

CAPLIN

# KeyMaster 5.0

---

## Administration Guide

May 2012

CONFIDENTIAL

# Contents

<b>1</b>	<b>Preface.....</b>	<b>1</b>
1.1	What this document contains.....	1
	About Caplin document formats .....	2
1.2	Who should read this document.....	2
1.3	Related documents.....	2
1.4	Typographical conventions .....	3
1.5	Feedback.....	4
1.6	Acknowledgments.....	4
1.7	Open Source Software.....	4
<b>2</b>	<b>Overview.....</b>	<b>5</b>
<b>3</b>	<b>Technical assumptions and restrictions.....</b>	<b>6</b>
<b>4</b>	<b>Installing KeyMaster.....</b>	<b>8</b>
4.1	Important note on security of installation .....	8
4.2	Prerequisites.....	8
4.3	Installing on Linux or Sun Solaris.....	8
4.4	Installing on a Windows platform.....	9
4.5	Installed Files.....	10
4.6	Generating the required keys.....	10
<b>5</b>	<b>Deploying KeyMaster.....</b>	<b>13</b>
5.1	Deployment on a Tomcat server.....	13
5.2	Deployment on a JBoss server.....	14
5.3	Deployment on a BEA WebLogic server.....	15
	Deployment on WebLogic 9.1 .....	15
	Deployment on WebLogic 8.1 .....	17
5.4	Modifying the web.xml configuration file .....	17
	Changing KeyMaster's URL .....	20
5.5	Testing KeyMaster with the application server.....	23
	Testing the XHRKeymaster servlet .....	24
<b>6</b>	<b>Setting up Liberator to work with KeyMaster.....</b>	<b>25</b>
6.1	Making the public key file available to Liberator.....	25
6.2	Modifying the Liberator configuration file.....	26
6.3	Modifying the users.xml authorization file for XMLauth .....	29
6.4	Modifying the cfgauth.conf authorization file.....	30

6.5	Configuring a Liberator that uses javaauth authentication.....	31
<b>7</b>	<b>Testing Java-based KeyMaster with Liberator .....</b>	<b>32</b>
7.1	Configuring the test files .....	32
7.2	Launching the test page.....	34
<b>8</b>	<b>Testing KeyMaster.NET with Liberator.....</b>	<b>37</b>
<b>9</b>	<b>Making KeyMaster production ready.....</b>	<b>38</b>
9.1	Configuring the SHA256 signature algorithm.....	39
<b>10</b>	<b>Integrating KeyMaster with a hardware Key Store.....</b>	<b>41</b>
10.1	Key Store prerequisites and assumptions .....	41
10.2	Generating keys using OpenSSL.....	42
	Generating a private key .....	43
	Generating the public key .....	43
	Generating the certificate request .....	43
	Obtaining a signed certificate .....	43
	Converting the private key to DER format .....	44
10.3	Importing the private key file and certificate into the Key Store.....	44
10.4	Verifying the key import operation.....	46
10.5	Installing the required libraries.....	47
10.6	Modifying the web.xml file for Key Store access.....	47
10.7	Testing KeyMaster works with the Key Store.....	52
10.8	Configuring Liberator to use a new public key.....	52
10.9	Testing Liberator works with the new public key.....	53
10.10	Tidying up.....	53
<b>11</b>	<b>Customizing KeyMaster.....</b>	<b>54</b>
<b>12</b>	<b>Troubleshooting.....</b>	<b>55</b>
12.1	Synchronizing the servers.....	55
12.2	Liberator log file messages.....	55
<b>13</b>	<b>More about configuring Keymaster.....</b>	<b>58</b>
13.1	Configuration in web.xml.....	58
13.2	Adding mapping data to the KeyMaster credentials token.....	58
13.3	Adding the user name to the user credentials token.....	59
13.4	Protocol and domain compatibility .....	60
<b>14</b>	<b>Configuration reference.....</b>	<b>61</b>
14.1	keygen.props configuration reference.....	61

14.2	keyimporter.props configuration reference .....	62
14.3	web.xml configuration reference.....	65
	<description> .....	67
	<display-name> .....	67
	<init-param> .....	68
	<param-name> .....	68
	<param-value> .....	69
	<servlet> .....	69
	<servlet-class> .....	70
	<servlet-mapping> .....	70
	<servlet-name> .....	71
	<url-pattern> .....	71
	<web-app> .....	72
	web.xml parameters .....	73
15	Glossary of Terms and Acronyms.....	89

# 1 Preface

## 1.1 What this document contains

This document describes how to configure and operate the Caplin KeyMaster product, to provide a secure and reliable user authentication service.

**Note:** This document applies to **Standard Java-based KeyMaster release 5.0.x** and above, and **KeyMaster.Net release 5.0.x** and above.

- If you are deploying the *Standard Java-based KeyMaster*, read the following sections:
  - [Installing KeyMaster](#) <sup>8</sup>
  - [Deploying KeyMaster](#) <sup>13</sup>
  - [Setting up Liberator to work with KeyMaster](#) <sup>25</sup>
  - [Testing Java-based KeyMaster with Liberator](#) <sup>32</sup>
  - [Making KeyMaster production ready](#) <sup>38</sup>
- If you are deploying Java-based KeyMaster and intend to store private keys in a dedicated secure hardware module (a “**Key Store**”), follow the instructions in the section [Important note on security of installation](#) <sup>8</sup>.

You will also find the following sections useful:

- [Customizing KeyMaster](#) <sup>54</sup>
- [Troubleshooting](#) <sup>55</sup>
- [More about configuring KeyMaster](#) <sup>58</sup>
- [Configuration reference](#) <sup>61</sup>
- If you are deploying a KeyMaster Signature Generator that you have implemented using *KeyMaster.NET*, read the following sections:
  - [Setting up Liberator to work with KeyMaster](#) <sup>25</sup>
  - [Testing KeyMaster.NET with Liberator](#) <sup>37</sup>
  - [Making KeyMaster production ready](#) <sup>38</sup>
  - [Troubleshooting](#) <sup>55</sup> (some sections).
  - [Protocol and domain compatibility](#) <sup>60</sup>

## About Caplin document formats

This document is supplied in three formats:

- ◆ Portable document format (*.PDF* file), which you can read on-line using a suitable PDF reader such as Adobe Reader®. This version of the document is formatted as a printable manual; you can print it from the PDF reader.
- ◆ Web pages (*.HTML* files), which you can read on-line using a web browser. To read the web version of the document navigate to the *HTMLDoc\_m\_n* folder and open the file *index.html*.
- ◆ Microsoft HTML Help (*.CHM* file), which is an HTML format contained in a single file. To read a *.CHM* file just open it – no web browser is needed.

### For the best reading experience

On the machine where your browser or PDF reader runs, install the following Microsoft Windows® fonts: Arial, Courier New, Times New Roman, Tahoma. You must have a suitable Microsoft license to use these fonts.

### Restrictions on viewing *.CHM* files

You can only read *.CHM* files from Microsoft Windows.

Microsoft Windows security restrictions may prevent you from viewing the content of *.CHM* files that are located on network drives. To fix this either copy the file to a local hard drive on your PC (for example the Desktop), or ask your System Administrator to grant access to the file across the network. For more information see the Microsoft knowledge base article at <http://support.microsoft.com/kb/896054/>.

## 1.2 Who should read this document

This document is intended for System Administrators and Software Developers who need to deploy Caplin Liberator within an existing single sign-on system or authentication service. It is assumed that the reader has an understanding of network systems, running Java programs from the command line, and using application servers such as Tomcat, JBoss, or BEA WebLogic.

## 1.3 Related documents

### ◆ KeyMaster Overview

Describes what KeyMaster is and what it can be used for, the architecture of the product, how it fits into the overall Caplin product architecture and third party/customer systems, and key concepts relating to the product. It also gives some examples of how the product can be used in real business situations.

### ◆ KeyMaster Java API Reference.

Defines the public Java classes and interfaces available in Java KeyMaster. Refer to it when customizing KeyMaster Java code.

### ◆ KeyMaster.NET API Reference

Defines KeyMaster.NET classes and interfaces that can be used to implement a Microsoft .NET application that generates KeyMaster user credentials tokens.

### ◆ XML Auth Administration Guide

Describes the XMLauth Module, and how it enables programmers and system administrators to use XML to create their own permissioning structures and control entitlement to objects held on Caplin Liberator.

◆ **Liberator Administration Guide**

Describes how to install and configure the Caplin Liberator server. It includes full reference information for the Liberator configuration.

◆ **Liberator Authentication C API Reference**

Describes how to implement custom authentication modules for Liberator, in C code.

◆ **JavaAuth API Reference**

Describes the library of classes (javaauth) that enables developers to create custom authentication modules for Liberator in Java.

◆ **StreamLink for Browsers API Reference**

The API reference documentation for StreamLink for Browsers. In release 4.5.2 and upwards, the section on “Using SL4B With KeyMaster” explains how StreamLink for Browsers can be configured to use KeyMaster for logging in to the Liberator.

◆ **StreamLink for Java API Reference**

The API reference documentation for StreamLink for Java. The section on “Caplin KeyMaster integration” explains how KeyMaster authentication can be integrated into web client applications that use StreamLink for Java.

◆ **StreamLink.NET API Reference**

The API reference documentation for StreamLink .NET.

◆ **StreamLink for Silverlight API Reference**

The API reference documentation for StreamLink for Silverlight.

## 1.4 Typographical conventions

The following typographical conventions are used to identify particular elements within the text.

<i><b>Type</b></i>	<i><b>Uses</b></i>
<b>aMethod</b>	Function or method name
<i>aParameter</i>	Parameter or variable name
<i>/AFolder/Afile.txt</i>	File names, folders and directories
<div style="border: 1px solid black; padding: 2px;">Some code;</div>	Program output and code examples
The value=10 attribute is...	Code fragment in line with normal text
Some text in a dialog box	Dialog box output
Something typed in	User input – things you type at the computer keyboard
<b>XYZ Product Overview</b>	Document name
◆	Information bullet point
■	Action bullet point – an action you should perform

**Note:** Important Notes are enclosed within a box like this.  
Please pay particular attention to these points to ensure proper configuration and operation of the solution.

**Tip:** Useful information is enclosed within a box like this.  
Use these points to find out where to get more help on a topic.

Information about the applicability of a section is enclosed in a box like this.  
For example: "This section only applies to version 1.3 of the product."

## 1.5 Feedback

Customer feedback can only improve the quality of our product documentation, and we would welcome any comments, criticisms or suggestions you may have regarding this document.

Visit our feedback web page at <https://support.caplin.com/documentfeedback/>.

## 1.6 Acknowledgments

*Adobe® Reader* is a registered trademark of Adobe Systems Incorporated in the United States and/or other countries.

*Windows* is a registered trademark of Microsoft Corporation in the United States and other countries.

*Silverlight* is a trademark of Microsoft Corporation in the United States and other countries.

*Sun, Solaris and Java*, are trademarks or registered trademarks of Oracle® Corporation in the U.S. and other countries.

*Linux®* is the registered trademark of Linus Torvalds in the U.S. and other countries.

*RSA®* is a registered trademark of RSA Security Inc.

*nCipher* is a trademark or registered trademark of nCipher Corporation Ltd.

*thawte* is a registered trademark of VeriSign in the United States and/or other countries.

## 1.7 Open Source Software

This Caplin component incorporates the following Open Source software:

Open Source item	Use in KeyMaster	Further information
Public key encryption software from The Legion Of The Bouncy Castle.	Used in Standard KeyMaster to generate encryption key pairs, and encrypt and decrypt digital signatures using these keys.	<a href="http://www.bouncycastle.org">www.bouncycastle.org</a>
OpenSSL	Cryptographic algorithms from the OpenSSL crypto library, and the OpenSSL commands for generating RSA key pairs and self-signed certificates.	<a href="http://www.openssl.org">www.openssl.org</a>



## 2 Overview

Caplin KeyMaster is used to integrate Caplin Liberator with an existing single sign-on system, so that end users do not need to explicitly log in to a Liberator in addition to their normal log in procedure. It provides a more secure and convenient authentication method than just using simple user names and passwords.

KeyMaster implements a secure method of user authentication via a user credentials token that is digitally signed using public key encryption.

It comprises two tools for enabling users to be authenticated: a key generator and a signature generator.

### ◆ Key Generator

The Key Generator is an application used to create an encryption key pair; one key is the private key and the other is the public key. KeyMaster uses the private key to sign a user credentials token that authenticates a user's access to the Liberator. The public key is exported to the data provider's Liberator for use during the authentication process.

A useable Key Generator is provided with the Standard KeyMaster product, as a Java servlet. Key pairs can also be generated using third-party tools, such as the OpenSSL key generation commands (see [www.openssl.org](http://www.openssl.org)). This is necessary if the KeyMaster Signature Generator is implemented using KeyMaster.NET (see the **KeyMaster.NET API Reference**), or if KeyMaster is to be integrated with a secure key storage hardware module ([see Integrating KeyMaster with a hardware Key Store](#)<sup>[41]</sup>).

### ◆ Signature Generator

The Signature Generator creates a user credentials token, which it digitally signs (encrypts) using the KeyMaster private key. This token is used by Caplin Liberator to validate an end-user's login to the Liberator server.

The Signature Generator is usually an application server module. A useable Signature Generator is provided with the Standard KeyMaster product, as a Java servlet. You can customize this servlet as required, or you can use it to guide the design of a similar module in another technology.

Using KeyMaster.NET, you can also implement the Signature Generator as a Microsoft .NET application (typically deployed as an ASP.NET web page); for more information, see the **KeyMaster.NET API Reference**.

Subsequent sections of this guide describe how to configure and operate these two tools.

**Note:** If you are unfamiliar with KeyMaster, you are recommended to read the **KeyMaster Overview** before reading this Administration Guide.

## 3 Technical assumptions and restrictions

### Platforms and Java

KeyMaster is supported on the following operating systems:


- ◆ Linux®
- ◆ Sun® Solaris™
- ◆ Microsoft® Windows® XP
- ◆ Microsoft Windows 2000
- ◆ Microsoft Windows 2003 Server

All these platforms must run the Java Runtime Environment (JRE™) or Java Development Kit (JDK™). The Java version must be at least the greater of:

- ◆ Java version 1.6
- ◆ The minimum Java version specified for the web application server under which KeyMaster will be deployed.

### Encryption software

The following encryption software is used in Standard KeyMaster:

- ◆ Public key encryption software from The Legion Of The Bouncy Castle  
See [Open Source Software](#) .
- ◆ The Key Generator generates the key pair using the Sun® **SHA1PRNG** secure random number generator and the RSA® key pair generation algorithm.
- ◆ Digital signatures are generated using the **MD5withRSA** algorithm.

For an explanation of this algorithm see the **KeyMaster Overview**.

**Note: MD5 limitations:** Since KeyMaster was first released, the cryptographic community have found that the MD5 algorithm can produce hash collisions. This potentially compromises the algorithm.

Caplin has retained MD5withRSA as the default digital signature algorithm for backward compatibility with previous versions of KeyMaster. *However, customers installing KeyMaster for the first time may wish to configure the software to use a more secure algorithm, such as SHA256.*

## Web application server versions

The instructions in this document for deploying KeyMaster on various web application servers (see [Deploying KeyMaster](#)<sup>(13)</sup>) assume the following server versions:

Server	Minimum version
Tomcat	5.0.16
JBoss	4.0.0
BEA WebLogic	8.1

## 4 Installing KeyMaster

These sections do not apply to KeyMaster.NET.

### 4.1 Important note on security of installation

**Note:** The instructions in this guide describe how to set up KeyMaster so that it can be tested easily. *However, in this state KeyMaster is not secure, and it should not be used in a production environment.*

To deploy KeyMaster securely you will need to integrate it with your single sign-on system in a way that is compliant with your organization's security policies, and is compatible with the web application server that hosts the KeyMaster servlets.

You may also wish to configure the software to use a more secure digital signature algorithm, such as SHA256.

For more information, see [Making KeyMaster production ready](#) <sup>38</sup>.

To ensure that KeyMaster is deployed in a highly secure manner, you may wish to store the private keys in a dedicated secure hardware module (a “**Key Store**”), instead of on disk. To do this:

- Follow the installation and configuration instructions for Standard KeyMaster:
  - [Installing KeyMaster](#) <sup>8</sup>
  - [Deploying KeyMaster](#) <sup>13</sup>
  - [Setting up Liberator to work with KeyMaster](#) <sup>25</sup>
  - [Testing Java-based KeyMaster with Liberator](#) <sup>32</sup>
- Then follow the instructions in [Integrating KeyMaster with a hardware Key Store](#) <sup>41</sup>.

### 4.2 Prerequisites

- Before installing Java-based KeyMaster on your machine, make sure there is a suitable version of the Java Runtime Environment (JRE) or Java Development Kit (JDK) installed; see [Technical assumptions and restrictions](#) <sup>6</sup>.

**Note:** On Microsoft Windows platforms there is a Microsoft version of the Java Virtual Machine (JVM) built into some versions of Windows Internet Explorer®. However, the web application server may not be able to run using this JVM. Check the installation requirements for your web application server – you may need to install the JRE or JDK from Sun Microsystems. See [Technical assumptions and restrictions](#) <sup>6</sup>.

### 4.3 Installing on Linux or Sun Solaris

The install kit is contained in a zip file called *KeyMaster-<version\_number>.zip*

1. Copy the zip file to a base directory where you want the installed software to be located, such as */apps/caplin*

Make sure that the directory and its sub-directories are accessible from your chosen application server (see [Deploying KeyMaster](#)<sup>[13]</sup>).

2. Unzip the file:

```
unzip KeyMaster-<version_number>.zip
```

The software will be unzipped into a new directory  
*/KeyMaster-<version\_number>*  
under your base directory,  
for example */apps/caplin/KeyMaster-4.4.0*

**Note:** In the rest of this guide the directory where the KeyMaster software is located is referred to as *\$KM\_INSTALL\_DIR*  
For example, *\$KM\_INSTALL\_DIR* could refer to  
*/apps/caplin/KeyMaster-4.4.0*

**Tip:** When you have finished installing KeyMaster read the release note  
*\$KM\_INSTALL\_DIR/RELEASENOTE.txt*  
before proceeding any further.

- Now see the section [Installed Files](#)<sup>[10]</sup>.

## 4.4 Installing on a Windows platform

The install kit is contained in a zip file called *KeyMaster-<version\_number>.zip*

1. Copy the zip file to a base directory where you want the installed software to be located, such as *C:\Program Files\apps\Caplin*

Make sure that the directory and its sub-directories are accessible from your chosen application server (see [Deploying KeyMaster](#)<sup>[13]</sup>).

2. Unzip the file using a suitable zip utility

The software will be unzipped into a new directory  
*\KeyMaster-<version\_number>* under your base directory,  
for example *C:\Program Files\apps\Caplin\KeyMaster-4.4.0*

**Note:** In the rest of this guide the directory where the KeyMaster software is located is referred to as *\$KM\_INSTALL\_DIR*  
For example, *\$KM\_INSTALL\_DIR* could refer to  
*C:\Program Files\apps\Caplin\KeyMaster-4.4.0*

**Tip:** When you have finished installing KeyMaster read the release note  
*\$KM\_INSTALL\_DIR/RELEASENOTE.txt*  
before proceeding any further.

- Now see the section [Installed Files](#)<sup>[10]</sup>.

## 4.5 Installed Files

*KeyMaster-4.4.x-xx.zip* contains the following files:

- ◆ *examples/flatfile/FlatFileServlet.java*
  - ◆ *examples/keygen.props*
  - ◆ *examples/keyimporter/KeyImporter.java*
  - ◆ *examples/keyimporter/KeyImportVerifier.java*
  - ◆ *examples/keyimporter.props*
  - ◆ *examples/news/NewsFormatter.java*
  - ◆ *examples/usercredentials/ExampleCredentialsProvider.java*
  - ◆ *doc/\** (The KeyMaster documentation set)
  - ◆ *deploy/keymaster.war*
  - ◆ *lib/keyMaster.jar*
  - ◆ *lib/bcprov-jdk<version>.jar* (The BouncyCastle encryption JAR )
  - ◆ *RELEASENOTE.txt*
  - ◆ *README.txt*
- Check the release notes in *RELEASENOTE.txt* for important information about the KeyMaster release that you are installing.
  - Now follow the instructions in [Generating the required keys](#) <sup>10</sup>.

