

## About ...

Harvinder Saluja is the Chief Java Architect and Founder of MindTelligent, Inc. He has over 16 years of industry experience and specializes in Java technology, the Java 2 Platform, Enterprise Edition (J2EE), Object-Oriented software design and development, and software patterns.

He currently consults with MindTelligent clients to help architect, design, develop, and deploy Java technology-based enterprise and Web services solutions and speaks regularly at industry conferences. His software architecture and design engagements include several State Agencies and mid to large sized businesses.

Harvinder has authored three courseware books:

- ✓ J2EE Development with J Developer 10g
- ✓ Build J2EE Applications using Oracle 10g AS
- ✓ Object Oriented Analysis and Design With Sparx Enterprise Architect

Harvinder is also involved in conducting J2EE workshops for the Federal and the State agencies. To view the client testimonials for the workshops conducted by Harvinder Saluja, please visit:

<http://www.mindtelligent.com/pdf/J2EEOracleEvals.pdf>

He has contributed to development of plug-ins and extensions for JDeveloper10g, Eclipse and Rational Application Developer for WebSphere Software. These extensions and plug-ins can be downloaded from.

<http://technet.oracle.com/products/jdev/htdocs/partners/addins/exchange/job/content.html>

Harvinder was awarded “JDeveloper Extensions Developer” of the year by Oracle for the year 2003 for plug ins to generate Java Design Pattern code.

<http://www.oracle.com/technology/oramag/oracle/03-nov/o63editorschoice.html#SALUJA>

Table of Contents

- 1. Objective.....3
- 2. Overview of SSL Keys and Certificates .....4
- 3. Using Keys and Certificates with OC4J and Oracle HTTP Server .....7
- 4. Enabling SSL in OC4J ..... 11
  - 4.1. Configuring Oracle HTTP Server for SSL..... 11
  - 4.2. Creating an SSL Certificate and Configuring HTTPS ..... 11

## 1. Objective

OC4J supports Secure Socket Layer (SSL) communication between Oracle HTTP Server and OC4J in an Oracle Application Server environment, using secure AJP. This is the secure version of Apache JServ Protocol, the protocol that Oracle HTTP Server uses to communicate with OC4J.

This tutorial focuses on:

- ✓ Overview of SSL Keys and Certificates
- ✓ Using Keys and Certificates with OC4J and Oracle HTTP Server
- ✓ Enabling SSL in OC4J
- ✓ Requesting Client Authentication
- ✓ Resolving Common SSL Problems

Note:

Secure communication between a client and Oracle HTTP Server is independent of secure communication between Oracle HTTP Server and OC4J. This chapter tutorial only secure communication between Oracle HTTP Server and OC4J.

## 2. Overview of SSL Keys and Certificates

- ✓ In SSL communication between two entities, such as companies or individuals, the server has a public key and an associated private key. Each key is a number, with the private key of an entity being kept secret by that entity, and the public key of an entity being publicized to any other parties with which secure communication might be necessary.
  
- ✓ The security of the data exchanged is guaranteed by keeping the private key secret, and by the complex encryption algorithm. This system is known as asymmetric encryption, because the key used to encrypt data is not the same as the key used to decrypt data.
  
- ✓ Asymmetric encryption has a performance cost due to its complexity. A much faster system is symmetric encryption, where the same key is used to encrypt and decrypt data. But the weakness of symmetric encryption is that the same key has to be known by both parties, and if anyone intercepts the exchange of the key, then the communication becomes insecure.
  
- ✓ SSL uses both asymmetric and symmetric encryption to communicate. An asymmetric key (PKI public key) is used to encode a symmetric encryption key (the bulk encryption key); the bulk encryption key is then used to encrypt subsequent communication. After both sides agree on the bulk encryption key, faster communication is possible without losing security and reliability.
  
- ✓ When an SSL session is negotiated, the following steps take place:
  1. The server sends the client its public key.
  2. The client creates a bulk encryption key, often a 128 bit RC4 key, using a specified encryption suite.
  3. The client encrypts the bulk key with the server's public key, and sends the encrypted bulk key to the server.

