DC
bob, 19.03.2005, PrivateKeyEntry,
Zertifikatsfingerabdruck (MD5): 89:3E:86:D2:4F:9C:E7:39:B6:71:8A:EF:00:C5:89:DC
wssipsts, 16.03.2009, PrivateKeyEntry,
Zertifikatsfingerabdruck (MD5): 14:A6:7A:CA:3D:11:86:9C:62:C0:34:2E:E4:7C:C4:75
------------------oasis.jks

------------------cacerts
D:\11Eclipse\workspace01\saml_bearer11ssl\bearer11ssl\certs>keytool -list -keystore cacerts -
storepass changeit

Keystore-Typ: JKS
Keystore-Provider: SUN

Ihr Keystore enthõlt 16 Eintrõge.

thawtepersonalfreemailca, 12.02.1999, trustedCertEntry,
Zertifikatsfingerabdruck (MD5): 1E:74:C3:86:3C:0C:35:C5:3E:C2:7F:EF:3C:AA:3C:D9
thawtepersonalbasicca, 12.02.1999, trustedCertEntry,
Zertifikatsfingerabdruck (MD5): E6:0B:D2:C9:CA:2D:88:DB:1A:71:0E:4B:78:EB:02:41
certgenca, 22.03.2002, trustedCertEntry,
Zertifikatsfingerabdruck (MD5): 8E:AB:55:50:A4:BC:06:F3:FE:C6:A9:72:1F:4F:D3:89
verisignclass3ca, 29.06.1998, trustedCertEntry,
Zertifikatsfingerabdruck (MD5): 78:2A:02:DF:DB:2E:14:D5:A7:5F:0A:DF:B6:8E:9C:5D
wlsdemobcca1024, 04.11.2002, trustedCertEntry,
Zertifikatsfingerabdruck (MD5): A1:17:A1:73:9B:70:21:B9:72:85:4D:83:01:69:C8:37
thawtepersonalpremiumca, 12.02.1999, trustedCertEntry,
Zertifikatsfingerabdruck (MD5): 3A:B2:DE:22:9A:20:93:49:F9:ED:C8:D2:8A:E7:68:0D
thawteserverca, 12.02.1999, trustedCertEntry,
Zertifikatsfingerabdruck (MD5): C5:70:C4:A2:ED:53:78:0C:C8:10:53:81:64:CB:D0:1D
ca, 01.09.2005, PrivateKeyEntry,
Zertifikatsfingerabdruck (MD5): CA:0A:6D:E3:A4:9F:E8:55:98:0A:F8:10:66:35:40:C6
verisignclass4ca, 29.06.1998, trustedCertEntry,
Zertifikatsfingerabdruck (MD5): 1B:D1:AD:17:8B:7F:22:13:24:F5:26:E2:5D:4E:B9:10
certgencab, 04.11.2002, trustedCertEntry,
Zertifikatsfingerabdruck (MD5): A2:18:4C:E0:1C:AB:82:A7:65:86:86:03:D0:B3:D8:FE
verisignserverca, 29.06.1998, trustedCertEntry,
Zertifikatsfingerabdruck (MD5): 74:7B:82:03:43:F0:00:9E:6B:B3:EC:47:BF:85:A5:93
verisignclass1ca, 29.06.1998, trustedCertEntry,
Zertifikatsfingerabdruck (MD5): 51:86:E8:1F:BC:B1:C3:71:B5:18:10:DB:5F:DC:F6:20
thawtepremiumserverca, 12.02.1999, trustedCertEntry,