## 4.6 Generating the required keys

KeyMaster uses three encryption key files: a public key file, a private key file, and a DER key file (which is a binary version of the public key). These key files must be generated before KeyMaster authentication can be used.

Follow these steps to generate the keys:

1. Make sure you have the following JAR files. These should be present in the *lib* directory of your KeyMaster installation:
  - ◆ The BouncyCastle encryption JAR (*bcprov-jdk<version>.jar*).
  - ◆ The KeyMaster JAR containing all the classes KeyMaster needs to execute (*keymaster.jar*).
2. KeyMaster uses a properties file *keygen.props* that is used to initialize and configure the Key Generator. There is an example *keygen.props* file in *\$KM\_INSTALL\_DIR/examples/*. You can use this file for your KeyMaster installation, if it is suitable for your needs. Alternatively you can create your own version of this file; refer to the [keygen.props configuration reference](#) <sup>61</sup>.

**Note:** The rest of this installation guide assumes the names of the encryption key files are as defined in the example version of *keygen.props*, namely *privatekey.store*, *publickey.store* and *publickey.der*

3. Check whether `$KM_INSTALL_DIR` contains existing key files that have the same name as the key files that are going to be created. For example, if you are going to use the example `keygen.props` properties file provided with the install kit, then check `$KM_INSTALL_DIR` for files called `privatekey.store`, `publickey.store` and `publickey.der`. If any or all of these files are already present then delete, move or rename them.
4. Now run the Key Generator. This is the Java class **com.caplin.keymaster.keygenerator.KeyGenerator**, included in `$KM_INSTALL_DIR/lib/keymaster.jar`

The Key Generator requires two command line arguments:

- ◆ The name of the properties file that will be used to create the keys
- ◆ An identifier for the key

The identifier can be set to any string value. It will be referred to in the `web.xml` file that configures the KeyMaster Signature Generator used to generate user credentials tokens – see [Modifying the web.xml configuration file](#)<sup>[17]</sup>.

The format of the command to run the Key Generator is:

**On Linux or Sun Solaris:**

```
java -classpath lib/bcprov-jdk14-125.jar:lib/keymaster.jar  
com.caplin.keymaster.keygenerator.KeyGenerator <properties-file> <key-identifier>
```

**On Windows:**

```
java -classpath lib/bcprov-jdk14-125.jar;lib/keymaster.jar  
com.caplin.keymaster.keygenerator.KeyGenerator <properties-file> <key-identifier>
```

Run the command from the `$KM_INSTALL_DIR` directory.

For example, to run the Key Generator using the example properties file, the command is:

**On Linux or Sun Solaris:**

```
java -classpath lib/bcprov-jdk14-125.jar:lib/keymaster.jar  
com.caplin.keymaster.keygenerator.KeyGenerator examples/keygen.props keyidl
```

**On Windows:**

```
java -classpath lib/bcprov-jdk14-125.jar;lib/keymaster.jar  
com.caplin.keymaster.keygenerator.KeyGenerator examples/keygen.props keyidl
```

If the command runs successfully the Key Generator displays this message:

```
"Retrieved a KeyStoreElement containing a public key from input key store  
for name keyidl"
```

**Tip:** Make a note of the key identifier that you specified in the key generator command (`keyidl` in the example above). You will need it later when you set up the `web.xml` file that configures the KeyMaster Signature Generator.  
See [Modifying the web.xml configuration file](#)<sup>[17]</sup>, and the parameter `encrypting.generator.key.identifier` in [web.xml parameters](#)<sup>[73]</sup>.

`$KM_INSTALL_DIR` should now contain the newly generated key files: `privatekey.store`, `publickey.store` and `publickey.der`.

If a key file with the same name as that specified in `keygen.props` already exists in `$KM_INSTALL_DIR`, then the Key Generator will overwrite the old key file and will display messages of the following form:

```
Adding the private key replaced the existing key for the same server:
com.caplin.keymaster.encrypted.PrivateKeyStoreElement@d1fa5
Retrieved a KeyStoreElement containing a public key from input key store
for name keyidl
Adding the public key replaced the existing key for the same server:
com.caplin.keymaster.encrypted.PublicKeyStoreElement@134e4fb
```



## 5 Deploying KeyMaster

These sections do not apply to KeyMaster.NET.

The following sections explain how to deploy KeyMaster on a number of different application servers. Once you have installed and configured the server, and configured KeyMaster to work with the server, the run-time module of KeyMaster will be able to generate user credentials tokens for the server to pass back to requesting clients.

### 5.1 Deployment on a Tomcat server

To deploy KeyMaster on a Tomcat web application server, carry out the following steps. These instructions assume you are deploying KeyMaster on a Tomcat server running under Linux or Sun Solaris.

**Note:** Make sure your version of the Tomcat server is at least the minimum specified in [Technical assumptions and restrictions](#) <sup>6</sup>.

1. Install Tomcat to your desired directory (called `$SERVER_HOME` in the rest of these instructions) and configure it as required.
2. You may want to start up Tomcat to test that it is working. If you do, then shut it down before proceeding with the next steps.
  - Start the Tomcat server by going to the directory `$SERVER_HOME/bin` and entering the command:  

```
./startup.sh
```
  - To stop the Tomcat server, go to the directory `$SERVER_HOME/bin` and enter the command:  

```
./shutdown.sh
```
3. Open the `$SERVER_HOME` directory.
4. Go to the directory `webapps`.
5. Copy the file `$KM_INSTALL_DIR/deploy/keymaster.war` to the current directory (`webapps`).  
The `keymaster.war` file contains files relating to the KeyMaster Signature Generator that will be unpacked into a directory called `keymaster` and used at run time.
6. Check whether the `webapps` directory already contains a directory called `keymaster`. If it does, then either delete the `keymaster` directory or change its name, as appropriate.
7. Start the Tomcat server.
8. Tomcat will now deploy the `keymaster.war` file and create the required KeyMaster structures in `$SERVER_HOME/webapps/keymaster/`
9. Go to the directory `$SERVER_HOME/webapps/keymaster/WEB-INF/`  
This directory contains a configuration file called `web.xml`. This file needs to be edited, so that KeyMaster knows where to find the private encryption key in order to generate user credentials tokens.
  - Follow the instructions in [Modifying the web.xml configuration file](#) <sup>17</sup>.

## 5.2 Deployment on a JBoss server

To deploy KeyMaster on a JBoss web application server, carry out the following steps. These instructions assume you are deploying KeyMaster on a JBoss server running under Linux or Sun Solaris.

**Note:** Make sure your version of the JBoss server is at least the minimum specified in [Technical assumptions and restrictions](#)<sup>[6]</sup>.

1. Install JBoss to your desired directory (called *\$SERVER\_HOME* in the rest of these instructions) and configure it as required.
2. You may want to start up JBoss to test that it is working. If you do, then shut it down before proceeding with the next steps.
  - Start the JBoss server by going to the directory *\$SERVER\_HOME/bin* and entering the command:  

```
./run.sh
```

Control will not be returned to your command window until you have stopped the server.
  - To stop the JBoss server, enter the command:  

```
ctrl/C
```
3. Open the *\$SERVER\_HOME* directory.
4. Go to the directory *server/default/deploy/*
5. Create a directory called *keymaster.war* and go to this directory.
6. Copy the file *\$KM\_INSTALL\_DIR/deploy/keymaster.war* to the current directory (*keymaster.war*).  
The *keymaster.war* file contains files relating to the KeyMaster Signature Generator that will be unpacked into a directory called *keymaster* and used at run time.
7. Extract the files from the *keymaster.war* file, using the Java jar command:  

```
jar -xvf keymaster.war
```
8. Go to the directory *\$KM\_INSTALL\_DIR/deploy/WEB-INF*.  
In this directory is a configuration file called *web.xml*. This file needs to be edited so that KeyMaster knows where to find the private encryption key in order to generate user credentials tokens.
  - Follow the instructions in [Modifying the web.xml configuration file](#)<sup>[17]</sup>.

## 5.3 Deployment on a BEA WebLogic server

This section explains how to implement a basic deployment of KeyMaster on a BEA WebLogic application server.

### Deployment on WebLogic 9.1

To deploy KeyMaster on a BEA WebLogic 9.1 application server, carry out the following steps.

#### Set up KeyMaster files

1. Create a KeyMaster deployment directory path in a location that can be accessed by the WebLogic server. The name of the lowest level directory in the path should be *keymaster.war*.  
For example */Caplin/KeyMasterDeploy/keymaster.war/*  
Go to this directory
2. Copy the file *\$KM\_INSTALL\_DIR/deploy/keymaster.war* to the current directory (*keymaster.war*).  
The *keymaster.war* file contains files relating to the KeyMaster Signature Generator that will be unpacked into a directory called *keymaster* and used at run time.
3. Extract the files from the *keymaster.war* file, using the Java jar command:  

```
jar -xvf keymaster.war
```
4. Copy the private encryption key file *privatekey.store* key from *\$KM\_INSTALL\_DIR* to the top level of the KeyMaster deployment directory.  
For example, copy the key file to */Caplin/KeyMasterDeploy/*

#### Deploy KeyMaster in WebLogic

1. Start the WebLogic server for the domain you wish to use  
The example domain installed with WebLogic version 9.1 is *wl\_server*.
4. Start up the Server Administration Console by opening a web browser and entering a URL of the form:  

```
http://server-address:7001/console
```

  
(Assuming that you have not changed the default port number of 7001.)
5. Log in and then under 'Domain Structure' click the link for 'Deployments'.
6. Click the Lock & Edit button to enable the Install button, then Click the Install button.
7. Browse to *\$KM\_INSTALL\_DIR/deploy/* and select the *keymaster.war* file.
8. Click Next.  
A page called 'Choose targeting style' is displayed.
9. Select the choice labeled 'Install this deployment as an application' and click Next.  
The 'Optional Settings' page is displayed.

10. The default deployment settings will be used, so click **Next**

The 'Review your choices page' is displayed.

The summary of choices will look like this:

**Deployment:** <file-path-to-KeyMaster-deployment>/keymaster.war  
**Name:** KeyMaster  
**Staging mode:** Use the defaults defined by the chosen targets  
**Security Model:** DDOnly: Use only roles and policies that are defined in the deployment descriptors.

11. Click the **Finish** button.

The 'Settings for KeyMaster page' is displayed.

12. Click the **Save** button.

The message '**Settings updated successfully**' should be displayed in the Messages area at the top of the page.

13. There will be a message in the Change Center bar on the top left hand side of the page:  
'Pending changes exist. They must be activated to take effect.'

Click the **Activate Changes** button (located below the message).

14. Now start the KeyMaster web application as follows:

15. Click on the **Control** tab

The displayed page should show that the KeyMaster application is in the state 'Prepared'.

16. Click the **Lock & Edit** button, to enable the **Start** button.

17. Select the KeyMaster application

18. Click the **Start** button and select 'Servicing all requests'.

The 'Start Application Assistant' page is displayed.

19. Click the **Yes** button

The message '**Start requests have been sent to the selected Deployments**' should be displayed in the Messages area at the top of the page.

The KeyMaster state should now be 'Active'.

20. Click the **Release Configuration** button

21. Close down the WebLogic Administration Console by clicking on the **Log out** tab.

22. Go to the directory `$KM_INSTALL_DIR/deploy/WEB-INF`.

In this directory is a configuration file called `web.xml`. This file needs to be edited so that KeyMaster knows where to find the private encryption key in order to generate user credentials tokens.

Follow the instructions in [Modifying the web.xml configuration file](#)<sup>17</sup>.

## Deployment on WebLogic 8.1

To deploy KeyMaster on a BEA WebLogic 8.1 application server, carry out the following steps.

1. Start the server for the domain you wish to use (the example domain installed with WebLogic is *mydomain*).
2. Open up a web browser and go to the URL of the console for the desired domain.  
For example:  
`http://xyz:7001/console`  
(Assuming that the default port number has not been changed.)
3. Log in and then go to “Your Deployed Resources” and underneath click the link for “Applications”.
4. Select “Deploy a new Application” and then click on applications.  
Following this click on “upload your files”.
5. A screen that tells you the types of files you can upload will now be displayed.  
KeyMaster currently has a war format .
6. Browse to the location of the *keymaster.war* file and then select “open”.  
Now click the upload button and the file will be copied to the relevant location.
7. The server will extract the files from the *.war* file and put them in  
*/mydomain/myserver/stage/\_appsdir\_keymaster\_war/keymaster.war/WEB-INF/*
8. One of the files extracted to the *WEB-INF* directory is called *web.xml*. This file needs to be edited, so that KeyMaster knows where to find the private encryption key in order to generate user credentials tokens.

Follow the instructions in [Modifying the web.xml configuration file](#)<sup>[17]</sup>.

## 5.4 Modifying the web.xml configuration file

This section and its subsections do not apply to KeyMaster.NET.

The *web.xml* configuration file needs to be edited, so that the KeyMaster Signature Generator knows where to find the private encryption key in order to generate user credentials tokens. You also specify in here the servlet's error logging environment.

The *web.xml* file is located in a directory of your web application server; the precise directory path depends on which application server you are using, but it ends in *WEB-INF/*. For example, if KeyMaster is deployed on a Tomcat application server, *web.xml* is in the directory *\$SERVER\_HOME/webapps/keymaster/WEB-INF/*.

Edit *web.xml* as follows:

1. Find the entry called `encrypting.generator.private.key.store.filename:`

```
<init-param>
  <param-name>encrypting.generator.private.key.store.filename</param-name>
  <param-value>privatekey.store</param-value>
  <description>File name and location for the private key</description>
</init-param>
```

Change the content of the `<param-value>` tag to specify the name and location of the previously created private key file (see [Generating the Required Keys](#)<sup>[10]</sup>). The directory will normally be the `$KM_INSTALL_DIR` directory, but it may be a different location depending on the application server being used (see the deployment instructions for your application server in [Deploying KeyMaster](#)<sup>[13]</sup>).

**Example:**

```
<init-param>
  <param-name>encrypting.generator.private.key.store.filename</param-name>
  <param-value>/Caplin/KeyMaster/privatekey.store</param-value>
  <description>File name and location for the private key</description>
</init-param>
```

In this example the `$KM_INSTALL_DIR` directory is `/Caplin/KeyMaster/`

2. Find the entry called `encrypting.generator.key.identifier`.  
This needs to be changed to the key identifier that was passed as the second argument to the KeyMaster Key Generator when the key was created (see [Generating the required keys](#)<sup>[10]</sup>).

**Example:**

```
<init-param>
  <param-name>encrypting.generator.key.identifier</param-name>
  <param-value>keyid1</param-value>
  <description>Name of the server the token is generated for.</description>
</init-param>
```

3. Change the name and location of the KeyMaster Signature Generator's error log file as required.

This is specified in the `web.xml` entry called `key.generator.FileNameAttribute`. If the entry does not contain a file path, then the log file will be located in a default directory whose location depends on the type of web application server:

- ◆ On a Tomcat or JBoss server this is the `'bin'` directory where the web application server was started.
- ◆ On a BEA WebLogic server this is the domain directory where WebLogic was started.

**Example:**

```
<init-param>
  <param-name>key.generator.FileNameAttribute</param-name>
  <param-value>servlet.log</param-value>
  <description>KeyMaster log file</description>
</init-param>
```

4. Specify the KeyMaster Signature Generator's logging level.

The logging level is specified in the `web.xml` entry called `key.generator.Level`. The level should be one of the logging levels defined in the Java class `java.util.logging.Level`. It is recommended that you initially set the logging level to `ALL`, so that KeyMaster will record the maximum amount of information about any installation problems. When you are satisfied that KeyMaster is behaving correctly, it is recommended that you change the logging level to `WARNING`. For more information on setting logging levels see the Servlet Configuration section in the **KeyMaster Java API Reference**.

**Example:**

```
<init-param>
  <param-name>key.generator.Level</param-name>
  <param-value>ALL</param-value>
  <description>KeyMaster logging level</description>
</init-param>
```

**Tip:** For more information on configuring the KeyMaster servlet in the *web.xml* file, see the [web.xml configuration reference](#) <sup>[65]</sup> section.

5. If you want client applications to access the KeyMaster servlet from a different URL, follow the instructions in [Changing KeyMaster's URL](#) <sup>[20]</sup>.
6. For the configuration changes to take effect, you will need to reload the KeyMaster web application and possibly restart the application server.  
  
If you are deploying KeyMaster to a newly installed JBoss server, start the server once you have modified *web.xml*.
7. Now check that KeyMaster has been correctly deployed on the web application server, by following the instructions in [Testing KeyMaster with the application server](#) <sup>[23]</sup>.

## Changing KeyMaster's URL

Standard Java-based KeyMaster's Signature Generator is accessed from clients through the URL /servlet/StandardKeyMaster. You may wish to access this servlet from a different URL; for example, because you have customized the servlet and wish to distinguish it from the standard one.

To configure a different URL, edit the *web.xml* configuration file, as follows:

1. Find the `<servlet-mapping>` tag with the child `<servlet-name>` tag whose content is "StandardKeyMaster":

```
<servlet-mapping>
  <servlet-name>StandardKeyMaster</servlet-name>
  <url-pattern>/servlet/StandardKeyMaster</url-pattern>
</servlet-mapping>
```

Change the URL in the `<url-pattern>` tag that follows the `<servlet-name>` to the new URL:

```
<servlet-mapping>
  <servlet-name>StandardKeyMaster</servlet-name>
  <url-pattern>/servlet/CustomizedKeyMasterName</url-pattern>
</servlet-mapping>
```

2. If client applications will access KeyMaster through StreamLink for Browsers 4.5.2 or later (for example, when the client application is Caplin Trader version 1.4.2 or later), you must also declare the new URL for access by the XHRKeymaster servlet.

Find the `<servlet-name>` defining XHRKeymaster, and locate the child `<param-name>` tag defining the parameter `keymaster.url`:

```
<servlet-name>XHRKeymaster</servlet-name>
<init-param>
  <param-name>keymaster.url</param-name>
  <param-value>/servlet/StandardKeyMaster</param-value>
  <description>The url of the Standard KeyMaster page,
    for XHRKeyMaster to attach to.</description>
</init-param>
```

Change the `<param-value>` to the new URL of the customized KeyMaster:

```
<servlet-name>XHRKeymaster</servlet-name>
<init-param>
  <param-name>keymaster.url</param-name>
  <param-value>/servlet/CustomizedKeyMasterName</param-value>
  <description>The url of the customized KeyMaster page,
    for XHRKeyMaster to attach to.</description>
</init-param>
```

Also see the [web.xml configuration reference](#)<sup>65)</sup> section, and the definition of the *web.xml* parameter `keymaster.url`<sup>84)</sup>



## Changing the KeyMaster Poll servlet's URL

When clients access Java-based KeyMaster through StreamLink for Browsers version 4.5.2 or later (for example, when the client application is Caplin Trader version 1.4.2 or later), StreamLink for Browsers accesses a servlet called Poll at regular intervals, so as to keep the session with the KeyMaster servlet alive.