4. The server decrypts the bulk encryption key using the server's private key.
  
5. This set of operations is called key exchange. After key exchange has taken place, the client and the server use the bulk encryption key to encrypt all exchanged data.

Note:

It is possible, but rare, for the client to have its own private and public keys as well.

- ✓ In SSL the public key of the server is sent to the client in a data structure known as an X.509 certificate. This certificate, created by a certificate authority (CA), contains, a public key, information concerning the owner of the certificate, and optionally some digital rights of the owner. Certificates are digitally signed by the CA, which created them using that CA's digital certificate public key.
  
- ✓ In SSL, the CA's signature is checked by the receiving process to ensure that it is on the approved list of CA signatures. This check is sometimes performed by analysis of certificate chains. This occurs if the receiving process does not have the signing CA's public key on the approved list. In that case the receiving process checks to see if the signer of the CA's certificate is on the approved list or the signer of the signer, and so on. This chain of certificate, signer of certificate, signer of signer of certificate, and so on is a certificate chain. The highest certificate in the chain (the original signer) is called the root certificate of the certificate chain.
  
- ✓ The root certificate is often on the approved list of the receiving process. Certificates in the approve list are called trust points or trusted certificates. A root certificate can be signed by a CA or can be self-signed, meaning that the digital signature that verifies the root certificate is encrypted through the private key that corresponds with the public key that the certificate contains, rather than through the private key of a higher CA. (Note that certificates of the CAs themselves are always self-signed.)

- ✓ A keystore is used to store certificates, including the certificates of all trusted parties, for use by a program. Through its keystore, an entity such as OC4J (for example) can authenticate other parties as well as authenticate itself to other parties. The keystore password is obfuscated. Oracle HTTP Server has what is called a wallet for the same purpose. Sun's SSL implementation introduces the notion of a truststore, which is a keystore file that includes the trusted certificate authorities that a client will implicitly accept during an SSL handshake.
  
- ✓ In Java, a keystore is a `java.security.KeyStore` instance that you can create and manipulate using the `keytool` utility that is provided with the Sun Microsystems JDK. The underlying physical manifestation of this object is a file. For information about `keytool`, please visit <http://java.sun.com/j2se/1.3/docs/tooldocs/win32/keytool.html>

### 3. Using Keys and Certificates with OC4J and Oracle HTTP Server

The following steps describe using keys and certificates for SSL communication in OC4J. These are server-level steps, typically executed prior to deployment of an application that will require secure communication, perhaps when you first set up an Oracle Application Server instance.

1. Use `keytool` to generate a private key, public key, and unsigned certificate. You can place this information into either a new keystore or an existing keystore.
2. Obtain a signature for the certificate, using either of the following two approaches.

Generate your own signature:

- a. Use `keytool` to "self-sign" the certificate. This is appropriate if your clients trust you as, in effect, your own certificate authority.

Alternatively, obtain a signature from a recognized certificate authority:

- b. Using the certificate from Step 1, use `keytool` to generate a certificate request, which is a request to have the certificate signed by a certificate authority.
- c. Submit the certificate request to a certificate authority.
- d. Receive the signature from the certificate authority, and import it into the keystore, again using `keytool`. In the keystore, the signature is matched with the associated certificate.

Note:

Oracle Application Server includes Oracle Application Server Certificate Authority (OCA). OCA enables customers to create and issue certificates for themselves and their users, although these certificates would probably be unrecognized outside a customer's organization without prior arrangements.

In addition to steps 1 and 2 above, execute the following steps as necessary:

1. If the OC4J certificate is signed by an entity that Oracle HTTP Server does not yet trust, obtain the certificate of the entity and import it into Oracle HTTP Server. The specifics depend on whether the OC4J certificate in question is self-signed, as follows.

If OC4J has a self-signed certificate (essentially, Oracle HTTP Server does not yet trust OC4J):

- a. From OC4J, use `keytool` to export the OC4J certificate. This step places the certificate into a file that is accessible to Oracle HTTP Server.
- b. From Oracle HTTP Server, use Oracle Wallet Manager to import the OC4J certificate.