Zertifikatsfingerabdruck (MD5): 06:9F:69:79:16:66:90:02:1B:8C:8C:A2:C3:07:6F:3A
mykey, 01.09.2005, trustedCertEntry,
Zertifikatsfingerabdruck (MD5): CA:0A:6D:E3:A4:9F:E8:55:98:0A:F8:10:66:35:40:C6
verisignclass2ca, 29.06.1998, trustedCertEntry,
Zertifikatsfingerabdruck (MD5): EC:40:7D:2B:76:52:67:05:2C:EA:F2:3A:4F:65:F0:D8
wlsdemobcca, 04.11.2002, trustedCertEntry,
Zertifikatsfingerabdruck (MD5): 5B:10:D5:3C:C8:53:ED:75:43:58:BF:D5:E5:96:1A:CF
------------------cacerts
```

In the following picture we present the keystore configuration of the example.

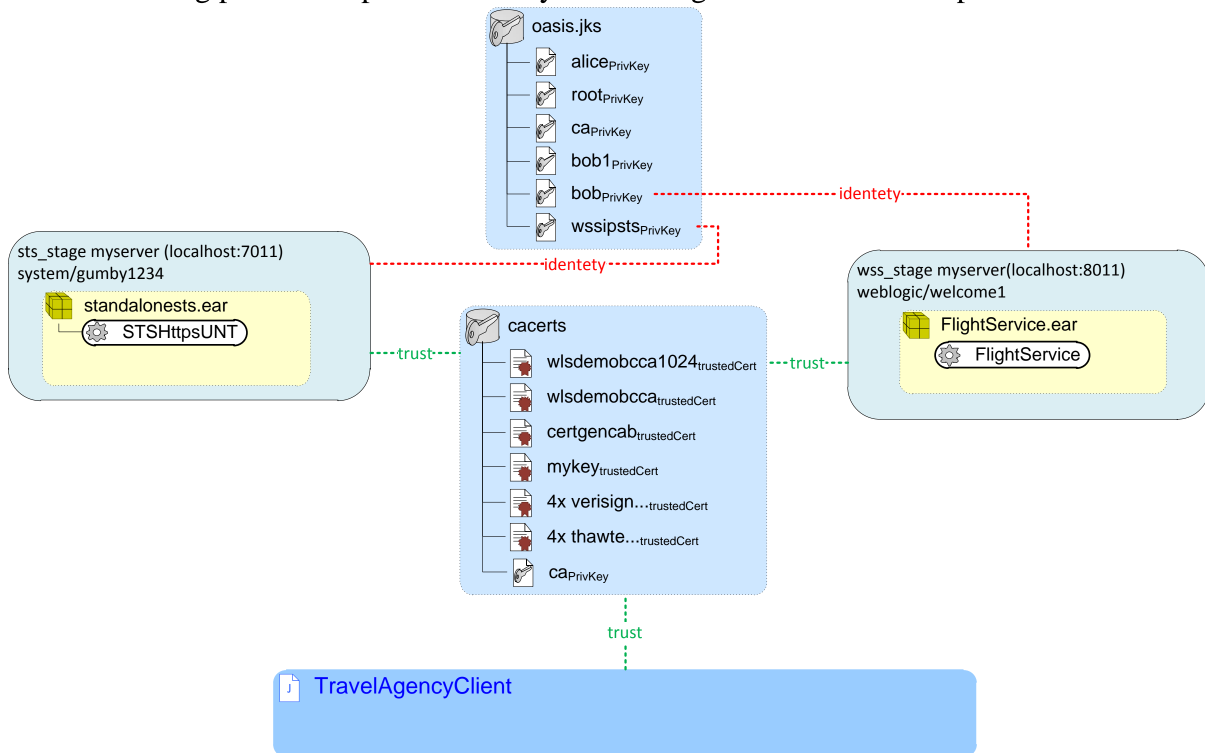


Figure 7. **Keystores used by the process for SSL communication.**

There are two java keystores used in this example. One for trust, containing trusted certificates and one for identity, containing private key entries. All three parties share the same trust store. The two server process share the identity store oasis.jks, but use different keys for server identity. The STS server uses the Key from wssipsts and the WSS-server uses bobs key. To decrypt the SSL stream, Wireshark needs access to the identitiy keys, which contain both, the private key and the public key. In the directory saml_bearer11ssl\bearer11ssl\certs we find these keys in PFX format which contains, both, the private and the public key. The PFX format is accepted by Wireshark and we need to configure these keys in the preference settings. From the *Edit* menu we open the preference dialog. On the left side under *Protocols* we choose *SSL*. On the left side we click the *Edit* button to edit the *RSA keys list*. This opens the *SSL Decrypt* dialog, where we click the *New* button and enter the keyfile and the details into the "*New Entry*" dialog. This is depicted in the following picture.

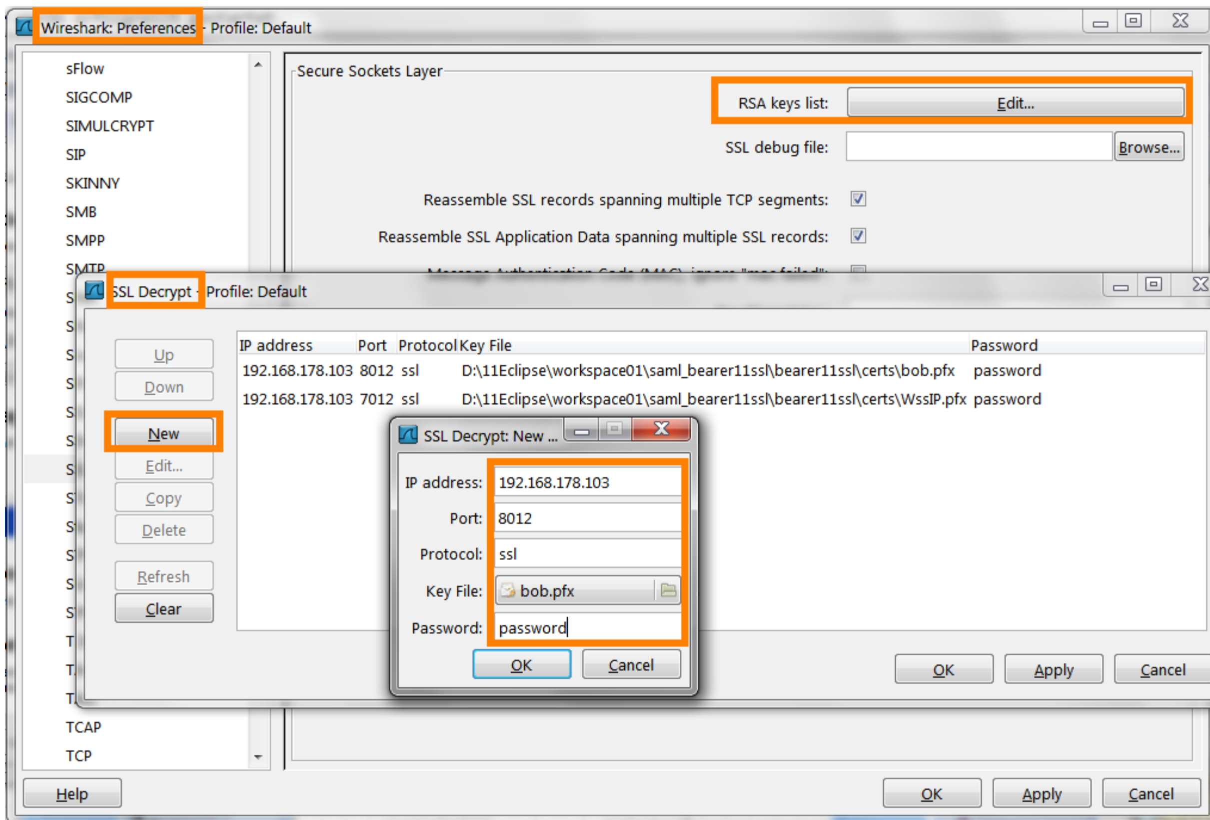


Figure 8. **Configuring keys for SSL decryption in Wireshark.**

We can find the passwords in properties.txt.
Now we are ready to run the analysis.

## 4.5 Analysis

We start the capture in Wireshark. Then we switch to Linux and start the Client by invoking ant run of build.xml within eclipse as depicted by the following picture.