If you wish to access the Poll servlet from a different URL, you must configure the URL by editing the *web.xml* configuration file, as follows:

1. Find the `<servlet-mapping>` tag with the child `<servlet-name>` tag whose content is "Poll":

```
<servlet-mapping>
  <servlet-name>Poll</servlet-name>
  <url-pattern>/servlet/Poll</url-pattern>
</servlet-mapping>
```

Change the URL in the `<url-pattern>` tag that follows the `<servlet-name>` to the new URL:

```
<servlet-mapping>
  <servlet-name>Poll</servlet-name>
  <url-pattern>/servlet/newPollLocation</url-pattern>
</servlet-mapping>
```

2. You must also declare the Poll servlet's new URL for access by the XHRKeymaster servlet.

Find the `<servlet-name>` defining Poll, and locate the child `<param-name>` tag defining the parameter `keymaster.poll.url`:

```
<servlet-name>XHRKeymaster</servlet-name>
<init-param>
  <param-name>keymaster.poll.url</param-name>
  <param-value>/servlet/Poll</param-value>
  <description>The url of the KeyMaster polling page,
    for XHRKeyMaster to attach to.</description>
</init-param>
```

Change the `<param-value>` to the new Poll URL:

```
<servlet-name>XHRKeymaster</servlet-name>
<init-param>
  <param-name>keymaster.poll.url</param-name>
  <param-value>/servlet/newPollLocation</param-value>
  <description>The url of the KeyMaster polling page,
    for XHRKeyMaster to attach to.</description>
</init-param>
```

Also see the [web.xml configuration reference](#)<sup>65</sup> section, and the definition of the *web.xml* parameter `keymaster.poll.url`<sup>85</sup>

## Changing the KeyMaster Dependencies servlet's URL

When clients access Java-based KeyMaster through StreamLink for Browsers version 4.5.2 or later (for example, when the client application is Caplin Trader version 1.4.2 or later), KeyMaster uses a servlet called Dependencies.

If you have changed the location of the XHRKeymaster servlet so that its URL is not `/servlet/XHRKeymaster`, you must change the location of the Dependencies servlet to match the new location of XHRKeymaster.

For example, if XHRKeymaster has been moved to the URL `/mylocation/myservlets/XHRKeymaster`, the Dependencies servlet must be moved to the URL `/mylocation/myservlets/dependencies/*`.

**Note:** Do not rename or modify the Dependencies servlet, otherwise KeyMaster will not work correctly with StreamLink for Browsers.

To specify the new URL for the Dependencies servlet, edit the *web.xml* configuration file, as follows:

- Find the `<servlet-mapping>` tag with the child `<servlet-name>` tag whose content is "Dependencies":

```
<servlet-mapping>
  <servlet-name>Dependencies</servlet-name>
  <url-pattern>/servlet/dependencies/*</url-pattern>
</servlet-mapping>
```

- Change the URL in the `<url-pattern>` tag that follows the `<servlet-name>` to the new URL:

```
<servlet-mapping>
  <servlet-name>Poll</servlet-name>
  <url-pattern>/mylocation/myservlets/dependencies/*</url-pattern>
</servlet-mapping>
```

Also see the [web.xml configuration reference](#) <sup>65</sup> section.

## 5.5 Testing KeyMaster with the application server

1. Start up a web browser.
2. If you are using a Tomcat, JBoss or BEA WebLogic application server, enter the address of your web application server, and (if required) its port number, followed by:

`/keymaster/servlet/StandardKeyMaster?username=aname`

where `aname` is any user name.

For example:

`http://myserver.abc.com:9127/keymaster/servlet/StandardKeyMaster?username=davids`

(Later on, when you set up the Liberator to work with KeyMaster, you will configure, in the `users.xml` file of XMLauth, the valid set of user names for authentication by KeyMaster – see [Modifying the users.xml authorization file](#)<sup>[29]</sup>.)

3. If the test has worked, text with the following format should be displayed on the screen:

```
credentials=ok
username=davids
token=0laUyFtRh1xkrZD85ASoegZtBc4C3gQBivbM
zXADZQHeM1/knTv3vyUUelazdharQHysA89zMptD+T4F8pGCjkY1tejaq/
VAATgnVkyetjs33NEIulsL0zXaSPxZs8sQ0zOG8v7E/
uV3Da46FB+jm1Z6VwJN7ioQWDD7SYKZaJ8
```

4. If the test fails, any errors will be logged in the servlet log specified in the `web.xml` file.
5. If clients will be accessing KeyMaster through StreamLink for Browsers version 4.5.2 or later (for example, when the client application is Caplin Trader version 1.4.2 or later), also follow the instructions in [Testing the XHRKeymaster servlet](#)<sup>[24]</sup>.
6. If you are integrating KeyMaster with a hardware Key Store, now follow the instructions in [Configuring Liberator to use a new public key](#)<sup>[52]</sup> otherwise follow the instructions in [Setting up Liberator to work with KeyMaster](#)<sup>[25]</sup>.

## Testing the XHRKeymaster servlet

This test is only needed if clients will need to access KeyMaster through StreamLink for Browsers version 4.5.2 or later (for example if you are using Caplin Trader version 1.4.2 or later). Make sure you have run the general application server test first – see [Testing KeyMaster with the application server](#)<sup>[23]</sup>.

1. In the web browser enter the address of your web application server, and its port number (if required), followed by the relative URL of the XHRKeymaster servlet, which is:

`/keymaster/servlet/XHRKeymaster`

For example, enter the web address:

`http://myserver.abc.com:9127/keymaster/servlet/XHRKeymaster`

You should see a web page with a single text box, containing the default user name "fred" and three buttons. The buttons allow you to test the XHRKeymaster's connection to the StandardKeyMaster.

<input type="text" value="fred"/>		
KeyMaster.requestToken	KeyMaster.startKeepAlive	KeyMaster.stopKeepAlive

### XHRKeyMaster test page

2. Click the button labeled KeyMaster.requestToken.

After a short pause, an alert should appear containing text with the following format:

```
Success: fred
VffpUBLvSArQhJbrTIpJ5aTgaPgbbBLmZuzpp/3niQHMRyVeU/K/+0H1XqJlJEjdH2M9r20p+
XdGGGcM1VsmRDHfcHnz hFN/ASwn/CU2vsIkH6SZMOPak++AKdSUX8q0Nuj/Zqa54Ip9G65sp
M0VOdM2hxTnIkNkSY0DextmNsw=~20090415120338~1~fred
```

3. If you are integrating KeyMaster with a hardware Key Store, now follow the instructions in [Configuring Liberator to use a new public key](#)<sup>[52]</sup> otherwise follow the instructions in [Setting up Liberator to work with KeyMaster](#)<sup>[25]</sup>.

## 6 Setting up Liberator to work with KeyMaster

This section and its subsections apply to Java-based KeyMaster and to KeyMaster.NET.

Once the encryption keys have been generated (see [Generating the Required Keys](#)<sup>[10]</sup>), you need to set up your Liberator so that it can work with KeyMaster.

To do this you:

- ◆ make the public key file available to Liberator,
- ◆ modify the Liberator configuration file *rttd.conf*,
- ◆ modify the *users.xml* authorization file if the Liberator uses the XMLauth module to authenticate users, or
- ◆ modify the *cfgauth.conf* authorization file if the Liberator uses the cfgauth module to authenticate users.

When you have completed the Liberator set up, follow the instructions in:

- ◆ [Testing Java-based KeyMaster with Liberator](#)<sup>[32]</sup> (if you have installed Java-based KeyMaster)
- or
- ◆ [Testing KeyMaster.NET with Liberator](#)<sup>[37]</sup> (if you have installed a KeyMaster Signature Generator developed using KeyMaster.NET).

### 6.1 Making the public key file available to Liberator

Copy the DER public key file in the KeyMaster *\$KM\_INSTALL\_DIR* directory to Caplin Liberator. The name of this file ends in *.der*, for example *publickey.der*.

Put the file in the */etc* directory of the Liberator installation.

**Tip:** If you copy the file using FTP (because the KeyMaster and Liberator are on different machines), make sure you use binary mode, as the DER file contains binary data.

## 6.2 Modifying the Liberator configuration file

Edit the Liberator's configuration file *etc/rttpd.conf* to set up the configuration items that will allow it to work with KeyMaster. These configuration items are:

Liberator configuration item	Description	Required?
<b>signature-validtime</b>	Length of time in seconds for which a user credentials token is valid. This time out applies to any user credentials token that does not have a specific <b>timeout</b> configuration item defined for it (see below).  The default value of <b>signature-validtime</b> is 600 seconds (10 minutes)	No
<b>add-sigkey</b>	Begins a signature key definition group.	Yes
<b>key-id</b>	Identifies this signature key definition group. (Corresponds to a <i>sigkey-id</i> attribute in the XMLauth <i>users.xml</i> configuration file.)	Yes
<b>timeout</b>	Length of time in seconds for which a user credentials token is valid. This overrides the <b>signature-validtime</b> configuration item.	Yes
<b>keyfile</b>	The filename and path of the public key file, in DER (binary) format.	Yes
<b>end-sigkey</b>	Ends the signature key definition group.	Yes
<b>signature-hashsize</b>	The size in buckets of the hash table for storing signature keys.  This configuration item can be changed to tune the Liberator's performance when authorizing users; set it to twice the number of user credentials tokens that are likely to be created within the configured time out period (as defined by the configuration items <b>signature-validtime</b> and <b>timeout</b> ).	No Default = 8192 buckets

Liberator configuration item	Description	Required?
hashing-algorithm	<p>The algorithm to use for validating the digital signature in user credentials tokens provided by KeyMaster.</p> <p>If you are deploying Java-based KeyMaster and KeyMaster has been configured to generate tokens using the SHA256withRSA algorithm (see the <i>web.xml</i> parameter <code>encrypting.generator.signature.algorithm</code> in <a href="#">web.xml parameters</a> [73]), change this configuration item to <code>sha256</code>.</p> <p>If you are deploying a Signature Generator developed using KeyMaster.NET, set this configuration item to the particular algorithm that you have built into the Signature Generator. In KeyMaster.NET, this algorithm is called the "hashing algorithm" – see the <b>KeyMasterHashingAlgorithm</b> enumeration in the <b>KeyMaster.NET API Reference</b>. The list of valid algorithm names is defined in the reference entry for <b>hashing-algorithm</b> in the <b>Liberator Administration Guide</b>.</p> <p><b>Note:</b> The <b>hashing-algorithm</b> configuration parameter is only available in Liberator versions 4.5.7 and above. Earlier versions of Liberator use the MD5withRSA digital signature algorithm, which is compatible with Java-based KeyMaster's default setting.</p>	<p>No Default = md5</p>

The most important configuration item is `add-sigkey` and its children. The `add-sigkey ... end-sigkey` configuration item group defines a public key. Liberator uses this key when it authenticates a user by verifying the digital signature in the user credentials token.

#### Example of `add-sigkey`:

```
## AUTH #####
#
#
auth-module          xmlauth

add-sigkey
  key-id              testkey
  timeout              300
  keyfile              %r/etc/publickey.der
  hashing-algorithm    sha256
end-sigkey
...
```

- ◆ The **key-id** configuration item (`testkey` in this example) identifies this signature key definition group. It is referred to in a `sigkey-id` attribute in the `users.xml` configuration file of XMLauth – see [Modifying the users.xml authorization file](#) [29].

**Note:** The value of **key-id** is *not* related to the key identifier specified in the command that generates the encryption keys (see [Generating the Required Keys](#) [10]).

- ◆ The **timeout** value has been set to 600 seconds in the example. This means that, when a user credentials token has been created, the Liberator will consider it to be invalid after 10 minutes. An end user must therefore connect to the Liberator within 10 minutes of KeyMaster granting the token; after this time the Liberator will reject attempts to log in using the token.
- ◆ The **keyfile** configuration item must point to the DER key file that was generated using KeyMaster (see [Making the public key file available to Liberator](#)<sup>[25]</sup>). The %r in the file path means the root directory of the Liberator installation.



## 6.3 Modifying the users.xml authorization file for XMLAuth

If the Caplin Liberator is configured to use the XMLAuth user authentication module, then you will need to configure in the *users.xml* file all the users who are authorized to access the Liberator via KeyMaster authentication.

The *users.xml* file is located in the Liberator's */etc* directory. It contains a `<USER>` tag for each authorized user. This file is usually created by a script which interfaces with an existing permissions system (for example DACS).

For each user who is to be authenticated with KeyMaster, add the following attributes to their `<USER>` tag:

XMLAuth <code>&lt;USER&gt;</code> tag attribute	Description
sigkey-id	Identifies the public encryption key that is to be used when verifying a user credentials token for this user.  This attribute must match a <b>key-id</b> configuration item within a signature key definition group ( <b>add-sigkey</b> ) in the Liberator configuration file <i>etc/rttpd.conf</i> . See <a href="#">Modifying the Liberator configuration file</a> <sup>26</sup> .
sigcheck	Must be set to "TRUE".

**Example of the *users.xml* file:**

```
<ET_USERS>
  <USER name="admin" pass="admin" logons="2">
    <PERM subject="*" action="grant" />
  </USER>
  <USER name="davids" logons="10"
    sigkey-id="testkey" sigcheck="TRUE">
    <PERM subject="*" action="grant" />
    <PERM subject="/DEMO/*" action="grant" />
  </USER>
</ET_USERS>
```

In this example the user 'davids' (name = "davids") is configured to log in to Liberator using KeyMaster authentication. The `sigkey-id` attribute "testkey" refers to the **keyid** configuration item called "testkey" in the example Liberator configuration file – see [Modifying the Liberator configuration file](#) <sup>26</sup>). Therefore, when a client application tries to log user 'davids' in to the Liberator, his user credentials token will be authenticated using the DER format public encryption key in *publickey.der* in the Liberator's *etc/* directory.

The entry for user 'admin' has no `sigkey-id` and `sigcheck`, attributes so 'admin' is configured to log in to the Liberator directly.

**Note:** When defining a Liberator user who is to be authenticated using KeyMaster, do not assign the user a password – omit the `pass` attribute option from the `<USER>` tag in *users.xml*.

For more information on configuring the *users.xml* file see the **XML Auth Administration Guide** .

## 6.4 Modifying the *cfgauth.conf* authorization file

If the Caplin Liberator is configured to use the *cfgauth* user authentication module, then you will need to configure in the *cfgauth.conf* file all the users who are authorized to access the Liberator via KeyMaster authentication.

The *cfgauth.conf* file is located in the Liberator's */etc* directory.

It contains an *add-user* entry for each authorized user. For each user who is to be authenticated with KeyMaster, add the following options to their *add-user* entry:

cfgauth add-user option	Description
<b>sigcheck</b>	Must be set to "TRUE".
<b>siguser</b>	Identifies the public encryption key that is to be used when verifying a user credentials token for this user.  This attribute must match a <b>key-id</b> configuration item within a signature key definition group ( <b>add-sigkey</b> ) in the Liberator configuration file <i>etc/rttpd.conf</i> . See <a href="#">Modifying the Liberator configuration file</a> <sup>26</sup> .

Example of an *add-user* entry in the *cfgauth.conf* file:

```
add-user
  username davids
  read 0 20 21 22
  licenses 2
  sigcheck TRUE
  siguser testkey
end-user
```

In this example the user 'davids' (*username davids*) is configured to log in to Liberator using KeyMaster authentication. The **siguser** option *testkey* refers to the **key-id** configuration item called "testkey" in the example Liberator configuration file – see [Modifying the Liberator configuration file](#) <sup>26</sup>. Therefore, when a client application tries to log user 'davids' on to the Liberator, his user credentials token will be authenticated using the DER format public encryption key in *publickey.der* in the Liberator's *etc/* directory.

**Note:** When defining a Liberator user who is to be authenticated using KeyMaster, do not assign the user a password – omit the password option from the **add-user** entry of *cfgauth.conf*.

For more information on configuring the *cfgauth.conf* file see the **Liberator Administration Guide**.

## 6.5 Configuring a Liberator that uses javaauth authentication

Your Liberator may use authentication that has been developed using the javaauth SDK (see the **JavaAuth API Reference**). If the authentication uses KeyMaster to provide single sign-on capability, then bespoke code will have been written to integrate KeyMaster with the javaauth implementation. Any configuration required to make Liberator work with KeyMaster will be specific to this implementation. Consult your system developers for information on how to set up the configuration.

## 7 Testing Java-based KeyMaster with Liberator

This section and its subsections do not apply to KeyMaster.NET.

This section explains how to test that the KeyMaster installation, web application server, and Liberator, all work together properly.

The test involves configuring some test files and then launching a web page. The web page communicates with KeyMaster and Liberator to authenticate a user.

First check that you have set up KeyMaster according to the steps described in [Installing KeyMaster](#)<sup>[8]</sup>. In particular make sure that:

- You have created the encryption keys – see [Generating the Required Keys](#)<sup>[10]</sup>.
- You have set up the Liberator to accept and validate User Credentials Tokens created by KeyMaster – see [Setting up Liberator to work with KeyMaster](#)<sup>[25]</sup>.
- You have deployed the *KeyMaster.war* file – see [Deploying KeyMaster](#)<sup>[13]</sup>.
- You have edited the *web.xml* file – see [Modifying the web.xml configuration file](#)<sup>[17]</sup>.

### 7.1 Configuring the test files

The KeyMaster test uses a Real Time Scripting Layer (RTSL) test page, supplied with the installation kit. The RTSL page has two parts, one HTML page called *test.html*, and an accompanying javascript file called *keymaster-config.js*. You need to edit both of these files before running the test.

The files are located in the *keymaster* directory of the web applications area in your web application server:

- ◆ If you are using a Tomcat server the files are in *\$SERVER\_HOME/webapps/keymaster/*
- ◆ If you are using a JBoss server the files are in *\$SERVER\_HOME/server/default/deploy/keymaster.war/*
- ◆ If you are using a WebLogic server the files are in the *keymaster.war* directory of the KeyMaster deployment area (see [Deployment on a BEA WebLogic server](#)<sup>[15]</sup>).

#### Editing the KeyMaster-config.js file

Edit the *keymaster-config.js* file so that it points to the URL of the Caplin Liberator and specifies a valid user name. The following lines must be changed:

```
var l_sLiberatorUrl = "http://caplin.liberator.address:portNo"  
var l_sUser = "validUser";
```

- Change the string assigned to the variable *l\_sLiberatorUrl* to the URL of your Liberator.

**Note 1:** If the Liberator is on a different machine to the web application server, then the Liberator URL must be a fully qualified domain name and port. The web application server and Liberator must share a common domain. For example, an application server at `myserver.example.com` and a Liberator at `myliberator.example.com` share the domain, `example.com`. See [Protocol and domain compatibility](#) <sup>[60]</sup>.

**Example of Liberator URL:**

```
var l_sLiberatorUrl = "http://myliberator.abc.com:8127"
```

- Change the string assigned to the variable `l_sUser` to be the name of a valid user that you previously specified in the Liberator's `users.xml` file (see [Modifying the users.xml authorization file](#) <sup>[29]</sup>).

**Example of Liberator user name:**

```
var l_sUser = "davids";
```

## Editing the test.html file

Edit the `test.html` file to point to the Liberator's Stream Link for Browsers (SL4B) page:

```
<SCRIPT id="sl4b"
  language="JavaScript"
  src="http://caplin.liberator.address:portNo/sl4b/index.js"
  rttpprovider="javascript"
  credentialsprovider="keymaster"
  configurationfile="keymaster-config.js">
</SCRIPT>
```

The `src` attribute `"http://caplin.liberator.address:portNo/sl4b/index.js"` defines the URL that points to the StreamLink for Browsers source page (`index.js` in the Liberator directory `htdocs/sl4b`).

- Change the first part of the URL (`http://caplin.liberator.address:portNo/`) to the URL of your Liberator; for example: `http://myliberator.abc.com:8127/`

```
src="http://myliberator.abc.com:8127/sl4b/index.js"
```

**Note 2:** If the Liberator is on a different machine to the web application server, then the Liberator URL must be a fully qualified domain name and port. The web application server and Liberator must share a common domain. For example, an application server at `myserver.example.com` and a Liberator at `myliberator.example.com` share the domain, `example.com`. See [Protocol and domain compatibility](#) <sup>[60]</sup>.