Alternatively, if OC4J has a certificate that is signed by another entity (that Oracle HTTP Server does not yet trust):

- c. Obtain the certificate of the entity in any appropriate way, such as by exporting it from the entity. The exact steps vary widely, depending on the entity.
- d. From Oracle HTTP Server, use Oracle Wallet Manager to import the certificate of the entity.

2. If the Oracle HTTP Server certificate is signed by an entity that OC4J does not yet trust, and OC4J is in a mode of operation that requires client authentication:
  - a. Obtain the certificate of the entity in any appropriate way, such as by exporting it from the entity. The exact steps vary widely, depending on the entity.
  - b. From OC4J, use `keytool` to import the certificate of the entity.

**Note:**

During communications over SSL between Oracle HTTP Server and OC4J, all data on the communications channel between the two is encrypted. The following steps are executed:

- ✓ The OC4J certificate chain is authenticated to Oracle HTTP Server during establishment of the encrypted channel.
- ✓ Optionally, if OC4J is in client-authentication mode, Oracle HTTP Server is authenticated to OC4J. This process also occurs during establishment of the encrypted channel.
- ✓ Any further communication after this initial exchange will be encrypted.

**Example: Creating an SSL Certificate and Generating Your Own Signature**

- ✓ This example corresponds to the step of obtaining a signature for the certificate, in the mode where you generate your own signature by using keytool to self-sign the certificate.
  
- ✓ First, create a keystore with an RSA private/public keypair, using the keytool command. The following example (in which % is the system prompt) uses the RSA keypair algorithm to generate a keystore to reside in a file named mykeystore, which has a password of 123456 and is valid for 21 days:

```
% keytool -genkey -keyalg "RSA" -keystore mykeystore -  
storepass 123456 -validity 21
```

**Note the following:**

- The keystore option specifies the name of the file in which the keys are stored.
- The storepass option sets the password for protecting the keystore.
- The validity option sets the number of days for which the certificate is valid.

The keytool prompts you for more information, as follows:

```
What is your first and last name?  
[Unknown]: Test User  
What is the name of your organizational unit?  
[Unknown]: Support  
What is the name of your organization?  
[Unknown]: Oracle  
What is the name of your City or Locality?  
[Unknown]: Redwood Shores  
What is the name of your State or Province?  
[Unknown]: CA  
What is the two-letter country code for this unit?  
[Unknown]: US  
Is <CN=Test User, OU=Support, O=Oracle, L=Reading,  
ST=Berkshire, C=GB> correct?  
[no]: yes  
  
Enter key password for <mykey>  
(RETURN if same as keystore password):
```

- ✓ The `mykeystore` file is created in the current directory. The default alias of the key is `mykey`.

## 4. Enabling SSL in OC4J

For secure communication between Oracle HTTP Server and OC4J, configuration steps are required at each end, as discussed in the following section.

### 4.1. Configuring Oracle HTTP Server for SSL

In Oracle HTTP Server, verify proper SSL settings in the `mod_oc4j.conf` file for secure communication. SSL must be enabled, with a wallet file and password specified, as follows:

- ✓ `Oc4jEnableSSL on`
- ✓ `Oc4jSSLWalletFile wallet_path`
- ✓ `Oc4jSSLWalletPassword pwd`

The `wallet_path` value is a directory path to the wallet file, without a file name. (The wallet file name is already known.) The `pwd` value is the wallet password.

### 4.2. Creating an SSL Certificate and Configuring HTTPS

The following example uses `keytool` to create a test certificate and shows all of the XML configuration necessary for HTTPS to work. To create a valid certificate for use in production environments, see the `keytool` documentation.

- ✓ Install the correct JDK

Ensure that JDK 1.3.x is installed. This is required for SSL with OC4J. Set the `JAVA_HOME` to the JDK 1.3 directory. Ensure that the JDK 1.3.x `JAVA_HOME/bin` is at the beginning of your path. This may be achieved by doing the following:

```
UNIX

$ PATH=/usr/opt/java130/bin:$PATH
$ export $PATH
$ java -version
java version "1.3.0"
```

```
Windows
```

```
set PATH=d:\jdk131\bin;%PATH%
```

- ✓ Ensure that this JDK version is set as the current version in your Windows registry. In the Windows Registry Editor under HKEY\_LOCAL\_MACHINE/SOFTWARE/JavaSoft/Java Development Kit, set 'CurrentVersion' to 1.3 (or later).