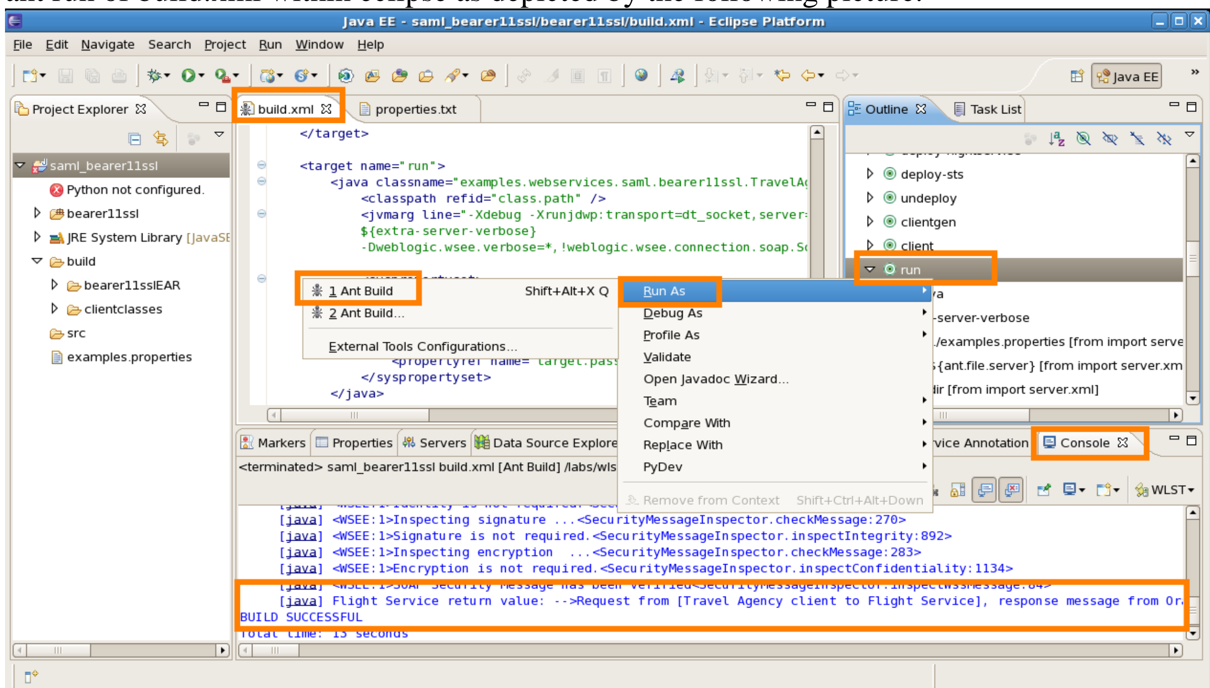


Figure 9. **Running the client on the Linux side.**

In the console we can observe that the scenario executed successfully. We stop the recording in Wireshark, to capture exactly one test run. The following picture shows the Wireshark windows after the successful capture.

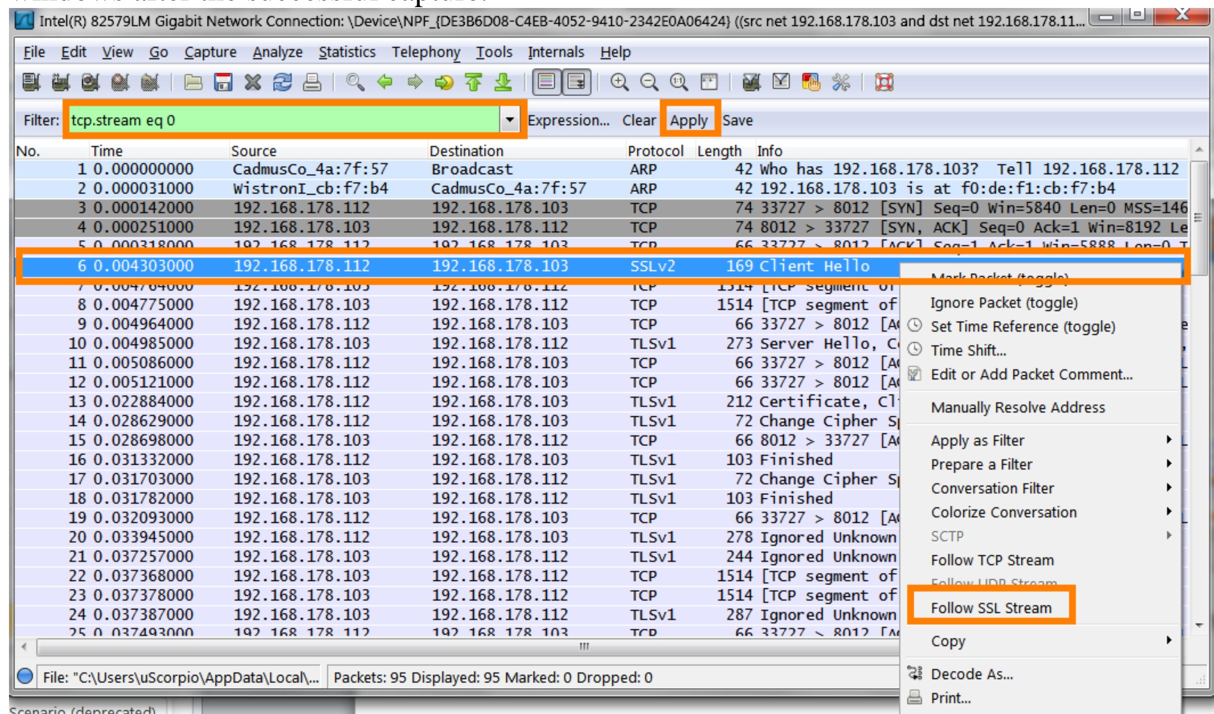


Figure 10. **Successful Capture of the Scenario in Wireshark.**

We can see in the main window that we have only captured the traffic between the ip-addresses 192.168.178.103 and 192.168.178.112, due to our applied capture filter. We apply an additional view filter with the dropdown box in the top, to filter out the separate TCP streams. We start with the filter “tcp.stream eq 0” and right click on the packet with “Client Hello” which indicates the start of the SSL Handshake. Since we have imported the SSL Keys of the servers, we can choose “Follow SSL Stream” from the dropdown box which would be grayed out otherwise.

This opens the first SSL-Stream in an extra window, as depicted in the following picture. We proceed analogous with the two other SSL streams.

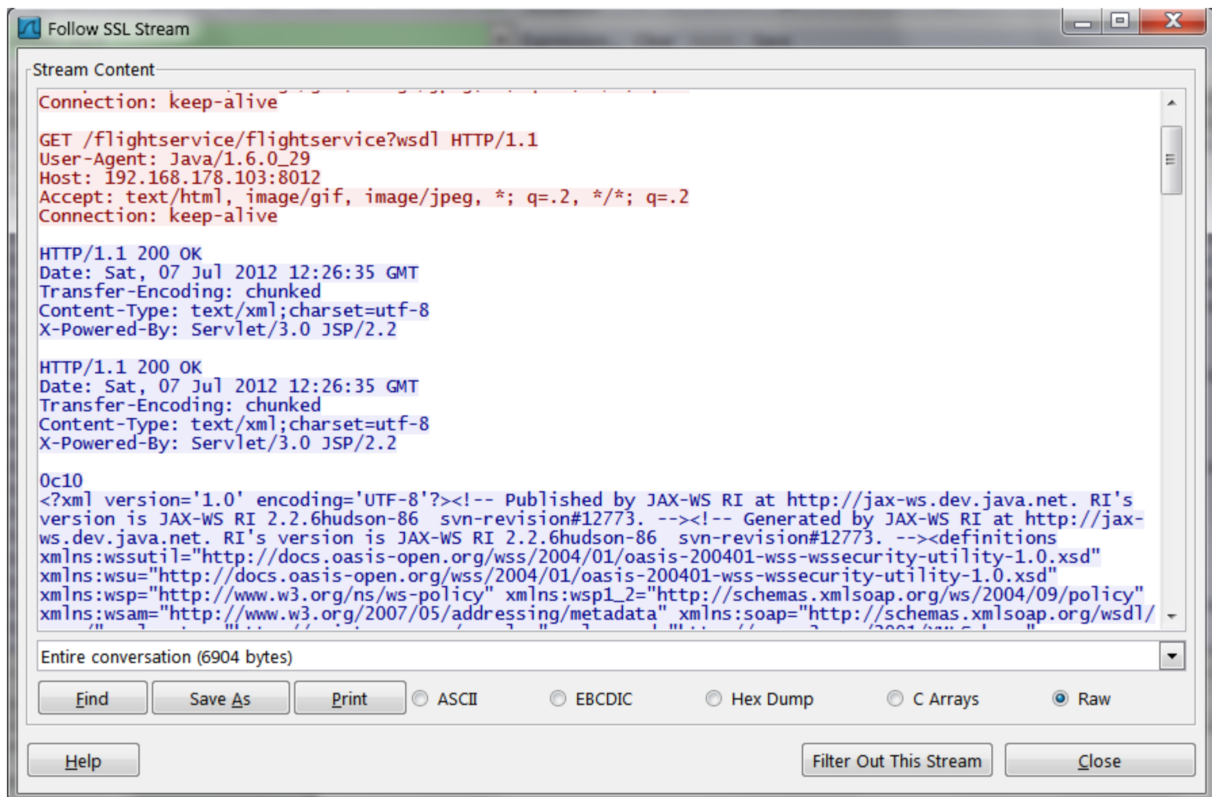


Figure 11. **Decrypted SSL Stream in the Wireshark SSL Stream Window.**

The window shows the entire conversation between the two communication parties and color codes the direction of the packet flow. Wireshark displays the encrypted TCP stream and tries to separate the packages, which works quite good, however for some reason it doubles the output.

Before diving into the details of the protocol messages, let’s first get an overview of the call sequence, as depicted in the following figure.

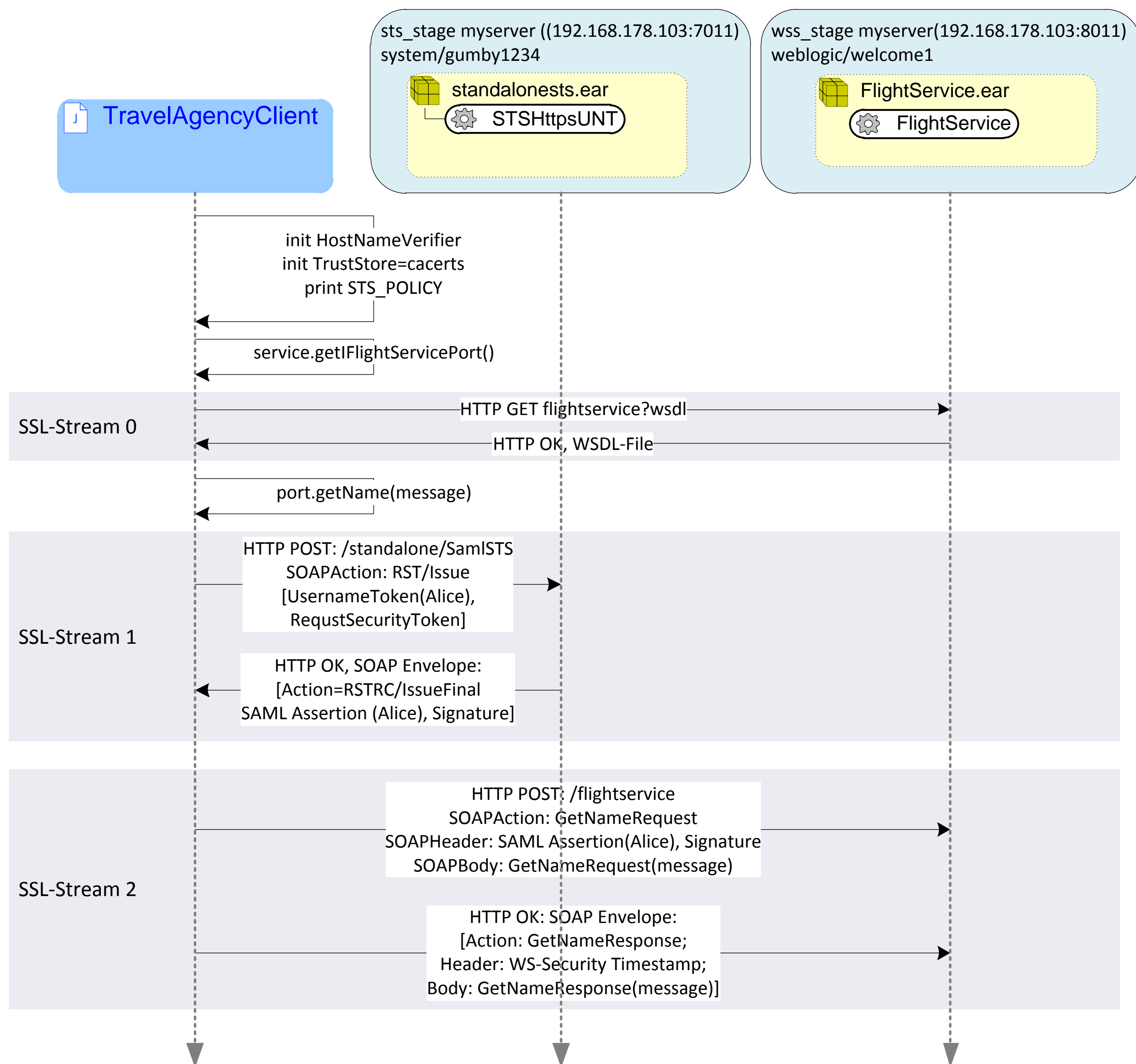


Figure 12. **Sequence Diagram of the flight service message flow.**