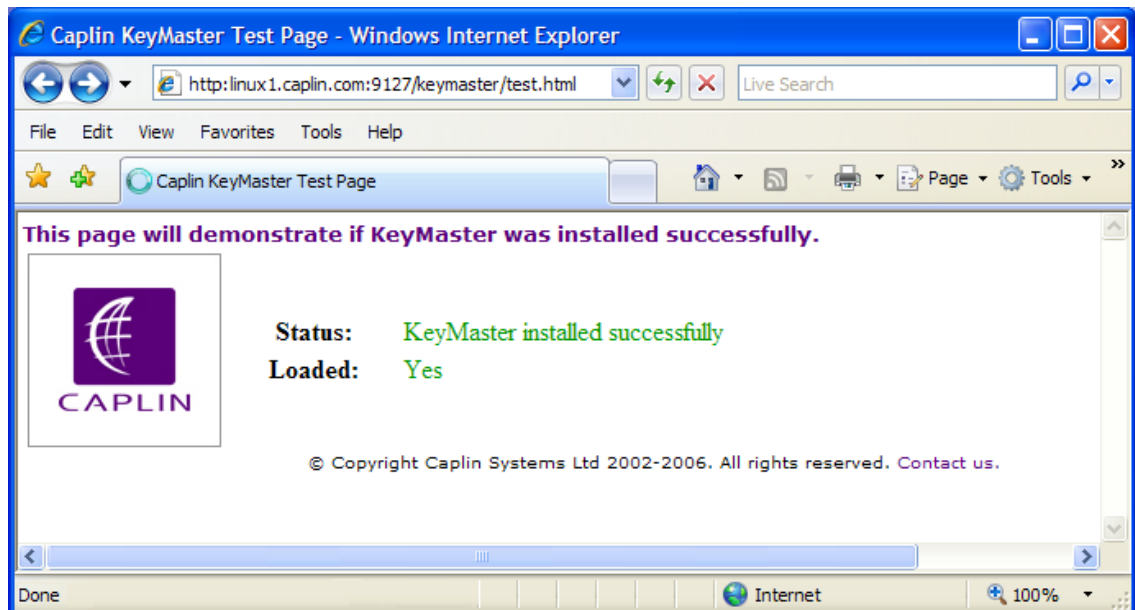
**Note 3:** Change the URL in place. Do not comment out the existing URL line and add the replacement underneath it, because putting HTML comment tags (`<!-- ... -->`) inside the `<SCRIPT>` tag may cause the JavaScript to execute incorrectly and the test will then fail.

## 7.2 Launching the test page

- First make sure that your application server and Liberator are both running.
- Now launch the test page:
  1. Start up a web browser.
  2. Enter the address of your web application server and (if required) its port number, followed by:  
`/KeyMaster/test.html`  
For example:  
`http://myserver.abc.com:9127/keymaster/test.html`

**Note:** If the Liberator is on a different machine to the web application server, then the application server address must be a fully qualified domain name. If the address is not a fully qualified domain name (for example it is an IP address in dot notation), then the test page will hang.

The test page contacts the KeyMaster Signature Generator and gets a user credentials token to access the Liberator. It then attempts to log in to the Liberator using this token. If the Liberator deems that the user is valid, and the token is also valid, then the test page will display a success message, as shown below.



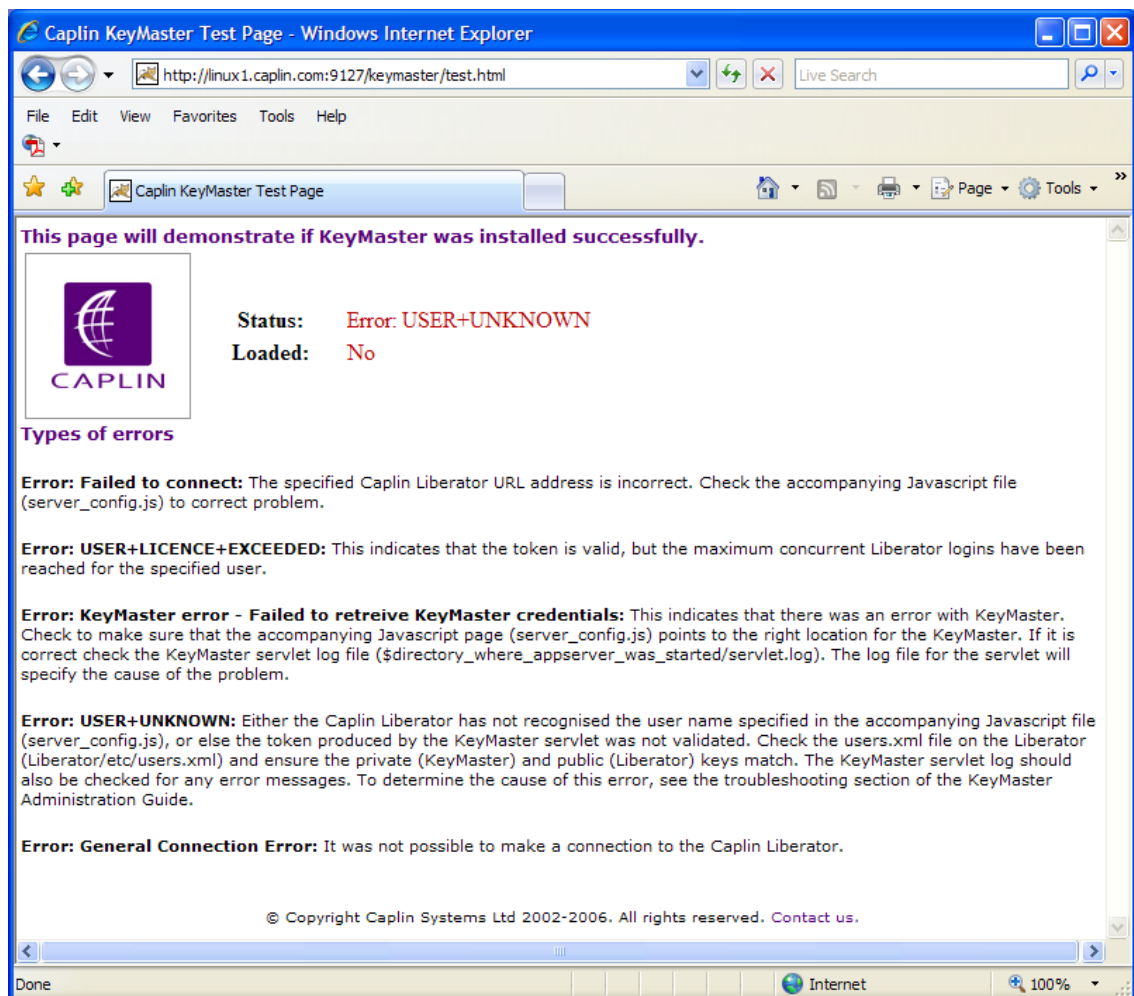
### Successful installation

**Tip:** If you are using Internet Explorer 7 and you wish to look at the Liberator's status page while you run the test, make sure that the status page is loaded in a separate copy of the browser. If you display both the KeyMaster test page and the Liberator status page as tabs within a single copy of the browser, the test will fail with a General Connection Error (see below).

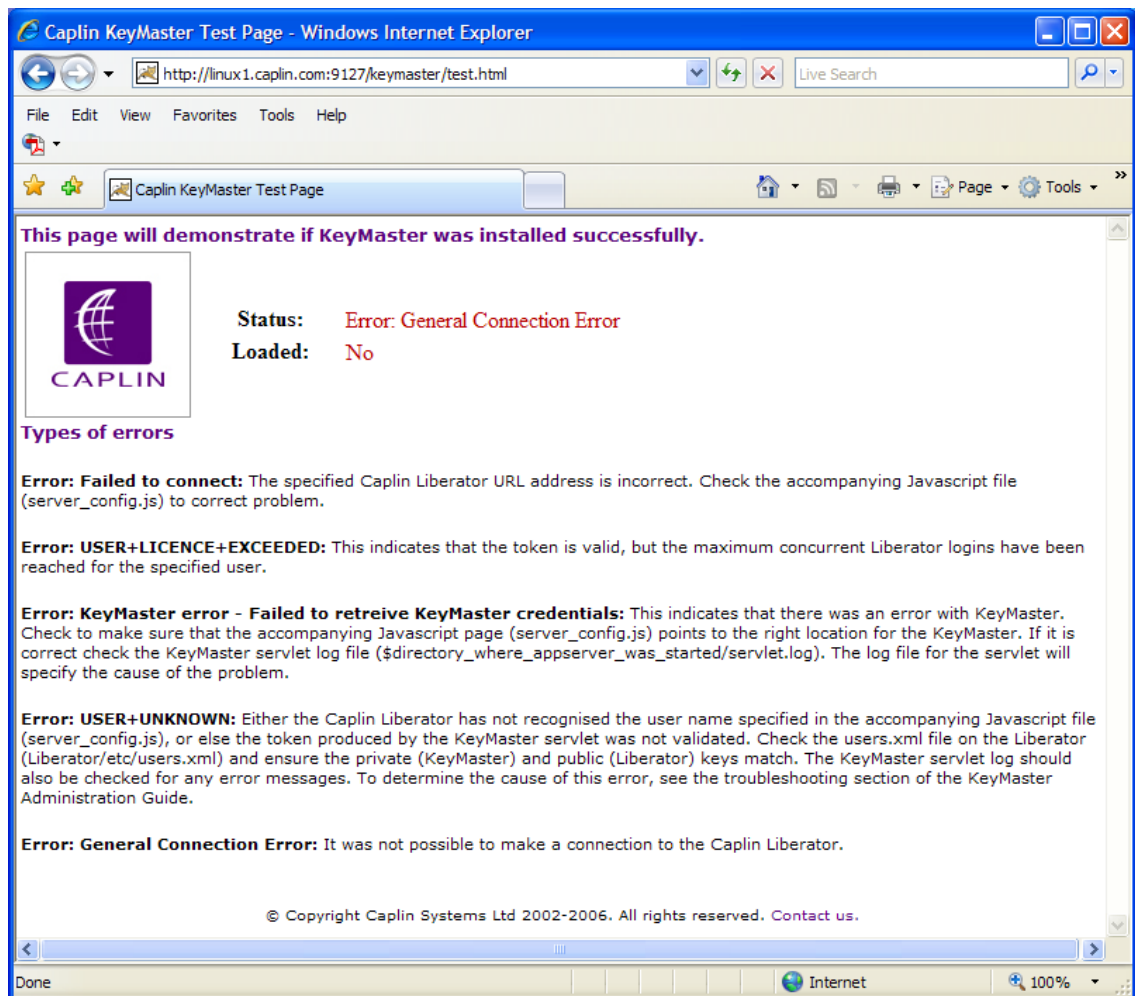
## Test errors

If the Liberator does not recognize the user, or the user credentials token is not valid, then the status line on the test page goes red and the page displays an error message.

Two typical error pages are shown below.



Example error page (1)



Example error page (2)

The Liberator login may also fail if the Liberator already has the maximum number of concurrent logins for the user. In this case the error message is

USER+LICENCE+EXCEEDED

## Determining the cause of an error

The page that reports errors lists the possible errors that it can detect, and suggests reasons for each error and how to investigate it further (see the example error pages above). If an error is reported, check the Liberator's event log file (*var/event-rtttd.log*) to see if there are problems with the user trying to log in. See the [Liberator log file messages](#)<sup>[55]</sup>.



## 8 Testing KeyMaster.NET with Liberator

This section applies only to KeyMaster.NET.

To test your .NET-based KeyMaster Signature Generator with Liberator ensure you have:

- ◆ Generated the encryption key pair as described in the “Generating Keys” section of the **KeyMaster.NET API Reference**.
- ◆ Set up the Liberator to accept and validate user credentials tokens created by KeyMaster – see [Setting up Liberator to work with KeyMaster](#)<sup>25</sup>.
- ◆ Deployed your Signature Generator to an IIS web server.

You can then use any KeyMaster-enabled StreamLink application to log in to Liberator using tokens generated by your .NET-based Signature Generator.

## 9 Making KeyMaster production ready

This section applies to Java-based KeyMaster and to KeyMaster.NET.

The instructions in this guide describe how to set up KeyMaster so that it can be tested easily. *However, in this state KeyMaster is not secure, and it should not be used in a production environment.*

### KeyMaster.NET

To make KeyMaster.NET production ready:

- You must configure the web application server to which you have deployed KeyMaster, so that access to the ASP.NET pages requires authentication.

### Java-based KeyMaster

To make Java-based KeyMaster production ready:

- You must configure the web application server to which you have deployed KeyMaster, so that access to the KeyMaster servlets requires authentication.
- For Java-based KeyMaster, you must ensure that the private key file defined in the *web.xml* parameter `encrypting.generator.private.key.store.filename` is only accessible to persons and processes trusted to create users able to log into the Liberator.
- In Java-based KeyMaster, the default setting of the *web.xml* parameter `http.remote.user` causes the KeyMaster Signature Generator to obtain the user name from an HTTP request parameter sent by the application. This will not be secure if the client can access the user name, so, depending on the configuration of your single sign-on system, you may need to change this setting so the servlet instead obtains the user name from the `REMOTE_USER` attribute of the HTTP header.

For more information see [Adding the user name to the user credentials token](#)<sup>[59]</sup>.

- **You should configure the software to use a more secure digital signature algorithm, such as SHA256withRSA.** The default algorithm, MD5withRSA is now considered to be insecure.

See [Configuring the SHA256 signature algorithm](#)<sup>[39]</sup>.

(MD5withRSA has been retained for backward compatibility with previous versions of KeyMaster.)

- If you intend to use Java-based KeyMaster with a hardware Key Store, see [Integrating KeyMaster with a hardware Key Store](#)<sup>[41]</sup>.

**Tip:** The correct way to ensure secure access to the KeyMaster servlets will vary depending on your web application server. Consult the documentation that came with the server for further information.

For Java-based KeyMaster, also see:

- The definitions of [encrypting.generator.private.key.store.filename](#)<sup>[78]</sup> and [http.remote.user](#)<sup>[83]</sup> in the [web.xml parameters](#)<sup>[73]</sup> section of the [web.xml configuration reference](#)<sup>[65]</sup>.

## 9.1 Configuring the SHA256 signature algorithm

**Note:** **MD5 limitations:** Since KeyMaster was first released, the cryptographic community have found that the MD5 algorithm can produce hash collisions. This potentially compromises the algorithm.

Caplin has retained MD5withRSA as the default digital signature algorithm for backward compatibility with previous versions of KeyMaster. *However, customers installing KeyMaster for the first time are recommended to use the more secure **SHA256** algorithm.*

To configure the **SHA256** signature algorithm:

1. It is recommended that you use the default SUN [JCE](#) <sup>[90]</sup> provider, **SunRsaSign**:

In the *web.xml* configuration file, remove the entries  
encrypting.generator.security.provider.class.name  
and encrypting.generator.security.provider.name:

```
<param-name>encrypting.generator.security.provider.class.name</param-name>
<param-value>org.bouncycastle.jce.provider.BouncyCastleProvider</param-value>

<param-name>encrypting.generator.security.provider.name</param-name>
<param-value>BC</param-value>
```

This causes the servlet to revert to using **SunRsaSign**.

2. In the *web.xml* configuration file set the entry encrypting.generator.signature.algorithm to SHA256withRSA:

```
<init-param>
  <param-name>encrypting.generator.signature.algorithm</param-name>
  <param-value>SHA256withRSA</param-value>
  <description>
    Optional parameter to set the algorithm used for signatures.
    Will default to MD5withRSA if this parameter is not present.
  </description>
</init-param>
```

**Tip:** If you want to continue using the Bouncy Castle JCE provider (see [Open Source Software](#) <sup>[4b]</sup>), and are confident that the SHA256 implementation it provides will work correctly in your environment:  
a) Omit step 1.  
b) At step 2 enter the correct name in <param-value> (it may not be SHA256withRSA).

3. Modify the **add-sigkey** item in the Liberator's configuration file *etc/rtpd.conf* to specify **sha256** as the hashing algorithm:

```
## AUTH #####  
##  
auth-module xmlauth  
  add-sigkey  
    key-id          testkey  
    timeout         300  
    keyfile         %r/etc/publickey.der  
    hashing-algorithm sha256  
end-sigkey
```

**Note:** The **hashing-algorithm** configuration parameter is only available in Liberator versions 4.5.7 and above.  
The only supported values are **md5** and **sha256**.  
Earlier versions of Liberator use the **MD5withRSA** digital signature algorithm, which is compatible with KeyMaster's default setting.

## 10 Integrating KeyMaster with a hardware Key Store

This section and its subsections do not apply to KeyMaster.NET.

KeyMaster generates tokens using a private key, which is normally saved to disk as a file. However, for added security, you can store private keys in a dedicated secure hardware module (a “**Key Store**”) instead of on disk.

In this case, JCE (Java Cryptography Extension) can be used to retrieve keys directly from the Key Store. JCE is a Java extension that allows classes to register as cryptography providers. KeyMaster can retrieve keys from any Key Store that implements the JCE provider interface.

**Tip:** In the sections that follow, the secure hardware module is referred to as the **Key Store** (space between the two words). Note that JCE also has a class called **KeyStore** (no space between the two words).

The following sections explain how to set up KeyMaster so that it can use a Key Store. In summary, having first installed, deployed, and tested KeyMaster without using the Key Store, you then:

- Generate a new set of keys and a certificate, using OpenSSL.
- Import the private key file and certificate into the Key Store.
- Verify the key import operation.
- Install the required libraries.
- Edit the *web.xml* configuration file so that the KeyMaster Signature Generator knows how to retrieve the private encryption key from the Key Store.
- Test that KeyMaster works with the Key Store.
- Configure the Liberator to use the new public key.

### 10.1 Key Store prerequisites and assumptions

Before continuing, ensure that you have successfully installed and deployed KeyMaster by following the instructions in the sections:

- ◆ [Installing KeyMaster](#) <sup>8</sup>
- ◆ [Deploying KeyMaster](#) <sup>13</sup>
- ◆ [Setting up Liberator to work with KeyMaster](#) <sup>25</sup>
- ◆ [Testing KeyMaster with Liberator](#) <sup>32</sup>

The instructions in the following sections assume that the private key file is called *privatekey.der*, the public key file is called *publickey.der*, and the certificate file is called *cert.crt*.

The example values given for configuration options are the correct values for integrating KeyMaster with an *nCipher* Hardware Security Module, which is one of the commercial products available for the secure storage of keys. You will need to change these values accordingly if you are using a different security module product.

**Note:** **Disclaimer:** The references to *nCipher* and *thawte™* in this document do not imply any endorsement by Caplin Systems Ltd of any products or services supplied by these organizations.

*nCipher* (<http://www.ncipher.com>) is just one of a number of suppliers of hardware security modules.

*thawte* (<http://www.thawte.com>) is just one of a number of organizations that provide certificate signing services.

## 10.2 Generating keys using OpenSSL

This section and its subsections do not apply to KeyMaster.NET.  
For information on how to generate key files in KeyMaster.NET,  
see the **KeyMaster.NET API Reference**.

KeyMaster uses three key files to produce tokens:

- ◆ An RSA private key for signing text.
- ◆ A matching RSA public key which is deployed to Liberator, allowing it to verify tokens.
- ◆ A certificate for the private key, which enables it to be added to a Java **KeyStore** class.

To integrate KeyMaster with your hardware Key Store, you must first generate the following key files using OpenSSL:

- ◆ An RSA private key in DER format (used to sign user credentials tokens).
- ◆ A matching RSA public key, also in DER format.
- ◆ An X.509 certificate for the private key, in CRT format.

The following sections explain how to generate these files using the OpenSSL library. OpenSSL (<http://www.openssl.org>) is open source software and can be freely downloaded.

**Note:** The files generated by the Caplin KeyGenerator class that you generated when installing KeyMaster (see [Generating the required keys](#)<sup>[10]</sup>) are not the right format for use with a hardware Key Store. You must use OpenSSL to generate a new set of keys.

**Tip:** Documentation detailing how to install OpenSSL as a command line tool is available on the OpenSSL web site at <http://www.openssl.org/docs/>.

To generate keys for your hardware Key Store using OpenSSL:

- Generate a private key.
- Generate the public key from the private key.
- Generate a certificate request from the private key.
- Obtain a signed certificate.
- Convert the private key to DER format.

The instructions in the following sections show the OpenSSL commands needed to perform these steps.

## Generating a private key

- Generate the private key file in PEM format:

```
openssl genrsa -out privatekey.pem 2048
```

This produces an unencrypted file called *privatekey.pem*, which is the RSA private key.

## Generating the public key

- Generate the public key from the private key:

```
openssl rsa -in privatekey.pem -pubout -outform DER -out publickey.der
```

This produces a file called *publickey.der*, which is the public key that Liberator will use to verify KeyMaster tokens.

**Tip:** Public encryption keys can be distributed freely without any security risk, so *publickey.der* does not need to be stored in the hardware security module.