## 1. Request a certificate

- a. Change directory to ORACLE\_HOME/j2ee
- b. Create a keystore with an RSA private/public keypair using the `keytool` command. In our example, we generate a keystore to reside in a file named 'mykeystore', which has a password of '123456' and is valid for 21 days, using the 'RSA' key pair generation algorithm with the following syntax:

```
keytool -genkey -keyalg "RSA" -keystore  
mykeystore -storepass 123456 -validity 21
```

In this tool,

- ✓ the `keystore` option sets the filename where the keys are stored
- ✓ the `storepass` option sets the password for protecting the keystore
- ✓ the `validity` option sets number of days the certificate is valid

The `keytool` prompts you for more information, as follows:

```
keytool -genkey -keyalg "RSA" -keystore mykeystore -  
storepass 123456 -validity 21
```

```
What is your first and last name?
```

```
[Unknown]: Test User
```

```
What is the name of your organizational unit?
```

```
[Unknown]: Support
What is the name of your organization?
[Unknown]: Oracle
What is the name of your City or Locality?
[Unknown]: Redwood Shores
What is the name of your State or Province?
[Unknown]: CA
What is the two-letter country code for this unit?
[Unknown]: US
Is <CN=Test User, OU=Support, O=Oracle, L=Reading,
ST=Berkshire, C=GB> correct?
[no]: yes

Enter key password for <mykey>
(RETURN if same as keystore password):
```

**Note:**

To determine your 'two-letter country code', use the ISO country code list at <http://www.bcpl.net/~jspath/isocodes.html>.

- The mykeystore file is created in the current directory. The default alias of the key is mykey.
2. If you do not have a `secure-web-site.xml` file, then copy the `default-web-site.xml` to `ORACLE_HOME/j2ee/home/config/secure-web-site.xml`.
  3. Edit `secure-web-site.xml` with the following elements:
    - . Add `secure="true"` to the `<web-site>` element, as follows:
      - a. `<web-site port="8888" display-name="Default OracleAS Containers for J2EE Web Site" secure="true">`

- b. Add the following new line inside the `<web-site>` element to define the keystore and the password.
- c. 

```
<ssl-config keystore="<Your-Keystore>"
  keystore-password="<Your-Password>" />
```

Where `<Your-Keystore>` is the full path to the keystore and `<Your-Password>` is the keystore password. In our example, this is as follows:

```
<!-- Enable SSL -->
<ssl-config keystore="../../keystore" keystore-
password="123456"/>
```

**Note:**

The keystore path is relative to where the XML file resides.

- d. Change the web-site port number, to use an available port. For example, the default for SSL ports is 443, so change the Web site port attribute to `port="4443"`. To use the default of 443, you have to be a super user.
  - e. Now save the changes to `secure-web-site.xml`.
4. If you did not have the `secure-web-site.xml` file, then edit `server.xml` to point to the `secure-web-site.xml` file.
    - . Uncomment or add the following line in the file `server.xml` so that the `secure-web-site.xml` file is read.
      - a. 

```
<web-site path="./secure-web-site.xml" />
```

**Note:**

Even on Windows, you use a forward slash, not a backslash, in the XML files.

- b. Save the changes to `server.xml`.
- 
5. Stop and restart OC4J to initialize the `secure-web-site.xml` file additions. Test the SSL port by accessing the site in a browser on the SSL port. If successful, you will be asked to accept the certificate, because it is not signed by an accepted authority.

When completed, OC4J listens for SSL requests on one port and non-SSL requests on another. You can disable either SSL requests or non-SSL requests, by commenting out the appropriate `*web-site.xml` in the `server.xml` configuration file.

```
<web-site path="./secure-web-site.xml" /> - comment out  
this to remove SSL  
  
<default-site path="./default-web-site.xml" /> - comment  
out this to remove non-SSL
```