The Client starts with some initializations, including the initialization of a Hostname Verifier and the Trust Store which points to the cacerts key store. After that, it retrieves the security policy of the STS from its environment and prints the policy to the output stream, which is visible at the console. Then the client gets the Service Port in the call to service.getIFlightServicePort(), which results into the first SSL Stream, that is SSL-Stream 0. The client retrieves the WSDL of the flightservice in a HTTP Get request and receives it in the HTTP Response package.
After some initialization of the port instance, the client makes the call to the web service in the code line port.getName(message). Subsequently the next two SSL streams can be observed on the wire. The whole Song and Dance of the Authentication, SAML assertion and subsequent web service call is done behind the scenes by Weblogic classes, within this single call. In the SSL-Stream 1 the client sends a HTTP POST request to the STS server containing the WS-Security protocol message RST (Request Security Token), packaged in a SOAP request. It contains a UsernameToken of Alice together with Alice's Password and the RST. The STS authenticates the subject of the RST, which is Alice and issues a SAML assertion, containing an authentication statement. The assertion is signed and returned in a SOAP envelope.

Subsequently the SSL-Stream 2 starts. The client makes a SOAP request to the flightservice using the SOAP Action GetNameRequest. The SOAP Header contains the SAML assertion together with a signature. The SOAP Body contains the SOAP request itself. The WSS server, which is hosting the flightservice, verifies the validity of the SAML assertion and authenticates the user “Alice”. It forwards the SOAP request to the web service. In the HTTP response it puts a SOAP envelope with a WS-Security element in the header, containing just a timestamp. In the SOAP body it transfers the response to the flightservice request which ends the call sequence.
Now we are prepared to look at the details of this communication, presented in the following sections.

#### 4.5.1 SSL Stream 0

The first request is the client asking for the WSDL of the flightservice in an HTTP GET Request.