## Generating the certificate request

To store a private key in a Java **KeyStore** class it must be accompanied by a certificate. The first step towards getting a certificate is to generate a certificate signing request (CSR) from the private key:

- Generate the certificate signing request:

```
openssl req -new -key privatekey.pem -out certrequest.csr
```

You will then be prompted to enter information to be encoded in the certificate, such as country, company name, and organizational unit. When you have entered all the information, a certificate request file called *certrequest.csr* is produced.

## Obtaining a signed certificate

To get a properly signed certificate, you can submit the certificate request (see [Generating the certificate request](#)<sup>43</sup>) to a recognized Certificate Authority such as thawte (<http://www.thawte.com>). However it is also possible to self-sign your CSR, which produces a self-signed certificate – this will be adequate for KeyMaster, because KeyMaster tokens are not public facing.

- To produce a self-signed certificate:

```
openssl x509 -req -in certrequest.csr -signkey privatekey.pem -out cert.crt
```

This produces a file called *cert.crt*, which is the signed certificate. The *certrequest.csr* file is no longer needed and should be securely deleted (for example, through a software shredder).

## Converting the private key to DER format

The private key generated through OpenSSL (see [Generating a private key](#)<sup>[43]</sup>) is a PEM format file, which is human readable. However JCE only accepts keys in DER format.

- To convert the key to DER format:

```
openssl pkcs8 -topk8 -inform PEM -outform DER -in private.pem -out  
private.der -nocrypt
```

This produces a file called *privatekey.der*, which is the reformatted private key. The *privatekey.pem* file is no longer needed and should be securely deleted (for example, through a software shredder).

## 10.3 Importing the private key file and certificate into the Key Store

- Import the newly generated private key into the Key Store.

Caplin provides a tool called **Key Importer** for this purpose. Key Importer reads a properties file to obtain the information needed to import the key. There is an example *keyimporter.props* file in *\$SKM\_INSTALL\_DIR/examples*. Alternatively you can create your own version of this file; see the [keyimporter.props configuration reference](#)<sup>[62]</sup>.

**Example keyimporter.props:**

```
key.importer.security.provider.class.name=com.ncipher.provider.km.nCipherKM  
key.importer.keystore.type=ncipher.sworld  
key.importer.keystore.provider.name=nCipherKM  
key.importer.keystore.location=/opt/keystore.dat  
key.importer.private.key.location=/opt/privatekey.der  
key.importer.private.key.alias=privatekey  
key.importer.certificate.location=/opt/cert.crt  
key.importer.certificate.alias=certificate  
  
key.importer.keystore.passphrase=keystorepassphrase  
key.importer.key.passphrase=keypassphrase
```

The last two properties are optional:

- ◆ `key.importer.keystore.passphrase` can be omitted if the Key Store is not protected by a passphrase.
- ◆ `key.importer.key.passphrase` can be omitted if you do not want the key that you are importing to be protected with a passphrase.

The format of the command to run Key Importer is:

```
java -classpath lib/keymaster.jar: <JCE-provider-classpath>  
com.caplin.keymaster.keyimporter.KeyImporter <key-importer-properties-  
file>
```

where:

<JCE-provider-classpath> is the Java classpath of the JCE provider class.

The JCE provider class is normally provided by the supplier of the Key Store hardware, and is the class referenced by the `key.importer.security.provider.class.name` property in *keyimporter.props*. <JCE-provider-classpath> can, of course, be the name of a JAR file containing the required class.



<key-importer-properties-file> is the path name of the properties file containing the information needed to import the key.

**Example command on Linux, Sun Solaris, or Windows:**

```
java -classpath lib/keymaster.jar:/opt/nfast/java/classes/nCipherKM.jar  
com.caplin.keymaster.keyimporter.KeyImporter examples/keyimporter.props
```

In this example the <JCE-provider-classpath> is the JAR file path /opt/nfast/java/classes/nCipherKM.jar

Key Importer reads in the private key file and adds it to the hardware Key Store. Key Importer produces the following output if it is successful.

```
KeyImportVerifier started  
Adding security provider with class name=com.ncipher.provider.km.nCipherKM ... done  
Getting KeyStore instance with type=ncipher.sworld, provider=nCipherKM ... done  
Loading empty KeyStore with passphrase=null ... done  
Checking private key filename ...done  
Reading private key file from location=/opt/privatekey.der ... done  
Creating PKCS8 encoded key spec from file ... done  
Generating PrivateKey object using KeyFactory with algorithm=RSA ... done  
Reading certificate file from location=/opt/cert.crt ... done  
Generating Certificate object using CertificateFactory with type=X.509 ... done  
Adding certificate to KeyStore with alias=certificate ... done  
Adding private key to KeyStore with alias=privatekey ... done  
Writing KeyStore file to location=/opt/keystore.dat ... done  
Key import successful.
```

If the hardware module provides a way to list the stored keys, then you should now be able to see new entries in the list. For example, *nCipher* provides a GUI application called KeySafe which can be used to list the keys contained within the hardware module.

**Note:** The Key Importer generates a file that acts as a reference to the hardware Key Store. This file will be required in subsequent steps.  
In this example, the file was written to */opt/keystore.dat*; that is, the location defined by the *key.importer.keystore.location* property in *keyimporter.props*.

**Tip:** Do not delete *keyimporter.props* because you will need to refer to it again when editing *web.xml* (see [Modifying the web.xml file for Key Store access](#)<sup>[47]</sup>).

**Tip:** The source code of the Key Importer tool is provided in *\$KM\_INSTALL\_DIR/examples/keyimporter*.

Also see the [keyimporter.props configuration reference](#)<sup>[62]</sup> section.

## 10.4 Verifying the key import operation

- Verify that the import operation detailed in [Importing the private key file and certificate into the Key Store](#)<sup>[44]</sup> was successful.

Caplin provides a tool called **Key Import Verifier** for this purpose. This tool works by attempting to load the Key Store and retrieve the key from it.

The format of the command to run Key Import Verifier is:

```
java -classpath lib/keymaster.jar:<JCE-provider-classpath>  
com.caplin.keymaster.keyimporter.KeyImportVerifier <key-importer-  
properties-file>
```

where:

<JCE-provider-classpath> is the Java classpath of the JCE provider class.

The JCE provider class is normally provided by the supplier of the Key Store hardware, and is the class referenced by the `key.importer.security.provider.class.name` property in *keyimporter.props*. <JCE-provider-classpath> can, of course, be the name of a JAR file containing the required class.

<key-importer-properties-file> is the path name of the properties file containing the information needed to import the key.

**Example command on Linux, Sun Solaris, or Windows:**

```
java -classpath lib/keymaster.jar;/opt/nfast/java/classes/nCipherKM.jar  
com.caplin.keymaster.keyimporter.KeyImportVerifier examples/keyimporter.  
props
```

In this example the <JCE-provider-classpath> is the JAR file path `/opt/nfast/java/classes/nCipherKM.jar`.

<key-importer-properties-file> is the same properties file that was used to import the keys (see [Importing the private key file and certificate into the Key Store](#)<sup>[44]</sup>), and contains all of the information needed to verify that the import operation was successful.

Key Import Verifier attempts to retrieve the key from the Key Store and uses it to sign some text. Key Import Verifier produces the following output if it is successful:

```
Key Import Verifier started  
Adding security provider with class name=com.ncipher.provider.km.nCipherKM ... done  
Getting KeyStore instance with type=ncipher.sworld, provider=nCipherKM ... done  
Loading KeyStore from file=/opt/keystore.dat with passphrase=null ... done  
Retrieving key with id=privatekey and passphrase=null ... done  
Casting retrieved key into a PrivateKey instance ...done  
Generating Signature object with algorithm=SHA256withRSA and provider=nCipherKM ... done  
Initialising Signature object ... done  
Signing some text ... done  
Key verification successful.
```

**Tip:** The source code of the Key Import Verifier tool is provided in `$KM_INSTALL_DIR/examples/keyimporter`.

## 10.5 Installing the required libraries

The JCE provider class must be available to the KeyMaster servlet on your application server. This may be a Java class file, or it may be contained within a JAR file.

For example, if the `key.importer.security.provider.class.name` property in `keyimporter.props` has the value `com.ncipher.provider.km.nCipherKM`, and this class exists in the file `nCipherKM.jar`, then `nCipherKM.jar` must be in the classpath of the KeyMaster servlet.

The correct location to put the class file varies depending on the application server used. For example, if you are using a Tomcat server, the libraries should be placed in the directory `$SERVER_HOME/webapps/keymaster/WEB-INF/lib`.

## 10.6 Modifying the web.xml file for Key Store access

- In the KeyMaster `web.xml` configuration file, edit the parameters applying to the `StandardKeyMaster` servlet, so that the KeyMaster Signature Generator knows how to retrieve the private encryption key from the Key Store:

```
<servlet>
  <servlet-name>StandardKeyMaster</servlet-name>
  <servlet-class>com.caplin.keymaster.servlet.StandardKeyMaster</servlet-class>

  <init-param>
    ...
  </init-param>
  ...
</servlet>
```

**Tip:** The `web.xml` file is located in a directory of your web application server; the precise directory path depends on which application server you are using, but it usually ends in `WEB-INF/`. For example, if KeyMaster is deployed on a Tomcat application server, `web.xml` is in the directory `$SERVER_HOME/webapps/keymaster/WEB-INF/`.

**Tip:** Some of the values that you need to set up in `web.xml` can be found in your `keyimporter.props` file.

Edit `web.xml` as follows:

1. Find the entry called `encrypting.generator.keystore.type`:

```
<init-param>
  <param-name>encrypting.generator.keystore.type</param-name>
  <param-value>standard</param-value>
  <description>
    Optional parameter that accepts the values "standard" or "hardware".
    Will default to "standard" if not present
  </description>
</init-param>
```

Change the content of the `<param-value>` tag from **standard** to **hardware**.

**Example:**

```
<init-param>
  <param-name>encrypting.generator.keystore.type</param-name>
  <param-value>hardware</param-value>
  <description>
    Optional parameter that accepts the values "standard" or "hardware".
    Will default to "standard" if not present
  </description>
</init-param>
```

**Note:** Since `encrypting.generator.keystore.type` is an optional parameter it may not exist in *web.xml*. If this is the case then the parameter must be added with the value `hardware` as shown in the previous example.

2. Find the entry called `encrypting.generator.key.identifier`:

```
<init-param>
  <param-name>encrypting.generator.key.identifier</param-name>
  <param-value>keyidl</param-value>
  <description>Name of the server the token is generated for.</description>
</init-param>
```

Change the content of the `<param-value>` tag to the alias of the private encryption key in the Key Store. This will be the value of the `key.importer.private.key.alias` property in *keyimporter.props*. For example, if the `key.importer.private.key.alias` property was assigned the value **privatekey**, then *web.xml* should be edited like this:

**Example:**

```
<init-param>
  <param-name>encrypting.generator.key.identifier</param-name>
  <param-value>privatekey</param-value>
  <description>Name of the server the token is generated for.</description>
</init-param>
```

3. Remove the `encrypting.generator.private.key.store.filename` property, as it is not used when KeyMaster is interacting with a hardware Key Store:

```
<del>
  <init-param>
    <param-name>encrypting.generator.private.key.store.filename</param-name>
    <param-value>/Caplin/KeyMaster/privatekey.store</param-value>
    <description>File name and location for the private key</description>
  </del>
</del>
```

4. Find the entry called `encrypting.generator.security.provider.class.name`:

```
<init-param>
  <param-name>encrypting.generator.security.provider.class.name</param-name>
  <param-value>org.bouncycastle.jce.provider.BouncyCastleProvider</param-value>
  <description>KeyMaster encrypting class</description>
</init-param>
```

Change the content of the `<param-value>` tag to the name of the [JCE](#)<sup>[90]</sup> provider's Java class used to generate the encrypted portion of the user credentials token. For example, if the new JCE provider class is **com.ncipher.provider.km.nCipherKM**, then *web.xml* should be edited like this:

**Example:**

```
<init-param>
... <param-name>encrypting.generator.security.provider.class.name</param-name>
... <param-value>com.ncipher.provider.km.nCipherKM</param-value>
... <description>KeyMaster encrypting class</description>
</init-param>
```

5. Find the entry called `encrypting.generator.security.provider.name`:

```
<init-param>
... <param-name>encrypting.generator.security.provider.name</param-name>
... <param-value>BC</param-value>
... <description>
  KeyMaster security provider - the name of the JCE provider.
</description>
</init-param>
```

Change the content of the `<param-value>` tag to the name of the [JCE](#)<sup>[90]</sup> provider used to generate the encrypted portion of the user credentials token. For example, if the new JCE provider is **nCipherKM**, then *web.xml* should be edited like this:

**Example:**

```
<init-param>
<param-name>encrypting.generator.security.provider.name</param-name>
... <param-value>nCipherKM</param-value>
... <description>
  KeyMaster security provider - the name of the JCE provider.
</description>
</init-param>
```

6. Add an entry called `encrypting.generator.hardware.keystore.type` where the content of the `<param-value>` tag matches the value entered in *keyimporter.props* for the `key.importer.keystore.type` property. For example, if the `key.importer.keystore.type` property was assigned the value `ncipherkm.sworld`, then the following entry should be added to *web.xml*:

**Example:**

```
<init-param>
  <param-name>encrypting.generator.hardware.keystore.type</param-name>
  <param-value>ncipherkm.sworld</param-value>
</init-param>
```

7. Add an entry called `encrypting.generator.hardware.keystore.keyfile` where the content of the `<param-value>` tag matches the value entered in *keyimporter.props* for the `key.importer.keystore.location` property. This will be the location and filename of the file generated by Key Importer. For example, if the `key.importer.keystore.location` property was assigned the value `/opt/keystore.dat`, then the following entry should be added to *web.xml*:

**Example:**

```
<init-param>
  <param-name>encrypting.generator.hardware.keystore.keyfile</param-name>
  <param-value>/opt/keystore.dat</param-value>
</init-param>
```

8. Find the entry called `encrypting.generator.signature.algorithm`:

```
<init-param>
  <param-name>encrypting.generator.signature.algorithm</param-name>
  <param-value>MD5withRSA</param-value>
  <description>
    Optional parameter to set the algorithm used for signatures.
    Will default to MD5withRSA if this parameter is not present.
  </description>
</init-param>
```

This parameter defines the algorithm used to create digital signatures using the private key, and takes one of the values **MD5withRSA** or **SHA256withRSA**. You will need to change the signature generation algorithm if your JCE provider does not provide the MD5withRSA algorithm (which is the case with *nCipher*, for example).

**Example:**

```
<init-param>
  <param-name>encrypting.generator.signature.algorithm</param-name>
  <param-value>SHA256withRSA</param-value>
  <description>
    Optional parameter to set the algorithm used for signatures.
    Will default to MD5withRSA if this parameter is not present.
  </description>
</init-param>
```

**Note:** **MD5 limitations:** Since KeyMaster was first released, the cryptographic community have found that the MD5 algorithm can produce hash collisions. This potentially compromises the algorithm.

Caplin has retained MD5withRSA as the default digital signature algorithm for backward compatibility with previous versions of KeyMaster. *However, customers installing KeyMaster for the first time are recommended to use the more secure **SHA256** algorithm.*

**Note:** Since `encrypting.generator.signature.algorithm` is an optional parameter it may not exist in `web.xml`. If this is the case, add this parameter with the value **MD5withRSA** or **SHA256withRSA**, as shown in previous example. MD5withRSA and SHA256withRSA are the only algorithms that can be used to generate KeyMaster tokens.

**Tip:** The algorithm you select will also need to be added to the Liberator configuration. See [Configuring Liberator to use a new public key](#)<sup>[52]</sup>.

9. If your Key Store is protected by a passphrase, then add an entry called `encrypting.generator.hardware.keystore.passphrase` where the content of the `<param-value>` tag matches the value entered in `keyimporter.props` for the `key.importer.keystore.passphrase` property. For example, if the `key.importer.keystore.passphrase` property was assigned the value **keystorepassphrase**, then this entry should be added to `web.xml`:

**Example:**

```
<init-param>
  <param-name>encrypting.generator.hardware.keystore.passphrase</param-name>
  <param-value>keystorepassphrase</param-value>
</init-param>
```

If your Key Store is not protected by a passphrase, then you do not need to add anything.

10. If you chose to protect your private encryption key with a passphrase, then add an entry called `encrypting.generator.hardware.key.passphrase` where the content of the `<param-value>` tag matches the value entered in `keyimporter.props` for the `key.importer.key.passphrase` property. For example, if the `key.importer.key.passphrase` property was assigned the value **keypassphrase**, then this entry should be added to `web.xml`:

**Example:**

```
<init-param>
... <param-name>encrypting.generator.hardware.key.passphrase</param-name>
... <param-value>keypassphrase</param-value>
</init-param>
```

If `key.importer.key.passphrase` was not defined in `keyimporter.props`, then the key is not protected by a passphrase and no configuration needs to be added.

**Tip:** For more information on configuring the KeyMaster servlet in the `web.xml` file, see the [web.xml configuration reference](#)<sup>[65]</sup> section.

**Note:** For the configuration changes to take effect, you will need to reload the KeyMaster web application and possibly restart the application server.

## 10.7 Testing KeyMaster works with the Key Store

Before testing that KeyMaster works with the Key Store, make sure you have:

- ◆ Imported the private encryption key into the hardware Key Store (see [Importing the key file and certificate into the Key Store](#)<sup>[44]</sup>).
  - ◆ Made the required JCE provider class available to the KeyMaster Signature Generator (see [Installing the required libraries](#)<sup>[47]</sup>).
  - ◆ Made the appropriate edits to the KeyMaster *web.xml* configuration file (see [Modifying the web.xml file for Key Store access](#)<sup>[47]</sup>).
- To test KeyMaster, follow the instructions in [Testing KeyMaster with the application server](#)<sup>[23]</sup>.

## 10.8 Configuring Liberator to use a new public key

You may need to modify the **add-sigkey** item in the Liberator's configuration file *etc/rtpd.conf*:

- If the filename and/or path of the public key file has been changed, modify the *keyfile* parameter to define the new pathname.
- If the digital signature algorithm has been changed (see [encrypting.generator.signature.algorithm](#)<sup>[50]</sup> in [Modifying the web.xml file for Key Store access](#)<sup>[47]</sup>), add a *hashing-algorithm* parameter to specify the new algorithm:

**Example of edited Liberator configuration item add-sigkey:**

```
## AUTH #####
##
auth-module xmlauth
add-sigkey
    key-id          testkey
    timeout         300
    keyfile        %r/etc/NewPublicKey.der
    hashing-algorithm sha256
end-sigkey
```

**Note:** The **hashing-algorithm** configuration parameter is only available in Liberator versions 4.5.7 and above.  
The only supported values are **md5** and **sha256**.  
Earlier versions of Liberator use the MD5withRSA digital signature algorithm, which is compatible with KeyMaster's default setting (but is now considered insecure).



## 10.9 Testing Liberator works with the new public key

- Restart the Liberator and test it with KeyMaster (see [Testing KeyMaster with Liberator](#)<sup>[32]</sup>).

## 10.10 Tidying up

- Securely delete (using a software shredder for example):
  - The copies of the private key and certificate that are not secured in the Key Store.
  - The old keys and certificates that were generated when you first installed KeyMaster.

## 11 Customizing KeyMaster

This section does not apply to KeyMaster.NET.

The KeyMaster distribution kit contains a standard Java version of the product ("Standard KeyMaster"). This can be configured by modifying various configuration files (see the sections [More about configuring KeyMaster](#)<sup>[58]</sup> and [Configuration reference](#)<sup>[61]</sup>). However, in many instances KeyMaster will need to be integrated into an existing single sign-on system, and this may require some product customization involving modifications and additions to KeyMaster's Java code.

The **KeyMaster Java API Reference** contains the specifications of the public KeyMaster Java classes that can be called and extended to produce customized versions of KeyMaster.

When you customize KeyMaster, you may also need to add or modify certain parameters that are specifically concerned with specifying Java modules. These parameters are defined in the *keygen.props* and *web.xml* files.