```
GET /flightservice/flightservice?wsdl HTTP/1.1
User-Agent: Java/1.6.0_29
Host: 192.168.178.103:8012
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
```

The WSS answers with a HTTP OK response which contains the WSDL as xml file.

```
HTTP/1.1 200 OK
Date: Sat, 07 Jul 2012 12:26:35 GMT
Transfer-Encoding: chunked
Content-Type: text/xml;charset=utf-8
X-Powered-By: Servlet/3.0 JSP/2.2
```

Here is the payload of this package. We use the eclipse xml editor to clean up the document and comment the output.

```
<?xml version='1.0' encoding='UTF-8'?><!-- Published by JAX-WS RI at http://jax-ws.dev.java.net.
          RI's version is JAX-WS RI 2.2.6hudson-86 svn-revision#12773. --><!-- Generated by JAX-WS RI at http://jax-ws.dev.java.net.
          RI's version is JAX-WS RI 2.2.6hudson-86 svn-revision#12773. -->
<definitions
          xmlns:wssutil="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
          xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
          xmlns:wsp="http://www.w3.org/ns/ws-policy" xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy"
          xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
          xmlns:tns="http://wsinterop.org/samples" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
          xmlns="http://schemas.xmlsoap.org/wsdl/" targetNamespace="http://wsinterop.org/samples"
          name="FlightService">
          <wsp:UsingPolicy wssutil:Required="true" />
          <wsp1_2:Policy wssutil:Id="Wssp1.2-2007-Saml1.1-Bearer-Https.xml">
                    <ns1:TransportBinding
                              xmlns:ns1="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">
                              <wsp1_2:Policy>
                                        <ns1:TransportToken>
                                                  <wsp1_2:Policy>
                                                            <ns1:HttpsToken />
                                                  </wsp1_2:Policy>
                                        </ns1:TransportToken>
                                        <ns1:AlgorithmSuite>
                                                  <wsp1_2:Policy>
                                                            <ns1:Basic256 />
                                                  </wsp1_2:Policy>
                                        </ns1:AlgorithmSuite>
                                        <ns1:Layout>
                                                  <wsp1_2:Policy>
                                                            <ns1:Lax />
                                                  </wsp1_2:Policy>
                                        </ns1:Layout>
                                        <ns1:IncludeTimestamp />
                              </wsp1_2:Policy>
                    </ns1:TransportBinding>
                    <ns2:SupportingTokens
                              xmlns:ns2="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">
                              <wsp1_2:Policy>
                                        <ns2:SamlToken
```

The WSDL File starts here.

This web service requires WS-Policies to be regarded. It gives a name for the policy with the ID element.

The Transport Binding specifies the requirements for the transport.

The Https token needs to be encrypted with a Basic256 algorithm.

A timestamp is required in the transport token.

Here starts the second security policy element

```
                                                    ns2:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-
securitypolicy/200702/IncludeToken/AlwaysToRecipient">
                                                    <wsp1_2:Policy>
                                                              <ns2:WssSamlV11Token11 />
                                                    </wsp1_2:Policy>
                                          </ns2:SamlToken>
                                </wsp1_2:Policy>
                      </ns2:SupportingTokens>
          </wsp1_2:Policy>
```

Here we see that a SAML 1.1 Token is required for authentication.

```
          <types>
                    <xsd:schema>
                              <xsd:import namespace="http://wsinterop.org/samples"
                                        schemaLocation="https://192.168.178.103:8012/flightservice/flightservice?xsd=1" />
                    </xsd:schema>
          </types>
```

This is the web services types specification.

```
          <message name="GetName">
                    <part name="parameters" element="tns:GetName" />
          </message>
          <message name="GetNameResponse">
                    <part name="parameters" element="tns:GetNameResponse" />
          </message>
```

This is the message specification, which specifies the messages GetName and GetNameResponse

```
          <portType name="IFlightService">
                    <operation name="GetName">
                              <input wsam:Action="http://wsinterop.org/samples/IFlightService/GetNameRequest"
                                        message="tns:GetName" />
                              <output wsam:Action="http://wsinterop.org/samples/IFlightService/GetNameResponse"
                                        message="tns:GetNameResponse" />
                    </operation>
          </portType>
```

In the port type it is specified that there is the operation GetName with it's input and output actions.

```
          <binding name="IFlightServicePortBinding" type="tns:IFlightService">
                    <wsp:PolicyReference URI="#Wssp1.2-2007-Saml1.1-Bearer-Https.xml" />
                    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
                              style="document" />
                    <operation name="GetName">
                              <soap:operation soapAction="" />
                              <input>
                                        <soap:body use="literal" />
                              </input>
                              <output>
                                        <soap:body use="literal" />
                              </output>
                    </operation>
          </binding>
```

In the binding we specify to use the policy named by the Id above and to use the SOAP protocol.

```
          <service name="FlightService">
                    <port name="IFlightServicePort" binding="tns:IFlightServicePortBinding">
                              <soap:address
                                        location="https://192.168.178.103:8012/flightservice/flightservice" />
                    </port>
          </service>
          <def />
</definitions>
```

The service element binds this web service to the actual URL of the wss-server.

This was SSL-Stream 0. Now the TravelAgencyClient interprets the WSDL and turns to the STS server to obtain a SAML assertion in SSL-Stream 1.

### 4.5.2 SSL Stream 1

This SSL Stream starts with a HTTP POST from the Client to the STS server.

```
POST /standalonests/SamlSTS HTTP/1.1
User-Agent: Oracle JAX-RPC 1.1
Content-Type: text/xml; charset=utf-8
SOAPAction: "http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/Issue"
Host: 192.168.178.103:7012
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
Content-Length: 2369
```

The client sends a HTTP POST request containing a SOAP document as payload. The SOAPAction attribute indicates that this is a WS-Trust Request Security Token (RST) protocol message. The xml file contained in this POST request contains the details.

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
          xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-trust/200512"
          xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
          xmlns:wsa="http://www.w3.org/2005/08/addressing" xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
          <env:Header xmlns:wsa="http://www.w3.org/2005/08/addressing">
                    <wsa:MessageID
xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility">uuid:11d1def534ea1be0:2ac1b022:1386167e68d:-7fff
                    </wsa:MessageID>
```

Soap addressing indicates an addressing schema, independent from the transport protocol.

```
                <wsa:Action xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility">http://docs.oasis-open.org/ws-sx/ws-
trust/200512/RST/Issue
                </wsa:Action>
                <wsa:To xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility">https://192.168.178.103:7012/standalonests/SamlSTS
                </wsa:To>
                <wsa:ReplyTo xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility">
                        <wsa:Address>http://www.w3.org/2005/08/addressing/anonymous
                        </wsa:Address>
                </wsa:ReplyTo>
                <wsse:Security
                        xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
                        env:mustUnderstand="1">
                        <wsse:UsernameToken wsu:Id="unt_47tW3YQ71iYGhBEu">
                                <wsse:Username>Alice</wsse:Username>
                                <wsse:Password
                                        Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-
profile-1.0#PasswordText">Interop1</wsse:Password>
                                <wsse:Nonce
                                        EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-
security-1.0#Base64Binary">7DY7+zAevmcQPgNC7/QibTVE0pAg7uwdyuqysMeBJbw=</wsse:Nonce>
                                <wsu:Created>2012-07-07T12:26:37Z</wsu:Created>
                        </wsse:UsernameToken>
                        <wsu:Timestamp>
                                <wsu:Created>2012-07-07T12:26:37Z</wsu:Created>
                                <wsu:Expires>2012-07-07T12:27:37Z</wsu:Expires>
                        </wsu:Timestamp>
                </wsse:Security>
        </env:Header>
        <env:Body>
                <wst:RequestSecurityToken>
                        <wst:TokenType>http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV1.1
                        </wst:TokenType>
                        <wsp:AppliesTo>
                                <wsa:EndpointReference>
                                        <wsa:Address>https://192.168.178.103:8012/flightservice/flightservice
                                        </wsa:Address>
                                </wsa:EndpointReference>
                        </wsp:AppliesTo>
                        <wst:Lifetime>
                                <wsu:Created>2012-07-07T12:26:37.302Z</wsu:Created>
                                <wsu:Expires>2012-07-07T12:27:37.302Z</wsu:Expires>
                        </wst:Lifetime>
                        <wst:KeyType>http://docs.oasis-open.org/ws-sx/ws-trust/200512/Bearer
                        </wst:KeyType>
                        <wst:RequestType>http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue
                        </wst:RequestType>
                </wst:RequestSecurityToken>
        </env:Body>
</env:Envelope>
```


The SoapAction tells that the client issues a RST protocol message.

This is the URL of the recipient of the RST.

The reply message will be sent via the HTTP response.

The UsernameToken states that Alice wants to be authenticated and provides her password.

A Nonce is a unique string needed to mitigate reply attacks.

As required by the flight service security policy the transport token contains a timestamp. It indicates a validity period of one minute.

The SOAP Body contains the actual RST.

This identifies the token as a SAML 1.1 requst.

The recipient for the SAML assertion is the flightservice

The validity period has transport token in this case.

The KeyType indicates that the bearer of the userid wants to authenticate.

The STS server replies with an HTTP OK and attaches a SOAP message containing the requested SAML assertion.

```
HTTP/1.1 200 OK
Date: Sat, 07 Jul 2012 12:26:37 GMT
Transfer-Encoding: chunked
Content-Type: text/xml; charset=utf-8
X-Powered-By: Servlet/3.0 JSP/2.2
```

Here is the SOAP message reply of the STS:


```
<?xml version='1.0' encoding='UTF-8'?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-trust/200512"
        xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
        xmlns:wsa="http://www.w3.org/2005/08/addressing" xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
        <env:Header xmlns:wsa="http://www.w3.org/2005/08/addressing">
                <wsa:Action xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility">http://docs.oasis-open.org/ws-sx/ws-
trust/200512/RSTRC/IssueFinal
                </wsa:Action>
                <wsa:MessageID>uuid:c149c057-6abf-467b-a354-ff6fb3666df4
                </wsa:MessageID>
                <wsa:RelatesTo>uuid:11d1def534ea1be0:2ac1b022:1386167e68d:-7fff
                </wsa:RelatesTo>
                <wsse:Security
                        xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
                        env:mustUnderstand="1">
                        <wsu:Timestamp>
                                <wsu:Created>2012-07-07T12:26:42Z</wsu:Created>
                                <wsu:Expires>2012-07-07T12:27:42Z</wsu:Expires>
                        </wsu:Timestamp>
                </wsse:Security>
        </env:Header>
        <env:Body>
                <wst:RequestSecurityTokenResponseCollection>
                        <wst:RequestSecurityTokenResponse>
                                <wst:TokenType>http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV1.1
                                </wst:TokenType>
                                <wsp:AppliesTo>
                                        <wsa:EndpointReference>
                                                <wsa:Address>https://192.168.178.103:8012/flightservice/flightservice
```


The SOAP action indicates that the body contains a Response Collection for previous the RST.

The Message ID of this SOAP document and the ID of the related SOAP request are given here.

This timestamp indicates the validity of this single soap message.

The Soap Body contains the Response to the previous RST, which is a collection in this case.

Tokentype, Recipient and Lifetime are the same as in the corresponding RST

```
                    </wsa:Address>
                </wsa:EndpointReference>
            </wsp:AppliesTo>
            <wst:Lifetime>
                <wsu:Created>2012-07-07T12:26:37.302Z</wsu:Created>
                <wsu:Expires>2012-07-07T12:27:37.302Z</wsu:Expires>
            </wst:Lifetime>
            <wst:KeySize>256</wst:KeySize>
            <wst:RequestType>http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue
            </wst:RequestType>
            <wst:RequestedSecurityToken>
                <Assertion xmlns="urn:oasis:names:tc:SAML:1.0:assertion"
                    xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
xmlns:samlp="urn:oasis:names:tc:SAML:1.0:protocol"
                    AssertionID="dfb16d9c335a88d675dfd36fc65c53c5" IssueInstant="2012-07-
07T12:26:41.711Z"
                    Issuer="www.oracle.com" MajorVersion="1" MinorVersion="1">
                    <Conditions NotBefore="2012-07-07T12:26:41.711Z"
                        NotOnOrAfter="2012-07-07T12:28:41.711Z" />
                    <AuthenticationStatement
                        AuthenticationInstant="2012-07-07T12:26:41.711Z"
```

The SAML Assertion starts here. The Issue and Authentication Instant carry the same timestamp, which is 4 seconds after the corresponding RST was created.

The Lifespan of the assertion is 60 seconds.

```
        AuthenticationMethod="urn:oasis:names:tc:SAML:1.0:am:unspecified">
                        <Subject>
                            <NameIdentifier
        Format="urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified"
        NameQualifier="www.oracle.com">Alice</NameIdentifier>
```

The subject of this authentication statement is “Alice”. It is further qualified by the NameQualifier.

```
                            <SubjectConfirmation>
        <ConfirmationMethod>urn:oasis:names:tc:SAML:1.0:cm:bearer
                                </ConfirmationMethod>
                            </SubjectConfirmation>
                        </Subject>
                    </AuthenticationStatement>
```

This states that Alice as the bearer of this user id was authenticated.

This element contains the signature of the assertion.

```
                    <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
                        <dsig:SignedInfo>
                            <dsig:CanonicalizationMethod
        Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
```

Algorithem with which this assertion was canonicalized.

```
                            <dsig:SignatureMethod
        Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
```

The assertion was signed using the rsa-sha1 algorithm.

```
                            <dsig:Reference
URI="#dfb16d9c335a88d675dfd36fc65c53c5">
```

This refers to the AssertionId and indicates that this signature applies only to the assertion element.

```
                                <dsig:Transforms>
                                    <dsig:Transform
        Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature" />
                                    <dsig:Transform
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
                                </dsig:Transforms>
                                <dsig:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
```

This indicates the algorithms that were applied to this assertion during creation of the signature.

```
        <dsig:DigestValue>OPbvF5evNblt1SwuPBQp19YMZIY=
                                </dsig:DigestValue>
                            </dsig:Reference>
                        </dsig:SignedInfo>
```

This is the digest of the assertion.


```
        <dsig:SignatureValue>wERnwVHhPW5qAE+kdU+pnYKDAIwsbiXP5wDYKHqjf5+SfYZB1Gnd+9KtF6N0SeYZqwfg5rpLatCmHTA/Wdn2u8P/UeRvXrqjrt9YstLYY
z4GuGZc1OXxgHEJicfiHQk5YJGS0ADlMoMAaPs2+UrBBteDNpVa2JvSk3agchn33ao=
                        </dsig:SignatureValue>
                        <dsig:KeyInfo>
                            <dsig:X509Data>
```