- ◆ If you have customized KeyMaster to use a different class for generating public and private encryption keys (the JCE provider's Java class):

Change the value of `key.generator.security.provider.class.name` in *keygen.props*, to the fully qualified name of the new Java class.

Also change the value of `key.generator.security.provider.name` in *keygen.props*.

- ◆ If you have customized KeyMaster to use a different class for generating the encrypted portion of the user credentials token (the JCE provider's Java class):

Change the value of `encrypting.generator.security.provider.class.name` in *web.xml* to the fully qualified name of the new Java class.

Also change the value of `encrypting.generator.security.provider.name` in *web.xml*.

- ◆ If you have customized KeyMaster to use a different class to obtain the user name:

Change the value of the `user.credential.provider` parameter in *web.xml*.

See [Adding the user name to the user credentials token](#)<sup>[59]</sup>.

- ◆ If you have added a new response formatter Java class to KeyMaster:

Add a new `formatter-type-{formatter_name}` parameter in *web.xml*, to define the name of the new class.

A response formatter class formats KeyMaster's response to a request for a user credentials token.

- ◆ If you have implemented a custom version of the KeyMaster Signature Generator:

When you want to test the KeyMaster installation deploying this servlet, you must first edit the *keymaster-config.js* file (see [Configuring the test files](#)<sup>[32]</sup> in [Testing KeyMaster with Liberator](#)<sup>[32]</sup>).

Edit the declaration of the `l_sKeyMasterUrl` variable to point to the URL of the custom KeyMaster servlet.

The following line must be changed:

```
var l_sKeyMasterUrl = "/keymaster/servlet/StandardKeyMaster";
```

**Tip:** For more information about the configuration parameters mentioned above, see the sections on [web.xml configuration reference](#)<sup>[65]</sup> and [keygen.props configuration reference](#)<sup>[61]</sup>.

## 12 Troubleshooting

This section and its subsections apply to Java-based KeyMaster and to KeyMaster.NET.

This section contains additional information on how to ensure that your KeyMaster installation runs correctly.

### 12.1 Synchronizing the servers

Make sure that the clock on the server running the Liberator is synchronized with the clock on the server where KeyMaster Signature Generator is running. If the clocks on these two servers are set to different times, the Liberator may falsely decide that a user credentials token has expired and it is likely to reject all user credentials tokens for this reason.

If the clocks are not correctly synchronized you will see the following message in the Liberator log file:

```
NOTIFY: Signature expired for key_id [key id] - [timestamp] denying login
```

Also see [Liberator log file messages](#) <sup>55</sup>

### 12.2 Liberator log file messages

The following table lists and explains the messages relating to KeyMaster authentication that can appear in the Liberator event log file (*var/event-rtttd.log*).

Log Message	Description
INFO: Token <[token]> is validated for <[key_id]> testkey	The specified user credentials token called [token] has been successfully validated.
NOTIFY: Signature expired for key_id [key id] - [timestamp] denying login	<p>A user credentials token has expired. A token is valid from the time it was created plus the number of seconds specified in the <b>signature-validtime</b> or <b>timeout</b> configuration item in <i>rtttd.conf</i> (see <a href="#">Modifying the Liberator configuration file</a> <sup>26</sup>).</p> <ul style="list-style-type: none"> <li>Make sure that clock on the server running the Liberator is synchronized with the clock on the server where KeyMaster Signature Generator is running.</li> </ul> <p>If the clocks on these two servers are set to different times, the Liberator may falsely decide that a user credentials token has expired (it is likely to reject all user credentials tokens for this reason).</p>
ERROR: Cannot load keyfile <[filename]>	<p>The DER format public key file called [filename], specified in <i>rtttd.conf</i>, is missing, corrupt or in the wrong format.</p> <ul style="list-style-type: none"> <li>Check that the key file configuration is specified correctly in <i>rtttd.conf</i>; look at the <b>key-id</b> configuration item in the <b>add-sigkey</b> item group (see <a href="#">Modifying the Liberator configuration file</a> <sup>26</sup>).</li> </ul>

Log Message	Description
ERROR: Could not find key_id [key id]	When the Auth Module asked for a check on a user credentials token, the key-id was found to be unknown. <ul style="list-style-type: none"> <li>Check that the <b>key-ids</b> match between <i>rtttd.conf</i> and the Auth Module configuration file (for example the <i>users.xml</i> file). Look in <i>rtttd.conf</i> at the <b>key-id</b> configuration item in the <b>add-sigkey</b> item group; in <i>users.xml</i> look at the <code>sigkey-id</code> attribute for each <code>&lt;USER&gt;</code> tag.</li> </ul>
ERROR: Malformed token <[KeyMaster token]> for key_id [key id]	The user credentials token provided to the Liberator is in the wrong format. In Standard KeyMaster it has the following format: <pre>&lt;base64 encoded signature&gt; ~&lt;timestamp&gt;~&lt;sequence number&gt;.</pre>
ERROR: Token verification failed for key_id [key id] <[token]>	The user credentials token failed to verify upon decryption. Either the key used to decrypt the signature does not match the key that KeyMaster used to encrypt it, or the token has been tampered with or corrupted in some way.
ERROR: Malformed timestamp for key_id [key id] <[token]>	The timestamp in the user credentials token is badly formed. It should have the format <code>YYMMDDHHMMSS</code> (for example, <code>20050126122011</code> ).
ERROR: Token [token] has already logged in for key_id [key id]	The user credentials token has already been used; a token can only be used once.
CRITICAL: Could not locate key file <etc/publickey1.der> for add-sigkey/key-id <testkey >	The entry in the <i>rtttd.conf</i> file for the public key could not be mapped to a public DER key file in the specified (or default) directory. Either the <i>rtttd.conf</i> entry is invalid, or the key file is missing. <ul style="list-style-type: none"> <li>Look in <i>rtttd.conf</i> at the <b>keyfile</b> configuration item in the <b>add-sigkey</b> item group. Check that the specified name and directory of the public DER key file match the name and location of the actual file.</li> <li>Check that the key file is actually present in the specified location.</li> </ul> <p>Note: The Liberator will fail to start if this error occurs (the Liberator displays the error message on the screen as well as logging it).</p>
CRITICAL: No keyfile defined for add-sigkey/key-id <testkey >	The entry in the <i>rtttd.conf</i> file for the public key does not specify a key file. <ul style="list-style-type: none"> <li>Look in <i>rtttd.conf</i> at the <b>add-sigkey</b> configuration item group; make sure that there is a <b>keyfile</b> configuration item in this group (see <a href="#">Modifying the Liberator configuration file</a> <sup>[26]</sup>).</li> </ul> <p>Note: The Liberator will fail to start if this error occurs (the Liberator displays the error message on the screen as well as logging it).</p>
CRITICAL: No key-id for an add-sigkey configuration group	The entry in the <i>rtttd.conf</i> file for the public key does not specify a key id. <ul style="list-style-type: none"> <li>Look in <i>rtttd.conf</i> at the <b>add-sigkey</b> configuration item group; make sure that there is a <b>key-id</b> configuration item in this group (see <a href="#">Modifying the Liberator configuration file</a> <sup>[26]</sup>).</li> </ul> <p><b>Note:</b> The Liberator will fail to start if this error occurs (the Liberator displays the error message on the screen as well as logging it).</p>

Log Message	Description
<pre>ERROR: 4012903328: error: 0407006A: rsa routines: RSA_padding_check_PKCS1_type_1: block type is not 01:rsa_pkl.c:100:  ERROR: 4012903328: error: 04067072: rsa routines: RSA_EAY_PUBLIC_DECRYPT: padding check failed: rsa_eay.c:699:  ERROR: Token verification failed for key_id &lt;testkey&gt; &lt;AAAtokenBBB&gt;</pre>	<p>The public / private key token verification has failed.</p> <ul style="list-style-type: none"><li>■ Verify that the private and public keys used in KeyMaster and the Liberator Auth Module are valid.</li><li>■ Check that the <i>publickey.der</i> file referenced in the <i>rtttd.conf</i> <b>add-sigkey</b> configuration item group is not corrupt. This is a binary file which can be corrupted if it is transferred via ftp in ASCII mode.</li></ul>

## 13 More about configuring Keymaster

The following sections contain information about configuring additional features of KeyMaster.

### 13.1 Configuration in web.xml

This section does not apply to KeyMaster.NET.

When deploying Standard Java KeyMaster, you modify the configuration file *web.xml* to specify the location of the private encryption key, and to define the KeyMaster error logging environment (see [Modifying the web.xml configuration file](#)<sup>[17]</sup>). The *web.xml* file can also contain a number of other configuration settings that determine the way Standard KeyMaster behaves, and some configuration settings that you modify when KeyMaster has been customized.

The format of the *web.xml* file is specified in the section on [web.xml configuration reference](#)<sup>[65]</sup>. The configuration items that you can specify in this file are defined in [web.xml parameters](#)<sup>[73]</sup>.

### 13.2 Adding mapping data to the KeyMaster credentials token

This section does not apply to KeyMaster.NET.

This feature lets you add structured data to the KeyMaster token that Liberator uses to validate an end-user's login. Liberator can be configured to use this data in subject mappings.

In the following example, the end-user's trader id is retrieved from a SSO system. Liberator is then configured to use this trader id when it maps the subjects of trade subscriptions.

1. Write a class that implements the **com.caplin.keymaster.servlet.MappingDataProvider** interface, and that has a constructor that does not take any arguments. KeyMaster calls this interface to get the mapping data it adds to the KeyMaster token.

#### Example implementation of MappingDataProvider

```
public class MyTradeIdMappingDataProvider implements MappingDataProvider
{
    Map<String, String> getMappingData(HttpServletRequest httpRequest)
    {
        String traderId = MySSOSystem.getTradeId(httpRequest.getRemoteUser());

        Map<String, String> mappingData = new HashMap<String, String>();
        mappingData.put("trader.id", traderId);
        return mappingData;
    }
}
```

**Note:** The keys in the map can only contain alphanumeric characters, the underscore, and the period (full stop) character.

For further information about the **MappingDataProvider** interface, refer to the **KeyMaster Java API Reference** documentation.

2. Set two *web.xml* parameters that configure the **StandardKeyMaster** servlet to encrypt the trader id in the KeyMaster token.

- Set **encrypting.encode.extra.data** to **enabled**.
- Set **mapping.data.provider** to **MyTradeIdMappingDataProvider**.

#### Example configuration (in *web.xml*)

```
<param-name>encrypting.encode.extra.data</param-name>
<param-value>enabled</param-value>

<param-name>encrypting.encode.extra.data</param-name>
<param-value>MyTradeIdMappingProvider</param-value>
```

3. Configure Liberator to use this data in the subject mapping of a subscription.

#### Example configuration (in *rtttd.conf*)

```
object-map    /PRIVATE/TRADE/%1    /PRIVATE/TRADE/{ trade.id } /%1
```

**Tip:** For further information about subject mapping configuration, refer to the **Liberator Administration Guide** document.

4. The Trading Adapter will now see the trade id in the subject requests it receives from Liberator, and can use this information when it communicates with the trading system.

## 13.3 Adding the user name to the user credentials token

This section does not apply to KeyMaster.NET.

If you deploy Standard Java KeyMaster with the default *web.xml* settings (plus the adjustments defined in [Modifying the web.xml configuration file](#) <sup>[17]</sup>), the generated user credentials tokens will contain just a date-time stamp and a unique sequence number, plus a digital signature of both these items.

For additional security you can include in the token the user name that was specified when the end user first signed on to the system.

There are two *web.xml* parameter settings that control how this is done.

- ◆ If you set the [encrypting.encode.extra.data](#) <sup>[74]</sup> parameter to **enabled**, the KeyMaster Signature Generator will put the user name in the user credentials token and include it in the digital signature.
- ◆ The setting of [http.remote.user](#) <sup>[83]</sup> determines where the user name is obtained from. The default setting of this parameter causes the KeyMaster Signature Generator to obtain the user name from an HTTP request parameter sent by the application (`?username=...`). This will not be secure if the client can access the user name. You can change the setting so that the servlet instead obtains the user name from the `REMOTE_USER` attribute of the HTTP header (assuming the single sign-on system supports transmitting the user name in this way).

**Tip:** Adding the user name to the user credentials token will guard against the token being hijacked and reused by someone other than the person who originally requested the token.

You can also implement a custom class that obtains the user name. You do this by replacing (or extending) the Standard KeyMaster class **com.caplin.keymaster.servlet.UserCredentialsProvider**. For example, the custom class could obtain the user name from a cookie passed between the client web page and the application server. You must change the value of the [user.credential.provider](#) parameter in *web.xml* to point to the new custom class.

For more information on the **UserCredentialsProvider** class see the **Javadoc reference documentation for the Caplin KeyMaster SDK**.

Also see [Making KeyMaster production ready](#)

## 13.4 Protocol and domain compatibility

This section applies to Java-based KeyMaster and to KeyMaster.NET.

When KeyMaster is in use, the client web browser will access both the Liberator (via a StreamLink connection) and KeyMaster (via a standard web connection).

Access to KeyMaster must be through the same protocol, HTTP or HTTPS, as the protocol used by the Liberator.

For example, assume the Liberator is at `myliberator.example.com` and KeyMaster is at `keymaster.example.com`.

If the connection to the Liberator is through **http**: `//myliberator.example.com`, then KeyMaster at must be accessed at **http**: `//keymaster.example.com`. Conversely, if the connection to the Liberator is through **https**: `//myliberator.example.com`, then KeyMaster must be accessed at **https**: `//keymaster.example.com`

If the Liberator is on a different machine to the web application server where KeyMaster resides, then the KeyMaster and Liberator must share a common domain. (More exactly, the StreamLink enabled page that the client application uses to access the Liberator must share a common domain with the Liberator.) For example, an application server at `myserver.example.com` and a Liberator at `myliberator.example.com` share the domain, `example.com`.

In this situation, the common domain must be defined to StreamLink. In StreamLink for Browsers this is done using the `SL4B commondomain` property; for more information see the **StreamLink for Browsers API Reference**.



## 14 Configuration reference

This section and its subsections do not apply to KeyMaster.NET.

This is the reference information for Java KeyMaster's configuration files.

### 14.1 keygen.props configuration reference

This section contains the reference information for the *keygen.props* configuration file. *keygen.props* is a properties file defining the characteristics of the KeyMaster Key Generator servlet. There is an example of this file in *\$KM\_INSTALL\_DIR/examples/*.

#### File format:

The file contains property name/value pairs. Each pair is separated with an '=' character, and each property must be defined on its own line.

#### Example:

```
key.generator.private.key.store.filename=privatekey.store
```

**Note:** *keygen.props* must contain an entry for each of the properties defined in the following table.

#### *keygen.props* properties

<i>keygen.props</i> property name	Example setting	Description
key.generator.private.key.store.filename	privatekey.store	The name and location of the file that the private key will be stored in.
key.generator.public.key.store.filename	publickey.store	The name and location of the file that the public key will be stored in.
key.generator.public.key.der.filename	publickey.der	The name and location of the <a href="#">DER</a> <sup>[89]</sup> formatted public key file. The file name must end in '.der'
key.generator.key.size	1024	The size of the generated key. Both the public and private keys will be this size.
key.generator.security.provider.class.name	org.bouncycastle.jce.provider.BouncyCastleProvider	The fully qualified name of the <a href="#">JCE</a> <sup>[90]</sup> provider's Java class that generates the encryption key pairs.  This class must also be in the Java classpath of the KeyMaster Key Generator servlet (it is usually in a JAR file that is included in the classpath – see <a href="#">Generating the Required Keys</a> <sup>[10]</sup> ). Also see <a href="#">Note 1 below</a> <sup>[62]</sup> .
key.generator.security.provider.name	BC	The name of the provider of the Java class used to generate the encryption key pairs. Also see <a href="#">Note 1 below</a> <sup>[62]</sup> .
key.generator.Level	INFO	Defines the Java logging level for the KeyMaster Key Generator's log file (see the

<i>keygen.props</i> property name	Example setting	Description
		key.generator.FileNameAttribute property). In the example <i>keygen.props</i> file shipped with KeyMaster the logging level is set to ALL (see the example below), which provides a very detailed level of logging for debug purposes.  Also see <a href="#">Note 2</a> and the <a href="#">Tip</a> below.
key.generator.FileNameAttribute	keygen.log	Defines the name and location of the KeyMaster Key Generator's log file. The location of a newly created log file is relative to the user's current directory at the time the Key Generator is run.  Note that each time the Key Generator is run the contents of any existing log file are overwritten; the new log entries are not appended to the file.

**Note 1:** key.generator.security.provider.class.name and key.generator.security.provider.name  
Only change these settings if you have customized KeyMaster to use a different encryption class.

**Note 2:** key.generator.Level  
In a production system it is recommend that the logging level normally be set to SEVERE or WARNING.

**Tip:** key.generator.Level  
The possible logging levels are defined in the standard Java documentation under [java.util.logging.Level](#).

#### Example *keygen.props* file:

```
key.generator.private.key.store.filename=privatekey.store
key.generator.key.size=1024
key.generator.public.key.store.filename=publickey.store
key.generator.public.key.der.filename=publickey.der
key.generator.security.provider.class.name=
    org.bouncycastle.jce.provider.BouncyCastleProvider
key.generator.security.provider.name=BC
key.generator.Level=ALL
key.generator.FileNameAttribute=keygen.log
```

## 14.2 keyimporter.props configuration reference

This section contains the reference information for the *keyimporter.props* configuration file. *keyimporter.props* is a properties file defining the characteristics of the KeyMaster Key Importer tool for importing key files into a *hardware Key Store*. There is an example of this file in *\$KM\_INSTALL\_DIR/examples/*.

#### File format:

The file contains property name/value pairs. Each pair is separated with an '=' character, and each property must be defined on its own line.

**Example:**

```
key.importer.keystore.type=ncipher.sworld
```

**Note:** *keyimporter.props* must contain an entry for each of the properties defined in the following table.

***keyimporter.props* properties**

<b><i>keyimporter.props</i> property name</b>	<b>Example setting</b>	<b>Description</b>
key.importer.certificate.alias	certificate	The identifier to use as a reference to the certificate in the Java <b>KeyStore</b> class of the JCE.
key.importer.certificate.location	/opt/cert.crt	The name and location of the certificate to use when adding the key to the Java <b>KeyStore</b> class of the JCE.
key.importer.key.passphrase	mykeypassphrase	Optional parameter defining a passphrase to access the private key stored in the Key Store.
key.importer.keystore.location	/opt/keystore.dat	Key Importer generates a file that defines how KeyMaster can access the Key Store. This property defines the name and directory path of this file.
key.importer.keystore.provider.name	nCipherKM	The name of the Key Store to which the private key and certificate are to be loaded. This information is required for the Java <b>KeyStore</b> class of the JCE – it is the second argument of the <b>getInstance()</b> method. The value required is normally specified by the supplier of the Key Store hardware.
key.importer.keystore.passphrase	mykeystorepassphrase	Optional parameter defining a passphrase to access the Key Store.
key.importer.keystore.type	ncipher.sworld	The type of Key Store to which the private key and certificate are to be loaded. This information is required for the Java <b>KeyStore</b> class of the JCE – it is the first argument of the <b>getInstance()</b> method. The value required is normally specified by the supplier of the Key Store hardware.
key.importer.private.key.alias	privatekey	The identifier to use as a reference to the key in the Java <b>KeyStore</b> class of the JCE.
key.importer.private.key.location	/opt/privatekey.der	The name and location of the private key to import into the Key Store.
key.importer.security.provider.class.name	com.ncipher.provider.km.nCipherKM	The fully qualified name of the JCE provider's Java class that generates the encryption key pairs. This is normally specified by the supplier of the Key Store hardware.

**Example *keyimporter.props* file:**

```
key.importer.security.provider.class.name=com.ncipher.provider.km.nCipherKM
key.importer.keystore.type=ncipher.sworld
key.importer.keystore.provider.name=nCipherKM
key.importer.keystore.location=/opt/keystore.dat
key.importer.private.key.location=/opt/privatekey.der
key.importer.private.key.alias=privatekey
key.importer.certificate.location=/opt/cert.crt
key.importer.certificate.alias=certificate
key.importer.keystore.passphrase=keystorepassphrase
key.importer.key.passphrase=keypassphrase
```

## 14.3 web.xml configuration reference

This section and its subsections do not apply to KeyMaster.NET.