This is the digest encrypted with the private key of wssipsts

This is the Base-64 encoded certificate of wssipsts, which is the identity certificate configured for the STS


```
        <dsig:X509Certificate>MIIDDDCCAfSgAwIBAgIQb6U6bec4ZHW96T5N2A/NdTANBgkqhkiG9w0BAQUFADAwMQ4wDAYDVQQKDAVPQVNJUzEeMBwGA1UEAwwVT0FT
SVMgSW50ZXJvcCBUZXN0IENBMB4XDTA1MTAyNzAwMDAwMFoXDTE4MTAyNzIzNTk1OVowQjEOMAwGA1UECgwFT0FTSVMxIDAeBgNVBAsTF09BU0lTIEludGVyb3AgVGVzdCBDZXJ0M
Q4wDAYDVQQDDAVXc3NJUDCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEA2X9ZWiek/59vvg+l/lmzWjBYiqoOuSI+ms3ief7RyhPNh/IrGE3VwU67HsygNeavE06S6xNfcNWUNL
qEdRmd/29WnubNH7hWJsqp7rn8g/mxNVkynCkJ1saKuD8ILiKfNg0e8UUE9QzwEz1fxw81OR0SbDitfTrDj8Q/ouCgEaUCAwEAAaOBkzCBkDAJBgNVHRMEAjAAMDMGA1UdHwQsMCo
wKKImhiRodHRwOi8vaW50ZXJvcC5iYnRlc3QubmV0L2NybC9jYS5jcmwwDgYDVR0PAQH/BAQDAgSwMB0GA1UdDgQWBBQb1AYE+P8ue/8qbgUJOKoyDXFqaTAfBgNVHSMEGDAWgBTA
nSj8wes1oR3WqqqgHBpNwkkPDzANBgkqhkiG9w0BAQUFAAOCAQEAeltzyUHj+/0i3Hsj5XvWrJ7mF+zBFwp7E6CPLP/urfMdl1VFaBttOCcdWRrm8GI3KsGQMV6dpzAykl1JDO7T6
IMSMYA1/YTsSH9S8xoubL/7IGYj3izKZ9LrV7fJJOHOerKLgIk/0X8DzH15jwel271s6Nh6DiXqU2Hf0YUmauLAH+rbiuNLlUKM5UkP4BtGqPw+6tvyaUOa3fzJs92WB+j5x91/xm
vNg+ZTp+TEfyINM3wZAHwoIzXtEViopCRsXkmLr+IBGszmUpZnPd2QuqDSSkQhlZmUAuNVPCTBoNuWBX/tvvAw3a3jl+DXB+Fn2JbRpoUdvkgAWCAJ6hrKgA==
                                </dsig:X509Certificate>
                            </dsig:X509Data>
                        </dsig:KeyInfo>
                    </dsig:Signature>
                </Assertion>
            </wst:RequestedSecurityToken>
        </wst:RequestSecurityTokenResponse>
    </wst:RequestSecurityTokenResponseCollection>
        </env:Body>
</env:Envelope>
```


This ends the SSL-Stream 1. Subsequently the client uses the SAML assertion to make the request to the flight service. This communication is captured in SSL-Stream 2.

### 4.5.3 SSL Stream 2

This stream starts with a HTTP POST request to the flight service with the SOAP action GetNameRequest. This is the actual web service request.

```
POST /flightservice/flightservice HTTP/1.1
Accept: text/xml, multipart/related
Content-Type: text/xml; charset=utf-8
SOAPAction: "http://wsinterop.org/samples/IFlightService/GetNameRequest"
User-Agent: JAX-WS RI 2.2.6hudson-86  svn-revision#12773
Host: 192.168.178.103:8012
Connection: keep-alive
Content-Length: 4152
```

The SOAP message of this Request is given below.

```
<?xml version='1.0' encoding='UTF-8'?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Header>
        <To xmlns="http://www.w3.org/2005/08/addressing">https://192.168.178.103:8012/flightservice/flightservice
        </To><
        <Action xmlns="http://www.w3.org/2005/08/addressing">http://wsinterop.org/samples/IFlightService/GetNameRequest
        </Action>
        <ReplyTo xmlns="http://www.w3.org/2005/08/addressing">
            <Address>http://www.w3.org/2005/08/addressing/anonymous</Address>
        </ReplyTo>
        <FaultTo xmlns="http://www.w3.org/2005/08/addressing">
            <Address>http://www.w3.org/2005/08/addressing/anonymous</Address>
        </FaultTo>
        <MessageID xmlns="http://www.w3.org/2005/08/addressing">uuid:1346b582-b1a8-48b9-bed6-0d1aeb8913f8
        </MessageID>
        <wsse:Security
            xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
            S:mustUnderstand="1">
            <Assertion xmlns="urn:oasis:names:tc:SAML:1.0:assertion"
                xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
xmlns:samlp="urn:oasis:names:tc:SAML:1.0:protocol"
                AssertionID="dfb16d9c335a88d675dfd36fc65c53c5" IssueInstant="2012-07-07T12:26:41.711Z"
                Issuer="www.oracle.com" MajorVersion="1" MinorVersion="1">
                <Conditions NotBefore="2012-07-07T12:26:41.711Z"
                    NotOnOrAfter="2012-07-07T12:28:41.711Z" />
                <AuthenticationStatement
                    AuthenticationInstant="2012-07-07T12:26:41.711Z"
                    AuthenticationMethod="urn:oasis:names:tc:SAML:1.0:am:unspecified">
                    <Subject>
                        <NameIdentifier
                            Format="urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified"
                            NameQualifier="www.oracle.com">Alice</NameIdentifier>
                        <SubjectConfirmation>
                            <ConfirmationMethod>urn:oasis:names:tc:SAML:1.0:cm:bearer
                            </ConfirmationMethod>
                        </SubjectConfirmation>
                    </Subject>
                </AuthenticationStatement>
                <dsig:Signature xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
                    <dsig:SignedInfo>
                        <dsig:CanonicalizationMethod
                            Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
                        <dsig:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-
sha1" />

                        <dsig:Reference URI="#dfb16d9c335a88d675dfd36fc65c53c5">
                            <dsig:Transforms>
                                <dsig:Transform


    Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature" />
                                <dsig:Transform
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
                            </dsig:Transforms>
                            <dsig:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />

    <dsig:DigestValue>OPbvF5evNblt1SwuPBQp19YMZIY=</dsig:DigestValue>
                        </dsig:Reference>
                    </dsig:SignedInfo>

    <dsig:SignatureValue>wERnwVHhPW5qAE+kdU+pnYKDAIwsbiXP5wDYKHqjf5+SfYZB1Gnd+9KtF6N0SeYZqwfg5rpLatCmHTA/Wdn2u8P/UeRvXrqjrt9YstLYY
z4GuGZc1OXxgHEJicfiHQk5YJGS0ADlMoMAaPs2+UrBBteDNpVa2JvSk3agchn33ao=
                    </dsig:SignatureValue>
                    <dsig:KeyInfo>
                        <dsig:X509Data>

    <dsig:X509Certificate>MIIDDDCCAfSgAwIBAgIQb6U6bec4ZHW96T5N2A/NdTANBgkqhkiG9w0BAQUFADAwMQ4wDAYDVQQKDAVPQVNJUzEeMBwGA1UEAwwVT0FT
SVMgSW50ZXJvcCBUZXN0IENBMB4XDTA1MTAyNzAwMDAwMFoXDTE4MTAyNzIzNTk1OVowQjEOMAwGA1UECgwFT0FTSVMxIDAeBgNVBAsTF09BU0lTIEludGVyb3AgVGVzdCBDZXJ0M
Q4wDAYDVQQDDAVXc3NJUDCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEA2X9ZWiek/59vvg+l/lmzWjBYiqoOuSI+ms3ief7RyhPNh/IrGE3VwU67HsygNeavE06S6xNfcNWUNL
qEdRmd/29WnubNH7hWJsqp7rn8g/mxNVkynCkJ1saKuD8ILiKfNg0e8UUE9QzwEz1fxw81OR0SbDitfTrDj8Q/ouCgEaUCAwEAAaOBkzCBkDAJBgNVHRMEAjAAMDMGA1UdHwQsMCo
wKKImhiRodHRwOi8vaW50ZXJvcC5iYnRlc3QubmV0L2NybC9jYS5jcmwwDgYDVR0PAQH/BAQDAgSwMB0GA1UdDgQWBBQb1AYE+P8ue/8qbgUJOKoyDXFqaTAfBgNVHSMEGDAWgBTA
nSj8wes1oR3WqqqgHBpNwkkPDzANBgkqhkiG9w0BAQUFAAOCAQEAeltzyUHj+/0i3Hsj5XvWrJ7mF+zBFwp7E6CPLP/urfMdl1VFaBttOCcdWRrm8GI3KsGQMV6dpzAykl1JDO7T6
IMSMYA1/YTsSH9S8xoubL/7IGYj3izKZ9LrV7fJJOHOerKLgIk/0X8DzH15jwel271s6Nh6DiXqU2Hf0YUmauLAH+rbiuNLlUKM5UkP4BtGqPw+6tvyaUOa3fzJs92WB+j5x91/xm
vNg+ZTp+TEfyINM3wZAHwoIzXtEViopCRsXkmLr+IBGszmUpZnPd2QuqDSSkQhlZmUAuNVPCTBoNuWBX/tvvAw3a3jl+DXB+Fn2JbRpoUdvkgAWCAJ6hrKgA==
                            </dsig:X509Certificate>
                        </dsig:X509Data>
                    </dsig:KeyInfo>
```

This is the end point URL of the recipient of this SOAP message.

This is the SOAP Action to be processed, i.e. the call to the web service.

Reply and Error Address are anonymous.

The recipient must understand this ws-security element.

This assertion is identical to the one of SSL-Stream 1 and the same notes apply here. However now it is contained in the SOAP Header and not in the Body.

```
                    </dsig:Signature>
               </Assertion>
               <wsu:Timestamp
                    xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
1.0.xsd">
                    <wsu:Created>2012-07-07T12:26:36Z</wsu:Created>
                    <wsu:Expires>2012-07-07T12:27:36Z</wsu:Expires>
               </wsu:Timestamp>
          </wsse:Security>
     </S:Header>
     <S:Body>
          <ns0:GetName xmlns:ns0="http://wsinterop.org/samples">
               <GetNameRequest>Travel Agency client to Flight Service</GetNameRequest>
          </ns0:GetName>
     </S:Body>
</S:Envelope>
```

The SOAP Body contains the web service request.

The STS now responds with a HTTP OK reply and attaches the SOAP Response.

```
HTTP/1.1 200 OK
Date: Sat, 07 Jul 2012 12:26:42 GMT
Transfer-Encoding: chunked
Content-Type: text/xml; charset=utf-8
X-Powered-By: Servlet/3.0 JSP/2.2
```

The content of the SOAP Reply is contained in the following xml document.

```
<?xml version='1.0' encoding='UTF-8'?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
     <S:Header>
          <To xmlns="http://www.w3.org/2005/08/addressing">http://www.w3.org/2005/08/addressing/anonymous</To>
          <Action xmlns="http://www.w3.org/2005/08/addressing">http://wsinterop.org/samples/IFlightService/GetNameResponse
          </Action>
          <MessageID xmlns="http://www.w3.org/2005/08/addressing">uuid:f8670c8c-a04e-4f27-bf70-4d263b80cc2c
          </MessageID>
          <RelatesTo xmlns="http://www.w3.org/2005/08/addressing">uuid:1346b582-b1a8-48b9-bed6-0d1aeb8913f8
          </RelatesTo>
          <wsse:Security
               xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
               S:mustUnderstand="1">
               <wsu:Timestamp
                    xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
1.0.xsd">
                    <wsu:Created>2012-07-07T12:26:46Z</wsu:Created>
                    <wsu:Expires>2012-07-07T12:27:46Z</wsu:Expires>
               </wsu:Timestamp>
          </wsse:Security>
     </S:Header>
     <S:Body>
          <ns0:GetNameResponse xmlns:ns0="http://wsinterop.org/samples">
               <GetNameResponse>Request from [Travel Agency client to Flight
                    Service], response message from Oracle FlightService HTTPS)
               </GetNameResponse>
          </ns0:GetNameResponse>
     </S:Body>
</S:Envelope>
```

The message ID relates to the previous SOAP request by the client in tSSL-Stream2.

The ws-security element only contains the lifespan for this response.

The SOAP Body contains the web service response text.

This ends the successful web service call covering three SSL streams.

# 5 Conclusion

We investigated a concrete case of a WS-Trust scenario, using an example from the Weblogic Server distribution. Starting from the original setup we isolated this example into a separate Eclipse project, which uses dedicated WLS servers. This modified example project was tested on different platforms and served as a test environment for this analysis. It can also be used as a starting point for proof of concepts and prototypes since it is easy and fast to set up and provides all necessary scripts to configure WLS domains with SSL and SAML.
Using this modified eclipse project we demonstrated a system configuration to analyze the SSL traffic between the participants. We thoroughly investigated the SSL streams, including a presentation and explanation of the captured SOAP messages. Through this analysis we deepened our understanding of WS-Trust, the implementation of WS-Trust based scenarios and the flow of protocol messages. The experience gathered in this workshop can help with configurations and problem resolutions in real world SOA architectures.

# 6 Links

This document as PDF:
http://dl.dropbox.com/u/16989587/weblogic-corner/WS-Security_unveiled/WS-Security_unveiled.pdf

Text file containing the output from building and running the project:
http://dl.dropbox.com/u/16989587/weblogic-corner/WS-Security_unveiled/saml_bearer11ssl.output.txt

WLS example documentation as PDF:
http://dl.dropbox.com/u/16989587/weblogic-corner/WS-Security_unveiled/WLS_Example_SAML_Bearer11ssl.pdf

Eclipse project file:
http://dl.dropbox.com/u/16989587/weblogic-corner/WS-Security_unveiled/saml_bearer11ssl.zip