This section contains the reference information for the *web.xml* configuration file. This configuration file defines the characteristics of the KeyMaster Signature Generator (the servlet that generates user credentials tokens). KeyMaster is shipped with a default *web.xml* file. You can edit this file and add to it, as required, to change certain aspects of the servlet's behavior.

The general format of the file is:

```
<web-app>

  <display-name>Caplin KeyMaster</display-name>
  <description>Caplin KeyMaster Servlet</description>

  <servlet>
    <servlet-name>StandardKeyMaster</servlet-name>
    <servlet-class>com. caplin. keymaster. servlet. StandardKeyMaster</servlet-class>

    <init-param>
      <param-name>name-of-a-configuration-parameter</param-name>
      <param-value>value-of-configuration-parameter</param-value>
      <description>Description of configuration parameter</description>
    </init-param>
    ...
    <init-param>
      <param-name>name-of-another-configuration-parameter</param-name>
      <param-value>value-of-another-configuration-parameter</param-value>
      <description>Description of another configuration parameter</description>
    </init-param>
    ...
  </servlet>

  <servlet>
    <servlet-name>XHRKeymaster</servlet-name>
    <servlet-class>com. caplin. keymaster. servlet. KeyMasterXHRFrame</servlet-class>

    <init-param>
      <param-name>name-of-a-configuration-parameter</param-name>
      <param-value>value-of-configuration-parameter</param-value>
      <description>Description of configuration parameter</description>
    </init-param>
    ...
  </servlet>

  <servlet>
    <servlet-name>Poll</servlet-name>
    <servlet-class>com. caplin. keymaster. servlet. Poll</servlet-class>
  </servlet>

  <servlet>
    <servlet-name>Dependencies</servlet-name>
    <servlet-class>com. caplin. keymaster. servlet. Dependencies</servlet-class>
  </servlet>

  <servlet>
    ...
  </servlet>
```

continued...

...continued

```
<servlet-mapping>
  <servlet-name>StandardKeyMaster</servlet-name>
  <url-pattern>/servlet/StandardKeyMaster</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>XHRKeymaster</servlet-name>
  <url-pattern>/servlet/XHRKeymaster</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>Poll</servlet-name>
  <url-pattern>/servlet/Poll</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>Dependencies</servlet-name>
  <url-pattern>/servlet/dependencies/*</url-pattern>
</servlet-mapping>

<servlet-mapping>
  ...
</servlet-mapping>

</web-app>
```

The *web.xml* file shipped with Standard KeyMaster contains definitions for four servlets, as highlighted in the previous file format example. The servlets are `StandardKeyMaster`, `XHRKeymaster`, `Poll`, and `Dependencies`, of which the last three are used to support accessing KeyMaster from StreamLink for Browsers. The `Poll` and `Dependencies` servlets do not require any parameters, so their `<servlet>` tags do not have any `<init-param>` child tags.

The next sections define the XML tags in alphabetical order.

Each section contains the following information:

- ◆ The tag name.
- ◆ A brief definition of what the tag does.
- ◆ A list of the immediate children of the tag, if any.
- ◆ More explanation of how to use the tag, if required.
- ◆ A code example showing the tag in use.

**Tip:** The items that you will most likely to want to change are the parameters defined in the child tags of `<init-param>`<sup>68</sup>. These are defined in the section on [web.xml parameters](#)<sup>73</sup>.

## <description>

**Tag:** <description>

This tag is used to provide a description of the contents of a [<display-name>](#)<sup>[67]</sup> or [<param-name>](#)<sup>[68]</sup> tag.

**Attributes:** None.

The tag has no child tags.

### Example 1:

```
<display-name>Caplin KeyMaster</display-name>
<description>
Caplin KeyMaster Servlet
</description>
```

### Example 2:

```
<init-param>
  <param-name>encrypting.generator.private.key.store.filename</param-name>
  <param-value>/Caplin/KeyMaster-4.4.0/privatekey.store</param-value>
  <description>File name and location for the private key</description>
</init-param>
```

## <display-name>

**Tag:** <display-name>

Defines the name of the Java web application.

**Note:** Do not modify this tag unless you are implementing a customized version of Caplin KeyMaster.

**Attributes:** None.

This tag has no child tags. It can be followed by a [<description>](#)<sup>[67]</sup> tag.

### Example:

```
<display-name>Caplin KeyMaster</display-name>
<description>
Caplin KeyMaster Servlet
</description>
```

## <init-param>

**Tag:** <init-param>

This tag defines a KeyMaster servlet configuration parameter.

**Attributes:** None.

The immediate children of the <init-param> tag are a [<param-name>](#) <sup>68</sup> tag, followed by a [<param-value>](#) <sup>69</sup> tag and an optional [<description>](#) <sup>67</sup> tag.

**Example:**

```
<init-param>
  <param-name>encrypting.generator.private.key.store.filename</param-name>
  <param-value>/Caplin/KeyMaster-4.4.0/privatekey.store</param-value>
  <description>File name and location for the private key</description>
</init-param>
```

## <param-name>

**Tag:** <param-name>

This tag defines the name of a KeyMaster servlet configuration parameter.

**Attributes:** None.

This tag has no child tags.

**Example:**

```
<init-param>
  <param-name>encrypting.generator.private.key.store.filename</param-name>
  <param-value>/Caplin/KeyMaster-4.4.0/privatekey.store</param-value>
  <description>File name and location for the private key</description>
</init-param>
```

The valid parameter names are defined in the [web.xml parameters](#) <sup>73</sup> section.



## <param-value>

**Tag:** <param-value>

This tag defines the value of the servlet configuration parameter whose name is defined in the associated [<param-name>](#) <sup>[68]</sup> tag.

**Attributes:** None.

This tag has no child tags.

**Example:**

```
<init-param>
  <param-name>encrypting.generator.private.key.store.filename</param-name>
  <param-value>/Caplin/KeyMaster-4.4.0/privatekey.store</param-value>
  <description>File name and location for the private key</description>
</init-param>
```

## <servlet>

**Tag:** <servlet>

Defines the configuration of a KeyMaster servlet.

**Attributes:** None.

The immediate children of the <servlet> tag are a [<servlet-name>](#) <sup>[71]</sup> and [<servlet-class>](#) <sup>[70]</sup> tag, followed by zero or more [<init-param>](#) <sup>[68]</sup> tags.

**Example:**

```
<servlet>
  <servlet-name>StandardKeyMaster</servlet-name>
  <servlet-class>com.caplin.keymaster.servlet.StandardKeyMaster</servlet-class>

  <init-param>
    <param-name>encrypting.generator.key.identifier</param-name>
    <param-value>keyidl</param-value>
    <description>The value that was passed as the second argument to the
      Key Generator when the key was created.
    </description>
  </init-param>

  </init-param>

  <init-param>
    <param-name>encrypting.generator.private.key.store.filename</param-name>
    <param-value>/Caplin/KeyMaster-4.4.0/privatekey.store</param-value>
    <description>File name and location for the private key</description>
  </init-param>

</servlet>
```

## <servlet-class>

**Tag:** <servlet-class>

This tag defines the name of the Java class that implements a servlet.

**Note:** Do not modify this tag unless you are implementing a customized version of Caplin KeyMaster.

**Attributes:** None.

The tag has no child tags.

**Example:**

```
<servlet>
  <servlet-name>StandardKeyMaster</servlet-name>
  <servlet-class>com.caplin.keymaster.servlet.StandardKeyMaster</servlet-class>
  ...
</servlet>
```

This tag is used to define the name of the Java class that implements the KeyMaster Signature Generator, as shown in the example above.

## <servlet-mapping>

**Tag:** <servlet-mapping>

This tag defines the mapping between a servlet and the URL that invokes it.

**Attributes:** None.

The immediate children of the <servlet-mapping> tag are a [<servlet-name>](#) followed by a [<url-pattern>](#) tag.

**Example:**

```
<servlet-mapping>
  <servlet-name>StandardKeyMaster</servlet-name>
  <url-pattern>/servlet/StandardKeyMaster</url-pattern>
</servlet-mapping>
```

## <servlet-name>

**Tag:** <servlet-name>

This tag defines the name of a servlet.

**Note:** Do not modify this tag unless you are implementing a customized version of Caplin KeyMaster.

**Attributes:** None.

The tag has no child tags.

**Example:**

```
<servlet>
  <servlet-name>StandardKeyMaster</servlet-name>
  <servlet-class>com.caplin.keymaster.servlet.StandardKeyMaster</servlet-class>
  ...
</servlet>
```

This tag is used to define the name of the KeyMaster Signature Generator, as shown in the example above.

It is also used in a [<servlet-mapping>](#) tag.

## <url-pattern>

**Tag:** <url-pattern>

This tag defines the URL that invokes the servlet named in the associated [<servlet-name>](#) tag.

**Note:** Do not modify the <url-pattern> tag whose associated <servlet-name> tag has the value StandardKeyMaster, unless you are implementing a customized version of Caplin KeyMaster.

**Attributes:** None.

The tag has no child tags.

**Example:**

```
<servlet-mapping>
  <servlet-name>StandardKeyMaster</servlet-name>
  <url-pattern>/servlet/StandardKeyMaster</url-pattern>
</servlet-mapping>
```

## <web-app>

**Tag:** <web-app>

The root tag for configuring the servlets of a Java web application.

**Attributes:** None.

The immediate children of the <web-app> tag are:

- ◆ a [<display-name>](#) tag, followed by
- ◆ an optional [<description>](#) tag, followed by
- ◆ one or more [<servlet>](#) tags, followed by
- ◆ one or more [<servlet-mapping>](#) tags.

There must be a separate <servlet-mapping> tag for each <servlet> tag.

For KeyMaster there only needs to be one <servlet> tag, to define the configuration of the Signature Generator, and a corresponding <servlet-mapping> tag.

**Example:**

```
<web-app>

  <display-name>Caplin KeyMaster</display-name>
  <description>Caplin KeyMaster Servlet</description>

  <servlet>
  ...
</servlet>

  <servlet-mapping>
  ...
</servlet-mapping>

</web-app>
```

## web.xml parameters

This section and its subsections do not apply to KeyMaster.NET.

The following sections define the valid parameters that can appear in the `<param-name>` tag (within an `<init-param>` tag) in the `web.xml` configuration file. The list of parameters below is in alphabetical order of parameter name.

- ◆ [encrypting.encode.extra.data](#)
- ◆ [encrypting.generator.hardware.key.passphrase](#)
- ◆ [encrypting.generator.hardware.keystore.keyfile](#)
- ◆ [encrypting.generator.hardware.keystore.passphrase](#)
- ◆ [encrypting.generator.hardware.keystore.type](#)
- ◆ [encrypting.generator.key.identifier](#)
- ◆ [encrypting.generator.keystore.type](#)
- ◆ [encrypting.generator.private.key.store.filename](#)
- ◆ [encrypting.generator.security.provider.class.name](#)
- ◆ [encrypting.generator.security.provider.name](#)
- ◆ [encrypting.generator.signature.algorithm](#)
- ◆ [extra.data.provider.classname](#)
- ◆ [formatter-type-{formatter\\_name}](#)
- ◆ [formatter-type-javascript](#)
- ◆ [formatter-type-streamlink](#)
- ◆ [http.remote.user](#)
- ◆ [keymaster.url](#)
- ◆ [keymaster.poll.url](#)
- ◆ [key.generator.FilenameAttribute](#)
- ◆ [key.generator.Level](#)
- ◆ [mapping.data.provider](#)
- ◆ [user.credential.provider](#)

## encrypting.encode.extra.data

**Parameter:** `encrypting.encode.extra.data`

**Required?** NO

**Description:**

In the standard (non customized) version of KeyMaster this *web.xml* parameter controls whether or not the user credentials token contains the user name in addition to the standard time stamp and sequence number.

If `encrypting.encode.extra.data` is set to `enabled`, and the parameters [extra.data.provider.classname](#)<sup>[81]</sup> and [mapping.data.provider](#)<sup>[87]</sup> are *not* defined, the user name is put in the token and is also included in the token's digital signature. If `encrypting.encode.extra.data` is not specified, or is set to any value other than `enabled`, the user name and any extra or mapping data are *not* put in the token.

You can customize KeyMaster so that the user credentials token contains other additional data, for example entitlement information related to the user name. To do this, set `encrypting.encode.extra.data` to `enabled`, and define the parameter [extra.data.provider.classname](#)<sup>[81]</sup>. In this case, the custom Java class defined by `extra.data.provider.classname` determines what additional data is put in the token; it is up to the custom class to include the user name if this is required.

You can customize KeyMaster so that the user credentials token contains mapping data from [mapping.data.provider](#)<sup>[87]</sup>. If `mapping.data.provider` is defined, the user name is also included in the token.

In the example *web.xml* file shipped with KeyMaster the `encrypting.encode.extra.data` parameter is set to `disabled`.

**Note:** The `encrypting.encode.extra.data` parameter only works with Caplin Liberator versions 3.6.7 and 4.0.1 or higher.

**Example:**

```
<param-name>encrypting.encode.extra.data</param-name>
<param-value>enabled</param-value>
```

Also see:

- ◆ The [http.remote.user](#)<sup>[83]</sup> parameter, which determines where the user name is obtained from.
- ◆ The [extra.data.provider.classname](#)<sup>[81]</sup> parameter, which defines a custom Java class that KeyMaster uses to add extra data to the user credentials token.
- ◆ The [mapping.data.provider](#)<sup>[87]</sup> parameter, which defines a custom Java class that KeyMaster uses to add mapping data to the user credentials token.

## encrypting.generator.hardware.key.passphrase

**Parameter:** `encrypting.generator.hardware.key.passphrase`

**Required?** NO

**Description:**

This *web.xml* parameter defines the passphrase that KeyMaster must use to retrieve the private encryption key from the hardware Key Store. This parameter must be included in *web.xml* if the private encryption key was protected by a passphrase when imported into the Key Store. It must match the value of the `key.importer.key.passphrase` property in *keyimporter.props* (see the [keyimporter.props configuration reference](#)<sup>[62]</sup> section and [Importing the private key file and certificate into the Key Store](#)<sup>[44]</sup>).

If this parameter is not present in *web.xml* then KeyMaster attempts to retrieve the private encryption key from the Key Store without supplying a passphrase.

**Example:**

```
<param-name>encrypting.generator.hardware.key.passphrase</param-name>
<param-value>
  mykeypassphrase
</param-value>
```

## encrypting.generator.hardware.keystore.keyfile

**Parameter:** `encrypting.generator.hardware.keystore.keyfile`

**Required?** YES if using a hardware Key Store, otherwise NO

**Description:**

The Key Importer tool generates a file that defines how KeyMaster can access the Key Store. This *web.xml* parameter defines the name and directory path of this file. It must have the same value as the *keyimporter.props* item `key.importer.keystore.location` (see [keyimporter.props configuration reference](#)<sup>[62]</sup>).

**Example:**

```
<param-name>encrypting.generator.hardware.keystore.keyfile</param-name>
<param-value>
  /opt/keystore.dat
</param-value>
```

Also see:

- ◆ [Importing the private key file and certificate into the Key Store](#)<sup>[44]</sup>.

## encrypting.generator.hardware.keystore.passphrase

**Parameter:** `encrypting.generator.hardware.keystore.passphrase`

**Required?** NO

**Description:**

This *web.xml* parameter defines the passphrase that KeyMaster must use to access the hardware Key Store. This parameter must be included in *web.xml* if the hardware Key Store is protected by a passphrase. It must match the value of the `key.importer.keystore.passphrase` property in *keyimporter.props* (see the [keyimporter.props configuration reference](#)<sup>[62]</sup> section and [Importing the private key file and certificate into the Key Store](#)<sup>[44]</sup>).

If this parameter is not present in *web.xml*, KeyMaster attempts to access the Key Store without supplying a passphrase.

**Example:**

```
<param-name>encrypting.generator.hardware.keystore.passphrase</param-name>
<param-value>
    keystorepassphrase
</param-value>
```

## encrypting.generator.hardware.keystore.type

**Parameter:** `encrypting.generator.hardware.keystore.type`

**Required?** YES if using a hardware Key Store, otherwise NO

**Description:**

This *web.xml* parameter defines the 'type' parameter to use when KeyMaster creates a Java **KeyStore** class to represent the hardware Key Store. It is the first argument of the **KeyStore.getInstance()** method. The value required is normally specified by the supplier of the Key Store hardware.

**Example:**

```
<param-name>encrypting.generator.hardware.keystore.type</param-name>
<param-value>
    ncipher.world
</param-value>
```



## encrypting.generator.key.identifier

**Parameter:** `encrypting.generator.key.identifier`

**Required?** YES

**Description:**

This *web.xml* parameter is the name of the key identifier that was passed as the second argument of the generator command when the key files were created. See [Generating the Required Keys](#)<sup>[10]</sup>.

**Example:**

```
<param-name>encrypting.generator.key.identifier</param-name>
<param-value>keyidl</param-value>
```

## encrypting.generator.keystore.type

**Parameter:** `encrypting.generator.keystore.type`

**Required?** NO

**Description:**

This *web.xml* parameter indicates whether KeyMaster should retrieve encryption keys from files held on normal disk or from a hardware Key Store. The accepted values for this parameter are:

- ◆ `standard`  
The encryption keys are held on disk.
- ◆ `hardware`  
The encryption keys are held in a hardware Key Store.

If this parameter is not present then KeyMaster assumes the default value `standard`.

**Example:**

```
<param-name>encrypting.generator.keystore.type</param-name>
<param-value>
  hardware
</param-value>
```

## encrypting.generator.private.key.store.filename

**Parameter:** `encrypting.generator.private.key.store.filename`

**Required?** NO if using a hardware Key Store, otherwise YES

**Description:**

This *web.xml* parameter is the full path to the private key store. Standard KeyMaster uses the private key store to generate user credentials tokens.

You can specify the path as a file path or as a Java classpath.

If you are using a hardware Key Store, this parameter can be omitted from *web.xml*, but if it is present then KeyMaster will just ignore it.

**Example file path specification (KeyMaster on Linux or Sun Solaris):**

```
<param-name>encrypting.generator.private.key.store.filename</param-name>
<param-value>
  /usr/local/jakarta-tomcat-5.0.16/webapps/keymaster/privatekey.store
</param-value>
```

**Example classpath specification (KeyMaster on Linux or Sun Solaris):**

```
<param-name>encrypting.generator.private.key.store.filename</param-name>
<param-value>
  classpath: com/caplin/keymaster/encrypted/privatekey.store
</param-value>
```

**Example file path specification (KeyMaster on Windows):**

```
<param-name>encrypting.generator.private.key.store.filename</param-name>
<param-value>C: /myKeyMaster/KeyMaster-4.4.0/privatekey.store</param-value>
```

If the file path is specified using the Windows backward slash notation, the slash characters must be paired ("\\"):

```
<param-name>encrypting.generator.private.key.store.filename</param-name>
<param-value>C: \\myKeyMaster\\KeyMaster-4.4.0\\privatekey.store</param-value>
```

## encrypting.generator.security.provider.class.name

**Parameter:** `encrypting.generator.security.provider.class.name`

**Required?** NO

### Description:

This *web.xml* parameter is the fully qualified name of the [JCE](#)<sup>[90]</sup> provider's Java class used to generate the encrypted portion of the user credentials token.

If this parameter is not set, the class defaults to the one used by the [SunRsaSign](#) provider.

If the parameter *is* set, the specified class is in addition to the default one. To use the new class in place of the default one, set the [encrypting.generator.security.provider.name](#)<sup>[80]</sup> parameter.

KeyMaster is shipped with public domain encryption software from The Legion Of The Bouncy Castle, and so in the example *web.xml* file shipped with KeyMaster, `encrypting.generator.security.provider.class.name` is set to point the Bouncy Castle encryption class (see the example below), and `encrypting.generator.security.provider.name` is set accordingly.

**Note:** Only change this setting if you have customized KeyMaster to use a different encryption class. If you do use a different encryption class, make sure that the class name is included in the classpath for the KeyMaster servlet.

### Example:

```
<param-name>encrypting.generator.security.provider.class.name</param-name>
<param-value>org.bouncycastle.jce.provider.BouncyCastleProvider</param-value>
```

Also see:

- ◆ The [key.generator.security.provider.class.name](#)<sup>[61]</sup> parameter of *keygen.props*.
- ◆ The [encrypting.generator.security.provider.name](#)<sup>[80]</sup> parameter.

## encrypting.generator.security.provider.name

**Parameter:** `encrypting.generator.security.provider.name`

**Required?** YES if [encrypting.generator.security.provider.class.name](#)<sup>[79]</sup> has been set.

### Description:

This *web.xml* parameter is the name of the provider of the Java class used to generate the encrypted portion of the user credentials token.

If this parameter is not set, the provider defaults to the [SunRsaSign](#) provider.

If the parameter is set (to a provider other than **SunRsaSign**), the corresponding encryption class name must be specified in the [encrypting.generator.security.provider.class.name](#)<sup>[80]</sup> parameter.

KeyMaster is shipped with public domain encryption software from The Legion Of The Bouncy Castle, and so in the example *web.xml* file shipped with KeyMaster, the `encrypting.generator.security.provider.name` parameter defines this particular provider (see the example below).

**Note:** Only change this setting if you have customized KeyMaster to use a different encryption provider.

### Example:

```
<param-name>encrypting.generator.security.provider.name</param-name>  
<param-value>BC</param-value>
```

Also see:

- ◆ The [encrypting.generator.security.provider.class.name](#)<sup>[79]</sup> parameter.

## encrypting.generator.signature.algorithm

**Parameter:** `encrypting.generator.signature.algorithm`

**Required?** NO

**Description:**

This *web.xml* parameter is the algorithm used to digitally sign KeyMaster user credentials tokens. Example values for this parameter are:

- ◆ MD5withRSA
- ◆ SHA256withRSA

If this parameter is not present then KeyMaster uses MD5withRSA as the default algorithm.

**Example:**

```
<param-name>encrypting.generator.signature.algorithm</param-name>
<param-value>
  SHA256withRSA
</param-value>
```

**Note: MD5 limitations:** Since KeyMaster was first released, the cryptographic community have found that the MD5 algorithm can produce hash collisions. This potentially compromises the algorithm.

Caplin has retained MD5withRSA as the default digital signature algorithm for backward compatibility with previous versions of KeyMaster. *However, customers installing KeyMaster for the first time may wish to configure the software to use a more secure algorithm, such as SHA256.*

## extra.data.provider.classname

**Parameter:** `extra.data.provider.classname`

**Required?** NO

**Description:**

This *web.xml* parameter is the name of a custom Java class that KeyMaster uses to add extra data to the user credentials token. If you define this parameter then you must also set the parameter [encrypting.encode.extra.data](#)<sup>74</sup> to enabled.

The custom Java class must implement the interface **com.caplin.keymaster.servlet.ExtraDataProvider**. The interface ensures that the extra data is also included in the digital signature.

If you require the user name to be included in the user credentials token you must specify this in your custom class.

**Example:**

```
<param-name>extra.data.provider.classname</param-name>
<param-value>com.caplin.ExampleDataProvider</param-value>
```

Also see:

- ◆ The [http.remote.user](#)<sup>[83]</sup> parameter, which determines where the user name is obtained from.
- ◆ The [encrypting.encode.extra.data](#)<sup>[74]</sup> parameter, which determines whether the user name or other additional data is put in the user credentials token.

## formatter-type-{formatter\_name}

**Parameter:** `formatter-type-{formatter_name}`

**Required?** NO

**Description:**

This *web.xml* parameter defines the name of a custom Java class that formats KeyMaster's response to a request for a user credentials token. {formatter\_name} uniquely identifies the formatter to be used; the name of the corresponding class is defined in the accompanying `<param-value>` tag (see the example below).

KeyMaster is shipped with two standard response formatters that are defined in the example *web.xml* file (see parameters [formatter-type-javascript](#)<sup>[82]</sup> and [formatter-type-streamlink](#)<sup>[83]</sup>). You only need to add additional `formatter-type-{formatter_name}` parameters if you are customizing KeyMaster with additional response formatter classes.

**Example:**

```
<param-name>formatter-type-news</param-name>
<param-value>examples.news.NewsFormatter</param-value>
<description>Name of the class to handle a NewsFormatter response</description>
```

## formatter-type-javascript

**Parameter:** `formatter-type-javascript`

**Required?** YES for Standard KeyMaster

**Description:**

This *web.xml* parameter defines the name of the Java class that formats KeyMaster's response to a request from a JavaScript application for a user credentials token. This class is shipped with KeyMaster and the corresponding `formatter-type-javascript` parameter is defined in the example *web.xml* file.

**Note:** Only modify the value of this parameter if KeyMaster is being customized to use a different formatter class for handling responses to Javascript requests.

**Example:**

```
<param-name>formatter-type-javascript</param-name>
<param-value>com.caplin.keymaster.servlet.</param-value>
<description>Name of the class to handle a JavaScript response</description>
```

## formatter-type-streamlink

**Parameter:** `formatter-type-streamlink`

**Required?** YES for Standard KeyMaster

**Description:**

This *web.xml* parameter defines the name of the Java class that formats KeyMaster's response to a request from a Caplin StreamLink application for a user credentials token. This class is shipped with KeyMaster and the corresponding `formatter-type-streamlink` parameter is defined in the example *web.xml* file.

**Note:** Only modify the value of this parameter if KeyMaster is being customized to use a different formatter class for handling responses to StreamLink requests.

**Example:**

```
<param-name>formatter-type-streamlink</param-name>
<param-value>com.caplin.keymaster.servlet.StreamLinkFormatter</param-value>
<description>Name of the class to handle a StreamLink response</description>
```

## http.remote.user

**Parameter:** `http.remote.user`

**Required?** NO

**Description:**

When an end user's client application requests a user credentials token, KeyMaster's Signature Generator will by default obtain the end user's user name from an HTTP request parameter sent by the application (`?username=...`). However, if the end user has logged on to the application using a single sign-on system, the single sign-on system may be able to transmit the user name in the REMOTE\_USER attribute of the HTTP header.

Setting the *web.xml* parameter `http.remote.user` to `enabled` causes KeyMaster to obtain the user name from the HTTP header, instead of from the HTTP request parameter. However, if `http.remote.user` is `enabled` but the REMOTE\_USER is null, then KeyMaster reverts to obtaining the user name from the `username` parameter in the HTTP request.

If the `http.remote.user` parameter is not specified, or is set to any value other than `enabled`, KeyMaster obtains the user name from the from the HTTP request parameter `username`.

In the example *web.xml* file shipped with KeyMaster, `http.remote.user` is set to `disabled`.

**Note:** Only set `http.remote.user` to `enabled` if your single sign-on system supports transmitting the user name in the REMOTE\_USER attribute of the HTTP header.

**Example:**

```
<param-name>http.remote.user</param-name>
<param-value>enabled</param-value>
```

Also see:

- ◆ The [user.credential.provider](#) <sup>[88]</sup> parameter, which defines the Java class that determines how the user name is retrieved.
- ◆ The [encrypting.encode.extra.data](#) <sup>[74]</sup> parameter, which determines whether the user name or other additional data is put in the user credentials token.
- ◆ The [extra.data.provider.classname](#) <sup>[81]</sup> parameter, which defines a custom Java class that KeyMaster uses to add extra data to the user credentials token.

## keymaster.url

**Parameter:** keymaster.url

**Required?** NO

**Description:**

If you are deploying a customized version of KeyMaster that uses a different `<servlet-mapping>` for the `StandardKeyMaster` servlet, then you also need to specify the url of the `StandardKeyMaster` servlet here, so that the `XHRKeymaster` servlet can access it.

**Note:** This parameter should only be specified in the `<servlet>` definition for `XHRKeymaster`, and must not be used in the definitions for any of the other KeyMaster servlets.

**Example:**

```
<servlet>
  <servlet-name>XHRKeymaster</servlet-name>
  <init-param>
    <param-name>keymaster.url</param-name>
    <param-value>/servlet/CustomizedKeyMasterName</param-value>
  </init-param>
  ...
</servlet>
```

where the `<servlet-mapping>` for the `StandardKeyMaster` servlet has been changed to:

```
<servlet-mapping>
  <servlet-name>StandardKeyMaster</servlet-name>
  <url-pattern>/servlet/CustomizedKeyMasterName</url-pattern>
</servlet-mapping>
```

Also see:

- ◆ [Changing KeyMaster's URL](#) <sup>[20]</sup>



## keymaster.poll.url

**Parameter:** `keymaster.poll.url`

**Required?** NO

**Description:**

If you are deploying a customized version of KeyMaster that uses a different `<servlet-mapping>` for the `Poll` servlet, then you also need to specify the url of the `Poll` servlet here, so that the `XHRKeymaster` servlet can access it.

**Note:** This parameter should only be specified in the `<servlet>` definition for `XHRKeymaster`, and must not be used in the definitions for any of the other KeyMaster servlets.

**Example:**

```
<servlet>
  <servlet-name>XHRKeymaster</servlet-name>
  <init-param>
    <param-name>keymaster.poll.url</param-name>
    <param-value>/servlet/newPollLocation</param-value>
  </init-param>
  ...
</servlet>
```

where the `<servlet-mapping>` for the `Poll` servlet has been changed to:

```
<servlet-mapping>
  <servlet-name>Poll</servlet-name>
  <url-pattern>/servlet/newPollLocation</url-pattern>
</servlet-mapping>
```

Also see:

◆ [Changing the KeyMaster Poll servlet's URL](#) 

## key.generator.FilenameAttribute

**Parameter:** `key.generator.FilenameAttribute`

**Required?** NO

**Description:**

This *web.xml* parameter defines the name and location of the KeyMaster Signature Generator's log file. This can be a full path name or a relative path. If a relative path name is used, then it will normally be relative to the application server's root directory, though this may not be the case for some application servers.

If this parameter is not specified, KeyMaster will by default create a log file called *keymaster.log* in the application server's root directory, though once again, this may not be the case for some application servers. In the example *web.xml* file shipped with KeyMaster, the log file name is set to *servlet.log* (see the example below).

**Example:**

```
<param-name>key.generator.FilenameAttribute</param-name>  
<param-value>servlet.log</param-value>
```

Also see:

- ◆ The [key.generator.Level](#) <sup>86</sup> parameter.

## key.generator.Level

**Parameter:** key.generator.Level

**Required?** NO

**Description:**

This *web.xml* parameter defines the Java logging level used to output information to the KeyMaster log file about what is happening within KeyMaster. If this parameter is not specified, KeyMaster will by default set the logging level to SEVERE, so that only the most serious problems will be logged.

In the example *web.xml* file shipped with KeyMaster, the logging level is set to set to ALL (see the example below), which provides a very detailed level of logging for debug purposes.

**Note:** In a production system it is recommended that the logging level normally be set to SEVERE or WARNING.

**Tip:** The possible logging levels are defined in the standard Java documentation under **java.util.logging.Level**.

**Example:**

```
<param-name>key.generator.Level</param-name>  
<param-value>ALL</param-value>
```

Also see:

- ◆ The [key.generator.FilenameAttribute](#) <sup>85</sup> parameter.

## mapping.data.provider

**Parameter:** `mapping.data.provider`

**Required?** NO

**Description:**

This *web.xml* parameter specifies the class that obtains the mapping data to be inserted in the KeyMaster token. The class must implement the `com.caplin.keymaster.servlet.MappingDataProvider` interface, which has one method `Map<String, String> getMappingData(HttpServletRequest httpRequest)`. If you use a **MappingDataProvider**, then you must also set the parameter `encrypting.encode.extra.data` to enabled.

If this parameter is not specified, no mapping data will be added to the token.

**Example:**

```
<param-name>mapping.data.provider</param-name>
<param-value>example.MyMappingDataProvider</param-value>
```

Also see:

- ◆ The [encrypting.encode.extra.data](#) <sup>74</sup> parameter, which must be enabled to add mapping data.

## user.credential.provider

**Parameter:** `user.credential.provider`

**Required?** NO

**Description:**

This *web.xml* parameter specifies the KeyMaster class that obtains the user name to be inserted in the user credentials token.

If this parameter is not specified, KeyMaster uses the class **com.caplin.keymaster.servlet.UserCredentialsProvider**, which obtains the user name according to the setting of the parameter [http.remote.user](#)<sup>[83]</sup>.

In the example *web.xml* file shipped with KeyMaster, `user.credential.provider` is set to the default class **com.caplin.keymaster.servlet.UserCredentialsProvider**.

**Note:** Only change this setting if you have customized KeyMaster to use a different class for obtaining the user name.

**Example:**

```
<param-name>user.credential.provider</param-name>
<param-value>com.caplin.keymaster.servlet.UserCredentialsProvider</param-value>
```

Also see:

- ◆ The [http.remote.user](#)<sup>[83]</sup> parameter, which determines where the user name is obtained from.

## 15 Glossary of Terms and Acronyms

This section contains a glossary of terms, abbreviations, and acronyms relating to the KeyMaster product.

Term	Definition
<b>Ajax</b>	<u>A</u> syncronous <u>J</u> avaScript and <u>X</u> ML A combination of Web technologies used to implement interactive Web clients
<b>ASP</b>	<u>A</u> ctive <u>S</u> erver <u>P</u> ages. A technology from Microsoft that dynamically generates web pages using server-side scripts. Also known as “Classic ASP”. Also see <b>ASP.NET</b> .
<b>ASP.NET</b>	A newer version of Microsoft's Active Server Pages ( <b>ASP</b> ) that dynamically generates web pages using <b>.NET</b> technology.
<b>Authentication</b>	In the context of KeyMaster, authentication is the process of identifying a user, for example by checking a user name and password that the user supplied when attempting to log in. Authentication must precede <b>authorization</b> .
<b>Auth Module</b>	A Caplin software module that performs <b>authentication</b> and <b>authorization</b> functions. <b>Caplin Liberator</b> uses Auth Modules to authenticate users who log in to the Liberator, and to determine the users' access permissions to Liberator objects. See also <b>javaauth</b> and <b>XMLauth</b> .
<b>Authorization</b>	In the context of KeyMaster, authorization is the process of determining the access rights that a user has to resources, such as data and functionality provided by computer software. Users cannot be authorized until they have been successfully authenticated – see <b>authentication</b> .
<b>Caplin Liberator</b>	Caplin Liberator is a real-time financial internet hub that delivers trade messages and market data to and from subscribers over any network.
<b>Caplin Trader</b>	A web application framework for constructing browser-based financial trading applications.
<b>DER</b>	<u>D</u> istinguished <u>E</u> ncoding <u>R</u> ules. Rules for encoding ASN.1 objects in binary format which define just one way to represent any ASN.1 value. DER encoding is typically used when the same object is encoded in ASN.1 format multiple times for digital signature verification.
<b>DER public key file</b>	In KeyMaster this is a file containing a KeyMaster public key in <b>DER</b> format. This file is used by <b>Caplin Liberator</b> to authenticate the <b>user credentials token</b> sent by a user application that wishes to log in to the Liberator.

Term	Definition
Digital signature	<p>An electronic signature that is used to authenticate the sender of a message or author of a document. The signature is usually encrypted in some manner (see <b>public key encryption</b>).</p> <p>KeyMaster inserts a digital signature in the <b>user credentials tokens</b> that it generates.</p>
javaauth	<p>An <b>Auth Module</b> in which the <b>authentication</b> and <b>authorization</b> rules are specified using Java code.</p>
JCE	<p><u>J</u>ava <u>C</u>ryptography <u>E</u>xtension</p> <p>A Java package that provides a framework for and implementations of encryption, key generation and key agreement, and Message Authentication Code (MAC) algorithms.</p> <p>For more information see Sun's <a href="#">Java Cryptography Extension Reference Guide</a>.</p>
Key Store	<p>A hardware repository that holds encryption keys and X.509 certificates.</p>
MD5withRSA	<p>A digital signature algorithm. See the section "Digital signature algorithms" in the <b>KeyMaster Overview</b>.</p>
.NET	<p>A Microsoft framework for developing distributed applications that run under Microsoft Windows® operating systems and can easily intercommunicate with applications and systems running on different operating systems.</p>
OpenSSL	<p>An open source implementation of the SSL (<u>S</u>ecure <u>S</u>ockets <u>L</u>ayer) and TLS (<u>T</u>ransport <u>L</u>ayer <u>S</u>ecurity) protocols. In KeyMaster, the basic OpenSSL cryptographic functions are used generate to RSA keys and certificates for storage in a hardware <b>Key Store</b>.</p> <p>See <a href="http://www.openssl.org">www.openssl.org</a>.</p>
Public key encryption	<p>A method of sending encrypted information between two parties without the need for them to exchange a key for encrypting and decrypting the information. Rather than using a single key it uses two related keys – a public key and a private key. (See the note on public key cryptography and digital signatures in the <b>KeyMaster Overview</b>.)</p>
RTTP	<p><u>R</u>ea<u>L</u> <u>T</u>ime <u>T</u>ext <u>P</u>rotocol</p> <p>Caplin's protocol for streaming real-time financial data from <b>Caplin Liberator</b> servers to client applications, and for transmitting trade messages between clients and Liberator in both directions.</p>
RTSL	<p><u>R</u>ea<u>L</u> <u>T</u>ime <u>S</u>cripting <u>L</u>ayer</p> <p>A functional interface that can be used from any JavaScript-type language within a browser to create and manage <b>RTTP</b> connections and access streaming data.</p>
SDK	<p><u>S</u>oftware <u>D</u>evelopment <u>K</u>it</p>
SHA256withRSA	<p>A digital signature algorithm. See the section "Digital signature algorithms" in the <b>KeyMaster Overview</b>.</p>
Single sign-on	<p>A user authentication process in which a user supplies just one set of <b>user credentials</b> (such as a user name and password). The user can then access multiple applications and systems without being prompted for credentials again.</p>

Term	Definition
<b>StreamLink</b>	The StreamLink libraries connect client applications to <b>Caplin Liberator</b> via the <b>RTTP</b> protocol. They provide an object oriented API, on top of RTTP, that provides access to RTTP functionality.
<b>StreamLink for Browsers</b>	StreamLink for Browsers is a JavaScript implementation of <b>StreamLink</b> that runs in Web browsers. It allows Ajax applications to communicate with <b>Caplin Liberator</b> .
<b>StreamLink for Java</b>	StreamLink for Java is a Java implementation of <b>StreamLink</b> . It allows Java applications to communicate with <b>Caplin Liberator</b> .
<b>Trading Adapter</b>	An application that uses the Trading Integration API to integrate with a trading system.
<b>User credentials</b>	Information used to authenticate a user; for example a user name and password.
<b>User credentials token</b>	A data structure, containing <b>user credentials</b> , that is passed from one application to another in order to authenticate the user.
<b>XMLauth</b>	An <b>Auth Module</b> in which the <b>authentication</b> and <b>authorization</b> rules are specified in XML format.

## Contact Us

Caplin Systems Ltd  
Cutlers Court  
115 Houndsditch  
London EC3A 7BR  
Telephone: +44 20 7826 9600  
[www.caplin.com](http://www.caplin.com)

The information contained in this publication is subject to UK, US and international copyright laws and treaties and all rights are reserved. No part of this publication may be reproduced or transmitted in any form or by any means without the written authorization of an Officer of Caplin Systems Limited.

Various Caplin technologies described in this document are the subject of patent applications. All trademarks, company names, logos and service marks/names ("Marks") displayed in this publication are the property of Caplin or other third parties and may be registered trademarks. You are not permitted to use any Mark without the prior written consent of Caplin or the owner of that Mark.

This publication is provided "as is" without warranty of any kind, either express or implied, including, but not limited to, warranties of merchantability, fitness for a particular purpose, or non-infringement.

This publication could include technical inaccuracies or typographical errors and is subject to change without notice. Changes are periodically added to the information herein; these changes will be incorporated in new editions of this publication.

Caplin Systems Limited may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time.

This publication may contain links to third-party web sites; Caplin Systems Limited is not responsible for the content of such sites.